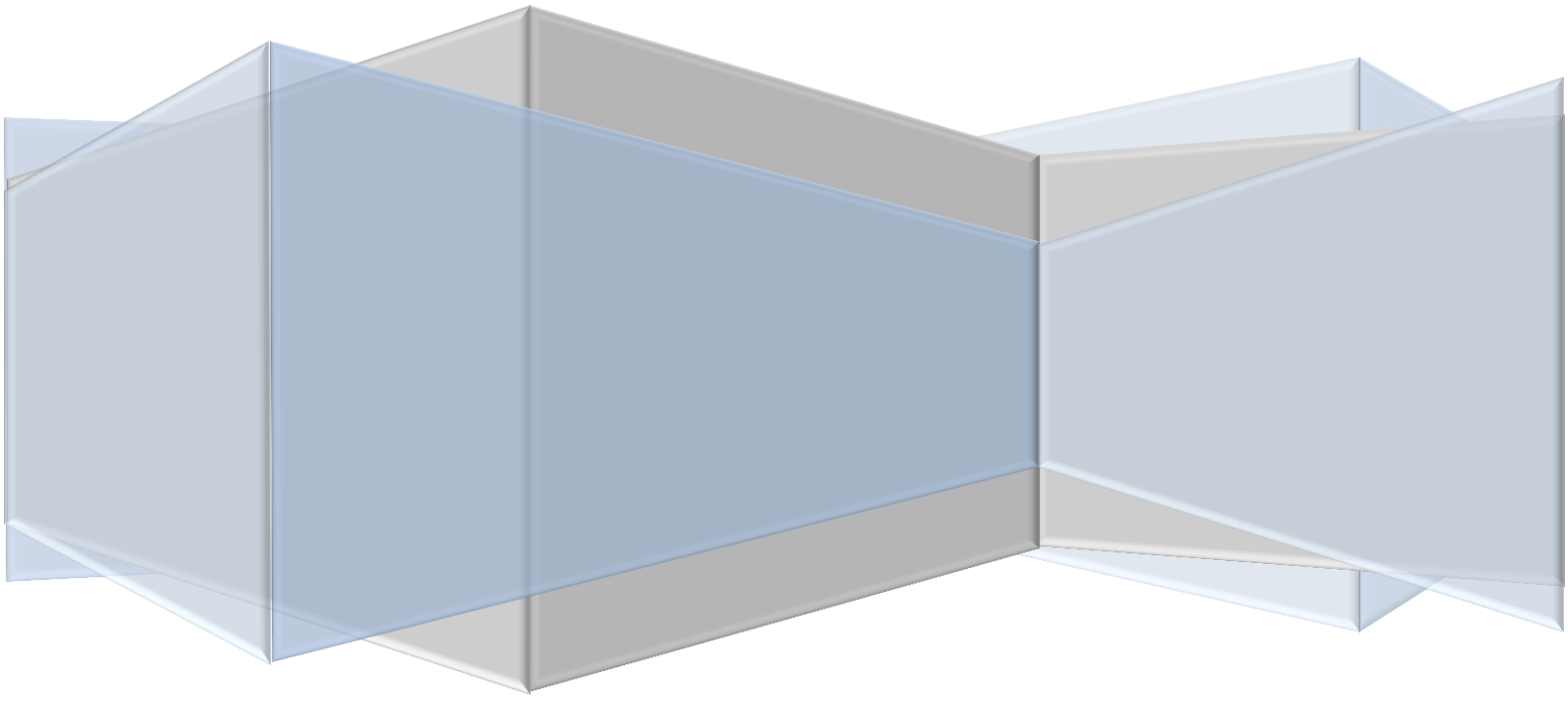
Università degli Studi di Padova

Interfaccia User-Friendly per la gestione di Firewall Linux

FIREWALL CREATOR

Daniele De Marco – Workteam SRL



Is studying computer science the best way to prepare to be a programmer? No. the best way to prepare is to write programs, and to study great programs that other people have written. In my case, I went to the garbage cans at the Computer Science Center and I fished out listings of their operating system. You got to be willing to read other people's code, then write your own, then have other people review your code. You've got to want to be in this incredible feedback loop where you get the world-class people to tell you what you're doing wrong.

Bill Gates

SOMMARIO

INTRODUZIONE

Firewall Creator nasce da un'idea di Enrico Conte, direttore dell'azienda veneta di San Donà di Piave *Workteam*. L'idea è di realizzare un software su piattaforma Windows che permetta di creare e gestire in maniera facile ed intuitiva dei Firewall Linux Iptables, in modo da poterne poi fare il deploy direttamente dal programma stesso.

L'applicazione è stata realizzata da Daniele De Marco, studente di Ingegneria Informatica presso l'*Università di Padova*. La scelta del linguaggio e dell'ambiente di sviluppo è caduta su Visual C# e Visual Studio 2010 Ultimate. Dato che l'applicazione, in parte di tipo client/server, prevede un piccolo demone da far girare su macchine Linux, si è brillantemente fatto uso di un nuovo software completamente gratuito e Open Source. Mono è infatti un IDE (*Integrated Development Environment*) di sviluppo che simula l'ambiente .NET C# su distribuzioni Linux, integrandosi opzionalmente con Visual Studio. Questo ha permesso di utilizzare un solo linguaggio e un solo ambiente di sviluppo per due diverse piattaforme, Linux e Windows.

Si è deciso di utilizzare XML (*Extensible Markup Language*) per la creazione della struttura dei Firewall e XSLT (*Extensible Stylesheet Language Transformation*) per la traduzione dello scheletro del Firewall in vero e proprio linguaggio Iptables Linux. All'interno del programma infatti ogni singolo pezzo del Firewal, sia esso un'interfaccia di rete, un comando o una regola, è rappresentato da un singolo nodo XML. Ogni nodo è a sua volta rappresentato dal suo tipo (Interface, Comments, NAT, Services...) e dai suoi attributi (IP, Policy, Table...). Il foglio di stile XSLT contiene decine di template, uno per ogni tipo di regola, e si occupa runtime di trasformare queste regole in linguaggio comprensibile da una macchina Linux. Il programma, grazie all'incredibile potenza offerta dal connubio di XML e XSLT, offre la possibilità di generare migliaia di regole Iptables diverse.

L’intera applicazione, sia la progettazione che il programma vero e proprio, è stata scritta in lingua inglese. I motivi sono assai scontati, il mondo della tecnologia informatica segue la filosofia anglosassone e sarebbe inutile forzare l’utilizzo dell’italiano in un panorama internazionale. All’interno di questa tesi numerosi termini tecnici sono stati scritti volutamente in lingua originale, una traduzione non avrebbe avuto senso e avrebbe reso probabilmente incomprensibili alcune funzionalità. Come conseguenza anche l’interfaccia del programma è, ovviamente, in lingua inglese.

La tesi si articola in due capitoli principali, il primo analizza una breve storia dei Firewall e la struttura attuale dei moderni sistemi di sicurezza Linux, permettendo così di comprendere a fondo e in maniera tecnica i motivi che hanno portato alla realizzazione di Firewall Creator. Nel secondo capitolo viene illustrato il programma vero e proprio, le tecnologie utilizzate e le sue funzionalità principali oltre ovviamente al codice sorgente.

E’ importante considerare che Firewall Creator è un software vero e proprio, con centinaia e centinaia di funzionalità, ed è difficile esaurire in una tesi tutti i suoi aspetti e le sue caratteristiche. Bisogna pensare a questo programma come un vero e proprio IDE di sviluppo che, in quanto tale, offre strumenti assai avanzati ma, allo stesso tempo, ovvi. Per capirci non si perderà tempo ad illustrare le funzionalità più banali come il copia-incolla o l’annulla-ripeti. Firewall Creator supporta tutto quello che potreste fare in un normale ambiente di sviluppo ma per i particolari si rimanda ad una guida in linea piuttosto che ad una tesi. Lo scopo di questo documento è capire quali sono i problemi di un sistemista e come Firewall Creator viene incontro per risolverli.

PERCHE' FIREWALL CREATOR

Nel processo di informatizzazione aziendale ricopre sempre più importanza il ruolo della sicurezza. Grazie alle moderne tecnologie infatti, gran parte dei servizi e delle operazioni vengono gestite da computer che si scambiano informazioni su reti LAN e Internet. Tali informazioni, è inutile dirlo, sono molto spesso sensibili e non dovrebbero per nessun motivo finire nelle mani sbagliate. Internet è una risorsa immensa ma allo stesso tempo molto pericolosa per chi non sa come muoversi. I Firewall sono il primo strumento di difesa per chi gestisce sistemi informativi. Tali "muri di fuoco" possono fare la differenza tra il fallimento e il successo. Può sembrare un'esagerazione ma non lo è, anzi. Pensate alle miriadi di dati che ogni giorno le banche si scambiano grazie a Internet. Se non ci fossero Firewall a proteggere il traffico e i dati salvati sui server, con molta probabilità le persone preferirebbero scambiarsi il denaro a mano piuttosto che rischiare di perderlo.

Grazie alla tecnologia fornita dai sistemi operativi Open Source Linux è possibile implementare Firewall di complessità e sicurezza relativamente alle effettive esigenze dell'azienda. A questo punto però ci si pone un'ulteriore problema, ovvero creare tali Firewall.

Realizzare, ma soprattutto gestire quotidianamente una rete e la sua sicurezza, non è affatto facile, nemmeno per un sistemista esperto.

Con Firewall Creator l'obiettivo principale è quello di offrire un'interfaccia user-friendly che permetta una gestione semplice, veloce ed efficace dei Firewall Linux.

Tramite il supporto per l'installazione remota e portable deve essere possibile creare e gestire il Firewall anche se non ci si trova sul posto di lavoro e persino se non è disponibile una connessione diretta con la macchina Firewall.

Inoltre la struttura basata su XML vuole rendere il programma non solo efficiente, ma anche adatto alla enorme velocità con cui il mondo IT si evolve. Gli aggiornamenti possono e devono essere implementati in maniera semplice e veloce direttamente dal sistemista, senza che esso debba contattare di volta in volta l'azienda che ha fornito il software, risparmiando non solo denaro, ma anche tempo.

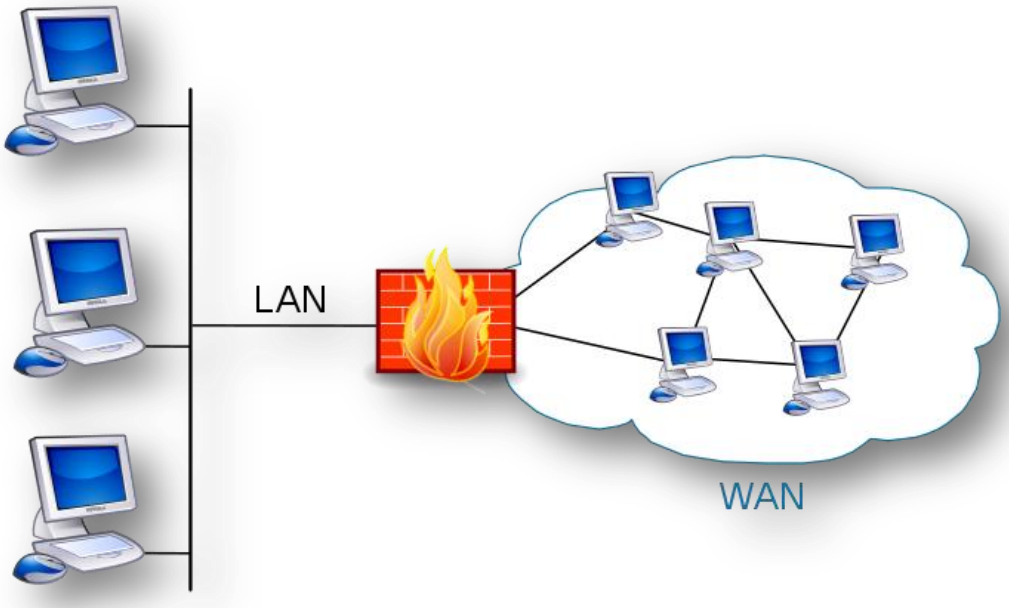
OBIETTIVI

- Interfaccia user-friendly
- Sicurezza relativamente alle esigenze dell'azienda
- Dinamicità del programma e dei Firewall stessi
- Gestione e modifica delle regole veloce e intuitiva
- Installazione remota e non
- Visione d'insieme e nel dettaglio allo stesso tempo

FIREWALL

Prima di entrare nei dettagli del progetto e della sua realizzazione è opportuno comprendere attentamente il concetto e il vero significato di un Firewall. Si andrà ad analizzare una breve storia e i motivi che hanno portato i sistemisti a implementare Firewall sempre più moderni ed efficaci. Infine si illusterà il funzionamento dei moderni software Firewall Linux e in particolare le tecnologie Netfilter ed Iptables.

A firewall is a device or set of devices designed to permit or deny network transmissions based upon a set of rules and is frequently used to protect networks from unauthorized access while permitting legitimate communications to pass. (Wikipedia, The Free Encyclopedia)



BREVE STORIA

Con l'avvento di Internet negli anni '80 molte aziende si trovarano a dover affrontare un problema assai importante. Se da un lato Internet e l'informatizzzazione sembravano un sogno, dall'altro celavano un vero e proprio incubo. Tutte le grandi aziende infatti possiedono un'enorme quantità di informazioni online, molte delle quali non dovrebbero essere in alcun modo divulgate, se non a particolari categorie di persone. Era necessario quindi un sistema in grado di "filtrare" le informazioni per garantire la sicurezza dei dati sensibili.

Quando nel 1988 diverse università californiane, come Berkeley e la UC San Diego, vennero attaccate da un Virus informatico, nessuno era preparato a un tale evento, del resto nessuno nemmeno se lo aspettava. Per fortuna il primo Virus non aveva fini malevoli ma era chiaro che l'infrastruttura delle reti era troppo vulnerabile e bisognava agire in tal senso.

Nello stesso anno alcuni ingegneri della DEC (*Digital Equipment Corporation*) si misero al lavoro per realizzare il primo vero e proprio Firewall basato sulla tecnologia *packet filter*. Come dice il nome stesso l'idea di base è quella di ispezionare ogni singolo pacchetto e di filtrarlo in base a un insieme di regole definito a priori dal sistemista. Tale sistema funziona sui layer più bassi del modello ISO/OSI, il filtraggio avviene infatti a livello network, fisico e , in parte, TCP.

Nel corso degli anni i Firewall si sono evoluti velocemente andando a coinvolgere anche gli strati più alti del modello ISO/OSI; gli *application layer firewall* infatti sono in grado di comprendere applicazioni e protocolli e, lavorando su tutti e sette i layer, sono di conseguenza più sicuri del packet filter.

In ultimo troviamo gli *stateful firewall*, anche qui dal nome è facile individuare il loro funzionamento, grazie al concetto di "stato" è possibile catalogare una serie di pacchetti in base alla loro storia. Tali Firewall sono però molto sensibili agli attacchi DoS (*Denial of Service*) dove un intruso può mandare in crash un server aprendo migliaia di connessioni contemporaneamente.

NETFILTER

Tra i vari componenti del Kernel Linux uno dei più importanti è sicuramente netfilter. Tale componente permette il filtraggio e l'intercettazione dei pacchetti che attraversano un dispositivo connesso alla rete. Le funzionalità offerte dal framework netfilter sono pressochè infinite, è possibile infatti realizzare Firewall di complessità elevata e con funzionalità aggiuntive come il NAT (*Network Address Translation*), ovvero la traduzione di indirizzi di rete e porte. Esattamente come Linux rispecchia la filosofia Free Software e come tale è rilasciato sotto licenza GNU General Public License. Netfilter è un componente standard delle distribuzioni Linux (2.4.x e 2.6.x) ed è stato introdotto per la prima volta nel 2000 anche se la prima implementazione risale al 1998 graze all'idea di Rusty Russel, un programmatore australiano.

IPTABLES

E' facile fare confusione tra netfilter e Iptables ed è opportuno precisarne la differenza. Netfilter non è altro che un'infrastruttura, una vera e propria API che il kernel Linux offre alle applicazioni che vogliono manipolare i pacchetti della rete. Iptables invece è un'interfaccia che, utilizzando netfilter, permette di interagire davvero sui pacchetti. Iptables è solo uno dei vari moduli che compogono netfilter ed è il modulo che verrà utilizzato da Firewall Creator.

SINOSSI

iptables [-t table] {-A|-D} chain rule-specification [options...]
iptables [-t table] -I [rulenum] rule-specification [options...]
iptables [-t table] -R rulenum rule-specification [options...]
iptables [-t table] -D chain rulenum [options...]
iptables [-t table] -S [chain]
iptables [-t table] {-F|-L|-Z} [chain] [options...]
iptables [-t table] -N chain
iptables [-t table] -X [chain]
iptables [-t table] -P chain target [options...]
iptables [-t table] -E old-chain-name new-chain-name

SINTASSI E FUNZIONAMENTO

Netfilter e Iptables operano sui pacchetti in transito sulla rete. Tramite le regole definite a priori nel Firewall è possibile eseguire delle operazioni su tali pacchetti in modo per esempio da accettarli, scartarli o manipolarli. Tutte le regole del Firewall vengono raggruppate in catene (*chain*) a loro volta raggruppate in tabelle (*tables*). Le catene possono essere create anche dagli utenti stessi tramite una semplice regola:

iptables [-t table] –N CHAIN_NAME

Quando un pacchetto attraversa una catena esamina immediatamente la sua prima regola, se la regola è soddisfatta allora esegue le azioni definite, altrimenti prosegue nella catena con la regola successiva. Quando giunge al termine della catena viene eseguita la sua regola di default che, in generale, è di tipo DROP, il che significa che il pacchetto verrà gettato via.

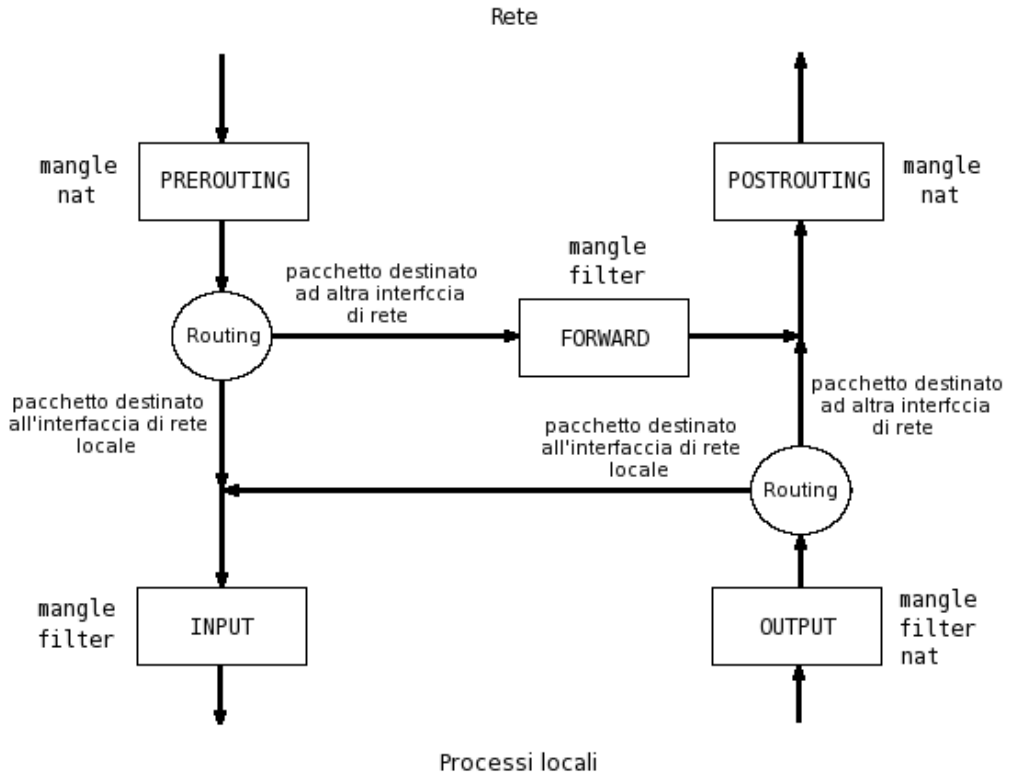
Di default Iptables offre quattro tipi diversi di tabelle, ognuna contenente delle catene predefinite:

- *filters* (INPUT, FORWARD, OUTPUT) – E' la tabella di default, viena automaticamente selezionata se non viene fornito nessun parametro –t, permette il filtraggio vero e proprio dei pacchetti ovvero permette di bloccarli o di lasciarli passare.
- *nat* (PREROUTING, OUTPUT, POSTROUTING) – E' la tabella responsabile per la modifica degli indirizzi e delle porte dei pacchetti. Quando viene instaurata una nuova connessione il primo pacchetto passa attraverso questa tabella.
- *mangle* (PREROUTING, INPUT, OUTPUT, FORWARD, POSTROUTING) – E' la tabella adibita all'alterazione dei pacchetti in transito e alle loro opzioni, un esempio famoso è il controllo del QoS (*Quality of Service*).
- *raw* (PREROUTING, OUTPUT) – Introdotta solo di recente permette di controllare il connection tracking per quei pacchetti che non si vogliono inoltrare in modo statefull.

Le catene elencate sopra indicano nient'altro che i flussi della rete:

- INPUT – Pacchetti in entrata destinati a processi in esecuzione all'interno del Firewall stesso.
- OUTPUT – Pacchetti in uscita generati da processi in esecuzione all'interno del Firewall stesso.
- FORWARD – Pacchetti in transito, non sono generati o destinati al Firewall stesso.

- PREROUTING – Permette di effettuare il Destination NAT (DNAT), ovvero cambiare l'indirizzo di destinazione del pacchetto non appena entra nel Firewall.
- POSTROUTING – Permette di effettuare il Source NAT (SNAT), ovvero cambiare il campo source di un pacchetto in uscita dal Firewall. Solitamente tale campo viene sostituito con l'indirizzo IP del Firewall stesso.
- OUTPUT (*nat*) – Come il POSTROUTING ma limitatamente ai pacchetti che vengono generati dai processi in esecuzione all'interno del Firewall.



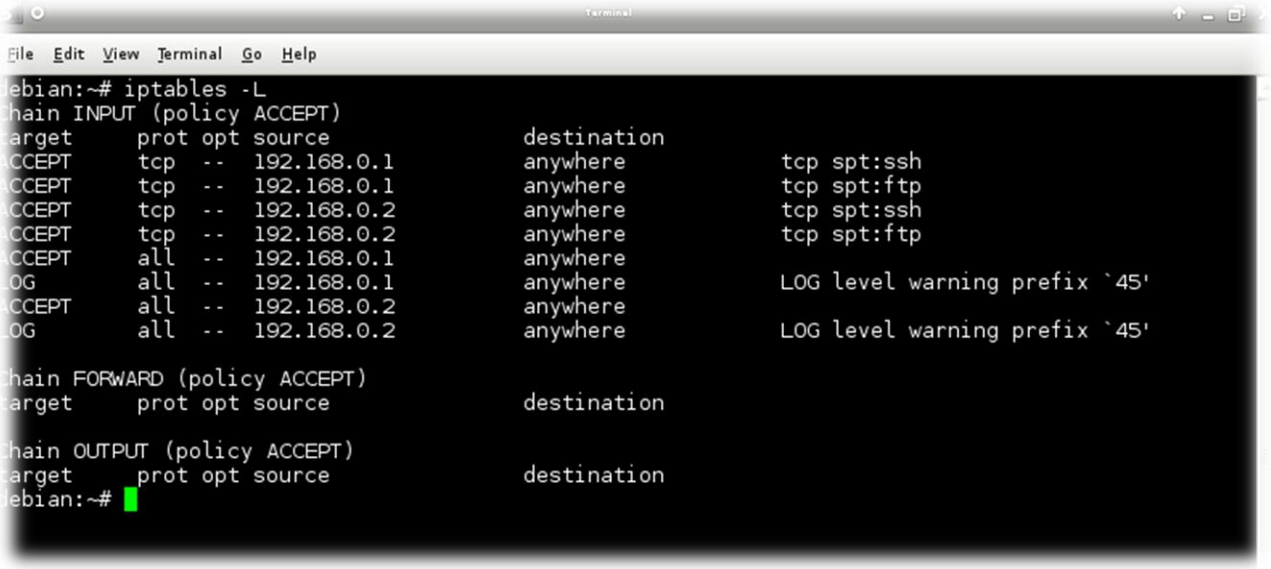
LO SCHEMA DI FUNZIONAMENTO DI NETFILTER/IPTABLES

Come è possibile notare anche dalla sinossi, la sintassi di un'istruzione Iptables è abbastanza intuitiva ma, al tempo stesso, decisamente complessa. Il numero di opzioni disponibili per ogni comando è molto elevato, il che comporta un alto grado di complessità nella gestione di operazioni che richiedono numerosi parametri. Di seguito verranno illustrati i comandi, i parametri e le estensioni più comuni. Elencarli tutti richiederebbe centinaia di pagine di descrizione e commento, ecco perchè si è deciso di elencare quelli più utilizzati da Firewall Creator. Per un maggiore approfondimento è comunque possibile consultare il manuale di Iptables digitando *man iptables* su una qualunque console Linux.

COMMANDS

- Append (-A) – Permette di appendere una nuova regola alla catena selezionata, la regola verrà inserita in coda, ovvero alla fine della catena.
  Esempio: *iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP*
- New (-N) – Permette di definire una nuova catena, la sintassi richiede solo il nome.
  Esempio: *iptables –N CHAIN_NAME*
- List (-L) – Permette di visualizare tutte le regole di una catena, se nessuna catena viene selezionata il comando visualizza automaticamente tutte le catene.
  Esempio: *iptables –L*

- Policy (-P) – Permette di definire una politica di default su una determinata catena, le catene selezionabili sono solo quelle di default, non è permesso utilizzare tale comando su catene user-defined. Se un pacchetto arriva alla fine della catena utilizzerà tale politica.
  Esempio: *iptables –P INPUT DROP*



IL COMANDO IPTABLES –L SU PIATTAFORMA DEBIAN

TARGETS

- ACCEPT – Lascia passare il pacchetto.
- DROP – Il pacchetto viene gettato via.
- RETURN – Il pacchetto viene "restituito" alla catena superiore. Se si è giunti al termine della catena superiore il destino del pacchetto viene deciso in base alla Policy definita sulla catena stessa.
- QUEUE – Il pacchetto viene inserito in una coda in modo che possa essere gestito dall'userspace.
- REJECT - Il pacchetto viene gettato via come con il target DROP ma viene inviato un messaggio ICMP a chi lo ha inviato.
- LOG – Viene eseguito il LOG sul pacchetto, ovvero il computer salva la ricezione del pacchetto in un file di log. Tale funzione è molto utile per i sistemisti in quanto permette di monitorare il corretto funzionamento della rete.

PARAMETERS

- Protocol (-p protocol) – Rappresenta il protocollo della regola, il campo protocol può essere uno fra tcp, udp, sctp, icmp, udplite, esp, ah. Se non viene specificato nulla viene preso di default all, ovvero qualunque protocollo.
- Source (-s address[/mask]) –Permette di specificare la sorgente, address può essere il nome di una network, di un host o più semplicemente un indirizzo IP tramite mask.
- Destination (-d address[/mask]) - Permette di specificare la destinazione, address può essere il nome di una network, di un host o più semplicemente un indirizzo IP tramite mask.
- Jump (-j target) – Permette di specificare il target della regola, in particolar modo con le catene user-defined. E' possibile utilizzare uno dei target definiti sopra ma anche altri target particolari come SNAT, DNAT o MARK.

Esempio completo: *iptables –A WT_CHAIN –p tcp –s 192.168.0.1 –d ROUTER_WT –j ACCEPT*

CONSIDERAZIONI

Analizzando la struttura dei Firewall Linux è facile comprendere immediatamente la necessità di una piattaforma di sviluppo che permetta di configurare e gestire in maniera veloce ed intuitiva Firewall moderni e di complessità elevata. I Firewall aziendali comprendono migliaia di regole che, molto spesso, devono essere definite manualmente dal sistemista, tale lavoro non solo richiede un enorme quantità di tempo ma può facilmente indurre in errore. In un mondo dove la sicurezza aziendale occupa un ruolo di primo piano è essenziale impedire che ciò accada. Firewall Creator deve venire incontro agli sviluppatori offrendo un'interfaccia semplice e dinamica che si adatti alle esigenze dell'utente e alla velocità con cui il mondo dell'informatica si evolve.

TECNOLOGIE UTILIZZATE

Verranno analizzate in questo paragrafo tutte le tecnologie che hanno permesso la realizzazione del progetto, è importante comprendere la potenzialità di tali strumenti di sviluppo e i motivi che hanno portato alla loro scelta. La necessità di realizzare un software dinamico e multipiattaforma porta con se difficoltà notevoli e una scelta sbagliata può risultare in gravi problemi di sviluppo.

MICROSOFT .NET FRAMEWORK

Prima di analizzare la tecnologia .NET è essenziale comprendere il concetto che si cela dietro la parola Framework.

COS'E' UN FRAMEWORK

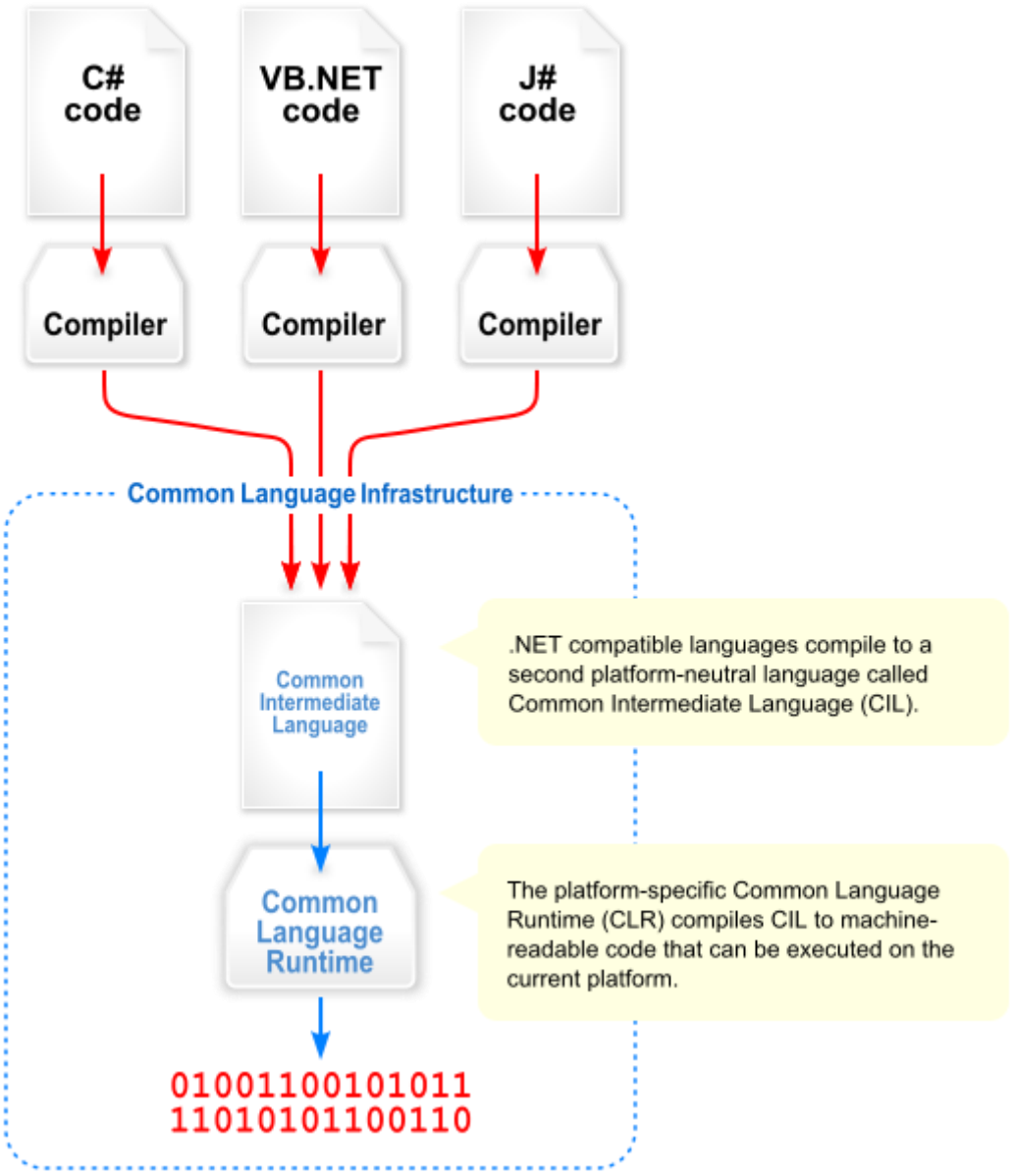
Un Framework non è altro che una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un Framework ci sono sempre delle librerie di codice che possono essere utilizzate con uno o più linguaggi di programmazione. Solitamente a tale struttura viene affiancato un IDE (*Integrated Development Environment*), ovvero un'interfaccia per lo sviluppo del software vero e proprio. Vedremo più avanti che l'IDE di riferimento per la tecnologia .NET è Visual Studio.

.NET

.NET (da leggersi come *dot net* in inglese) è un framework software rilasciato da Microsoft nel 2002, anche se il suo sviluppo era già iniziato nei lontani anni novanta. La release più recente, che è quella utilizzata da Firewall Creator è la 4.0 ed è stata rilasciata il 12 Aprile del 2010. Tale release comprende un'enorme libreria di supporto che fornisce interfacce utente, accesso ai dati, database connectivity, crittografia, applicazioni web, algoritmi e network communications. I linguaggi di programmazione supportati sono numerosi come ad esempio C#, Visual Basic o J#.

ARCHITETTURA

Come detto in precedenza il Framework .NET supporta numerosi linguaggi di programmazione ed è qui che entrano in azione per così dire le specifiche introdotte da Microsoft che sono il vero e proprio cuore dell'architettura. Il CLI (*Common Language Interface*) è una specifica che descrive il codice eseguibile e l'ambiente di esecuzione, il suo compito è quello di definire un ambiente che permetta a più linguaggi di programmazione di essere utilizzati su diverse piattaforme. Questo permette ai programmatori di non dover riscrivere lo stesso programma per architetture diverse. Sarà il Framework ad occuparsi di tradurre ogni volta. E' importante comprendere che il CIL è indipendente dalla piattaforma e che non è un'implementazione ma una semplice specifica. Il CLR invece (*Common Language Runtime*) non è altro che l'ambiente di esecuzione del CIL e permette l'effettiva traduzione in linguaggio macchina. Volendo si potrebbe pensare al CLR come la vera e propria macchina virtuale del Framework .NET.



ARCHITETTURA DELLA PIATTAFORMA .NET

C#

C# (da leggersi come *see sharp* in inglese) è un linguaggio di programmazione *object-oriented* sviluppato da Microsoft all'interno del progetto .NET. L'autore, Anders Heijsberg, è lo stesso che ha dato vita a Delphi e Turbo Pascal. C# è il linguaggio di riferimento per la tecnologia .NET, non a caso ne rispecchia tutti gli obiettivi e le caratteristiche. La sintassi prende spunto da linguaggi famosi come Delphi, C++ e Java, mentre gli strumenti di programmazione visuale sono del tutto simili a quelli di Visual Basic. Lo scopo principale di Microsoft, come definito anche nel documento ufficiale ECMA (*European Computer Manufacturers Association*), era quello di realizzare un linguaggio semplice, moderno, general-purpose e orientato agli oggetti. Come prevedibile C# supporta le specifiche CLI di .NET e il 12 Aprile 2010 è stato aggiornato alla versione 4.0 così come il Framework stesso. Non è facile definire il linguaggio C# come interpretato o compilato, la definizione corretta è che soddisfa entrambi i sistemi allo stesso tempo. Come abbiamo visto nel paragrafo precedente il codice sorgente viene compilato secondo i criteri JIT (*just in time*). Un compilatore JIT non fa altro che tradurre

dinamicamente, in fase di runtime, in modo da aumentare le performance. Questo significa che la traduzione in linguaggio macchina avviene solo al caricamento del programma prima col CIL e poi col CLR.

CTS (COMMON TYPE SYSTEM)

C# ha un sistema di tipi unificato, questo significa che ogni tipo, compresi quelli primitivi (come integer), sono una sottoclasse di *System.object*. Non a caso anche i tipi primitivi implementano il metodo ToString(). Tale sistema prende il nome di CTS (*Common Type System*) e divide i tipi di dato in due grandi categorie, *Value Types* e *Reference Types*. I primi, Value Types, sono vere e proprie aggregazioni di dati, non hanno un'identità referenziale e non ha senso eseguire comparazioni semantiche. Quando si effettuano uguaglianze viene comparato il valore vero e proprio del dato con l'istanza. Un esempio sono i tipi *int*, *float* e *char*. I Reference Types invece hanno la nozione di identità referenziale, ogni istanza è diversa dall'altra, anche se il contenuto è uguale. Ecco perchè non sempre è possibile eseguire confronti tra tali tipi di dato. Esempi comuni sono *System.object* e *System.String*.
Il sistema CTS offre due funzionalità molto importanti e che hanno reso C# estremamente flessibile e potente:

- Boxing e Unboxing – L'operazione di boxing consiste nel convertire un valore di tipo Value in uno di tipo Reference. Unboxing è l'operazione inversa ovvero converte un valore precedentemente "boxato" in uno di tipo Value, ovviamente utilizzando un cast.
- Generics – I tipi di dato generico sono la più grande feature del linguaggio C#, grazie ad essi è possibile definire classi e metodi che non specificano il tipo di dati usato fin quando non vengono istanziati. Questo permette di evitare l'uso di boxing e unboxing e di risparmiare di conseguenza tempo e risorse in runtime.

Di seguito un codice di esempio di tali funzionalità.

```
//Boxing and Unboxing

int test = 42;          // Value type.
object box = test;      // test is boxed to box.
int unbox = (int)box;   // Unboxed back to value type.

// Generics

public class GenericList<T>
{
    void Add(T input) { }
}

class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int.
        GenericList<int> list1 = new GenericList<int>();

        // Declare a list of type string.
        GenericList<string> list2 = new GenericList<string>();

        // Declare a list of type ExampleClass.
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

LIBRERIE

Esattamente come Java anche C# ha i suoi package che in questo caso vengono chiamati *namespace*. Le classi della libreria sono organizzate all'interno di numerosi namespace che raggruppano quelle con funzionalità simili. Alcuni esempi sono *System.Windows.Forms* per la gestione delle finestre in ambito GUI (*Graphic User Interface*) oppure *System.XML* per l'elaborazione e la gestione di file XML.

SINTASSI

Di seguito il classico esempio del programma Hello World.

```
using System;

class Example
{
      public static void Main()
      {
            Console.WriteLine("Hello World!");
      }
}
```

La gestione dei commenti e della documentazione viene gestita in modo simile a Javadoc ma tramite l'utilizzo di XML.

```
public class Example
{
      /// <summary>A summary of the method.</summary>
      /// <param name="Param">A description of the parameter.</param>
      /// <remarks>Remarks about the method.</remarks>
      public static void Test(int Param)
      {
      }
}
```

VISUAL STUDIO 2010 (IDE)

Visual Studio è un ambiente di sviluppo integrato (IDE) sviluppato da Microsoft. Viene utilizzato per realizzare applicazioni Console, Web e GUI (*Graphic User Int*erface) grazie al potente editor Windows Forms e WPF (*Windows Presentation Foundation*). Come tutti gli IDE moderni supporta numerose funzioni essenziali per lo sviluppo di applicazioni complesse. L'editor del codice implementa una funzione di autocompletamento del testo che utilizza la tecnologia *Microsoft IntelliSense* e rende la scrittura del codice sorgente veloce ed intuitiva. Ovviamente è disponibile un debugger che funziona sia a livello source che a livello macchina, supporta applicazioni multi-thread e può essere richiamato, quando un programma .NET crasha, anche al di fuori di Visual Studio. Come detto in precedenza tale IDE permette di realizzare applicazioni con interfaccia grafica utente e offre un potente editor visuale che, nella sua versione più recente, fa un intenso uso di XML. Esistono numerose versioni disponibili per Visual Studio, ognuna delle quali si rivolge a un particolare tipo di utenza. Esiste anche una versione completamente gratuita rivolta agli studenti universitari e delle scuole medie superiori, tale iniziativa fa parte del progetto *MSDN Accademic Alliance*. Nel corso dello sviluppo di Firewall Creator tuttavia, si farà utilizzo della versione completa, ovvero la versione 2010 Ultimate.

Nell'esempio seguente viene mostrata la potenzialità di IntelliSense.

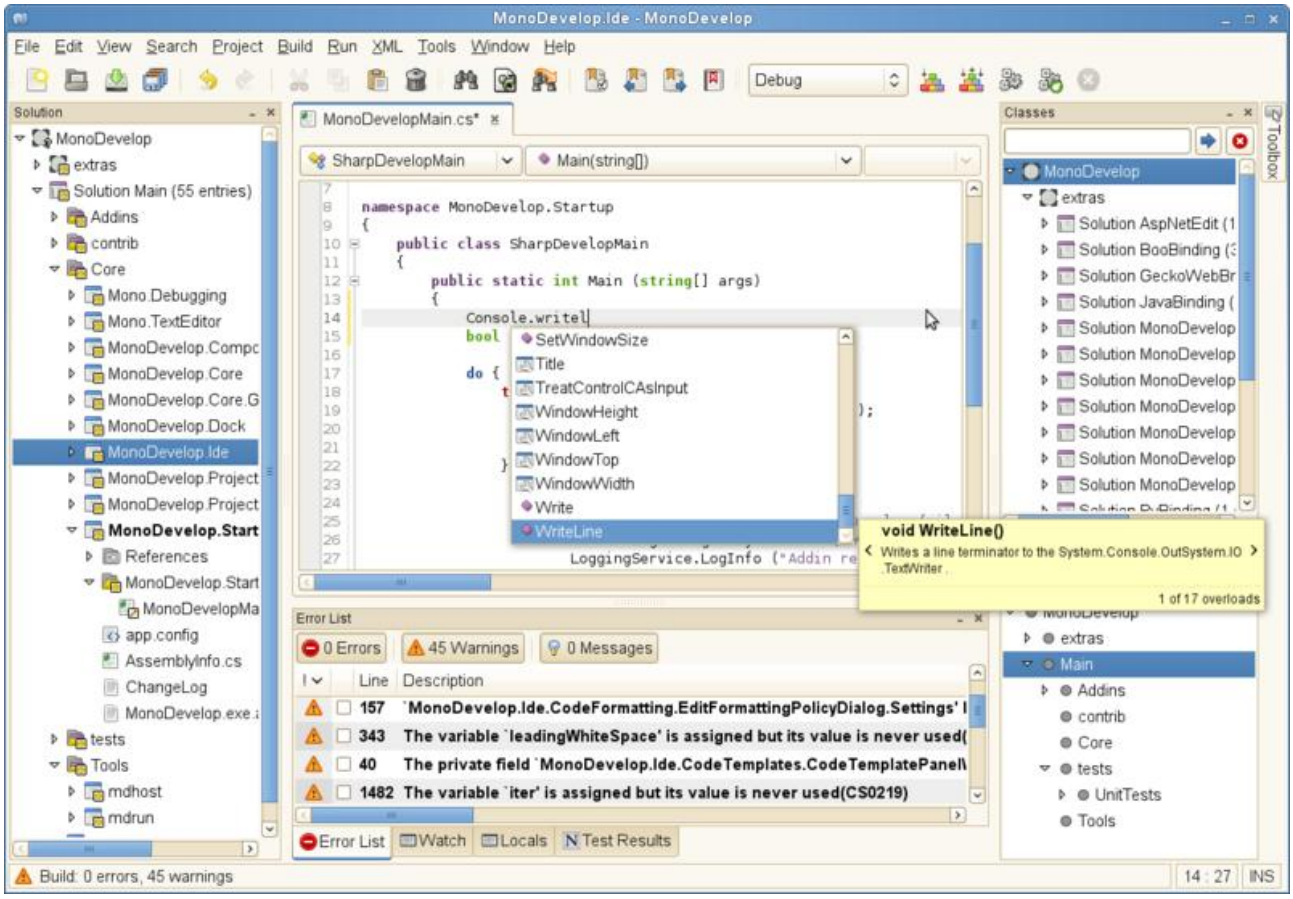
```
public class Example
{
    public void Method1();
    public void Method2(char c, int n);
}

//When the developer references the above class in source code, as soon as the
user types the period after ex, IntelliSense automatically lists all the
available member functions (in this case Method1 and Method2) and all the
available member attributes.

Example ex;
ex.
```

MONO

La necessità di sviluppare in ambiente .NET per più piattaforme diverse ha portato col tempo un gruppo di appassionati a realizzare un vero e proprio porting dell'environment Microsoft in ambiente Linux e Mac OS. Il progetto, chiamato Mono, conta ormai migliaia di utenti e permette, grazie al potente IDE, di realizzare applicazioni cross-platform con un'unica piattaforma di sviluppo. Come tutti i progetti Linux-based, anche Mono viene rilasciato sotto licenza Open-Source ed è costantemente aggiornato.

MONO DEVELOP (IDE)

L'IDE di sviluppo proposto dal Progetto Mono è Mono Develop, un ambiente del tutto simile a Visual Studio e che ripropone le stesse caratteristiche che lo hanno reso efficiente ed intuitivo come il completamento automatico del codice. Mono Develop è disponibile per tutti i sistemi operativi più famosi e verrà utilizzato nello sviluppo di Firewall Creator per realizzare il lato server dell'applicazione che, come scopriremo più avanti, girerà sotto piattaforma Linux.



L'AMBIENTE DI SVILUPPO MONO DEVELOP SU PIATTAFORMA LINUX

XML

XML (*Extensible Markup Language*) non è altro che un insieme di regole utilizzate per codificare un documento in modo che sia leggibile ed interpretabile da una macchina. L'obiettivo principale è quello di offrire un design semplice, generalizzato e che ben si adatti al mondo di Internet. Il primo standard, prodotto da W3C (*World Wide Web Consortium,*) risale al 1996 ma è solo negli ultimi anni che XML ha ottenuto un enorme successo. Centinaia di linguaggi di programmazione sono stati infatti sviluppati sulla base di XML, l'esempio più famoso (che utilizziamo nella vita di tutti i giorni) è RSS (*RDF Site Summary*). Il formato XML inoltre, viene utilizzato come base anche nelle suite di Office, OpenOffice e Apple iWork.

STRUTTURA

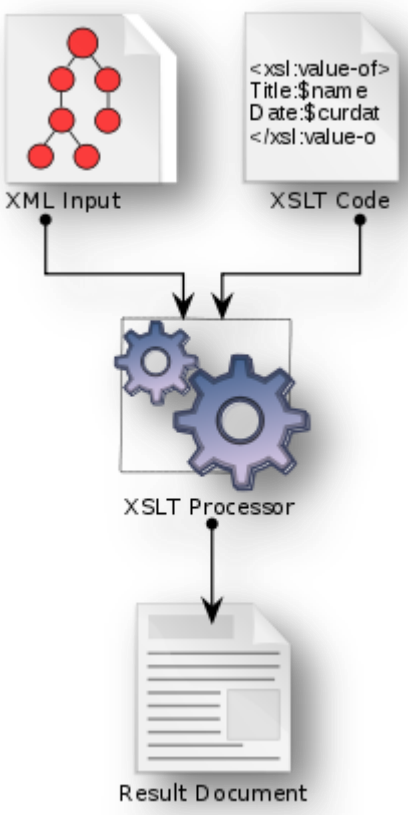
Anche se la definizione potrebbe trarre in inganno in realtà XML va ben al di là della semplice codifica di documenti. L'utilizzo più frequente infatti è quello adibito alla rappresentazione di strutture dati. Andremo ora ad analizzare i costrutti più importanti del linguaggio XML facendo particolare attenzione alla codifica dei testi che, nel caso di Firewall Creator, giocherà un ruolo molto importante.

- XML Declaration – Ogni file XML inizia con questa particolare riga di codice, i due attributi all'interno definiscono la versione di XML utilizzata (Firewall Creator utilizza la 1.0) e la codifica del testo. Firewall Creator da la possibilità all'utente di sceglire il tipo di econding da utilizzare in modo che il Firewall possa funzionare su macchine con codifica diversa.
- Carattere – XML in definitiva è un insieme di caratteri, qualunque carattere Unicode legale può essere utilizzato.
- Parser e Applicazione – Il parser non è altro che un processo che analizza i markup e passa le informazioni sulla struttura a una applicazione. Il parsing di un file XML in C# viene eseguito tramite le classi *System.XML*.
- Markup e Contenuti – I caratteri che compongono un file XML vengono divisi in Markup e Contenuti, la distinzione è puramente sintattica. Le stringhe di Markup iniziano con il carattere "<" e terminano con il carattere ">". Le stringhe di caratteri che non sono Markup sono automaticamente dei Contenuti.
- Tag – Un tag è un costrutto Markup che inizia con "<" e finisce con ">", ne esistono tre tipi, start-tags (<example>), end-tags (</example>) e empty-element-tags (<line-break />).
- Elementi – E' un costrutto che inizia con uno start-tags e finisce con un relativo end-tags. Anche un empty-element-tags è un elemento. I caratteri all'interno dei due tag sono il contenuto è possono contenere a loro volta degli elementi, dei markup e dei figli (child elements). Un esempio è <Example>Hello World!</Example>.
- Attributi – Consistono nella coppia nome valore che esiste all'interno di start-tags e end-tags oppure di empty-element-tags. Un esempio è .

SINTASSI

Di seguito un esempio di un file XML generato da Firewall Creator, l'esempio è banale ma al tempo stesso permette di visualizzare tutti i principali costrutti che compongono la struttura del linguaggio XML. E' possibile vedere per esempio il codice relativo all'XML Declaration nella prima riga.

```
<?xml version="1.0" encoding="UTF-8" ?>
<firewall>
      <eth name="0">
            <rule type="override" override="$IPTABLES -P INPUT DROP" />
      </eth>
</firewall>
```



COME FUNZIONA XSLT

XSL

Il termine XSL (*Extensible StyleSheet Language*) si riferisce ad una famiglia di linguaggi usati per trasformare il linguaggio XML. Lo standard XML prevede che i contenuti di un documento siano separati dalla formattazione della pagina con cui verranno pubblicati. E' fondamentale comprendere questa distinzione in quanto sta alla base dell'enorme potenzialità di XML. Tramite XSL è possibile "visualizzare" il contenuto di un file XML in numerose forme diverse, per esempio tramite una pagina web, un file di testo o una presentazione. XSL si compone di tre linguaggi diversi, *XSL Transformations*, *XSL Formatting Objects* e *XML Path*. Presteremo particolare attenzione a XSLT, in quanto Firewall Creator farà uso di un enorme foglio di stile in grado di tradurre i Firewall XML in linguaggio Iptables Linux.

XSL TRANSFORMATION

XSLT è il linguaggio di trasformazione XML, tramite esso è possibile trasformare un file XML in un altro file XML oppure in un qualsiasi altro formato come ad esempio HTML, XHTML o RTF. Per "generare" una trasformazione sono necessari due file, il file XML di partenza e un foglio di stile XSLT. La trasformazione vera e propria avviene tramite un Processore XSLT che, nel caso di Firewall Creator, viene gestito dal Framework .NET stesso tramite le classi *System.XML.XSL*.

SINTASSI

La sintassi di XSLT è molto complessa e permette di generare, a partire da un file XML, decine di presentazioni diverse. Tramite l'utilizzo di template è possibile definire con facilità come verrà trasformato un preciso elemento di un file XML. L'esempio seguente prende spunto dai file XML e XSLT generati da Firewall Creator.

```
<!--- XML Element to be converted -->

<RULE type="SNAT" output="$ETH" protocol="tcp"
rangeaddresses="194.236.50.155:32000" />

<!--- XSLT code selected runtime-->

<xsl:if test="@type='SNAT'">
      <xsl:if test="boolean(string (@protocol))">
            <xsl:if test="not(boolean(string (@destination))) and
not(boolean(string (@source)))">
                  <xsl:if test="not(boolean(string (@porttype)))">
                        $IPTABLES -t nat -A POSTROUTING
                        -o <xsl:value-of select="@output"/>
                        -p <xsl:value-of select="@protocol"/>
                        -j SNAT --to-source <xsl:value-of
select="@rangeaddresses"/>
                        <xsl:text>&#13;&#10;</xsl:text>
```

```
                    </xsl:if>
                </xsl:if>
            </xsl:if>
</xsl:if>

<!-- The actual result, an Iptables rule that can be executed over a Linux
machine -->

$IPTABLES -t nat -A POSTROUTING -o $ETH -p tcp -j SNAT --to-source
194.236.50.155:32000
```

APPLICATIVI DI SUPPORTO

Per la scrittura del foglio di stile XSL si è fatto uso di Visual Studio, l'IDE Microsoft infatti supporta anche la scrittura di questi tipi di file. Lo stesso vale anche per XML anche se in realtà i fogli XML vengono scritti direttamente da Firewall Creator stesso e di conseguenza l'IDE è stato utilizzato solo per controllare il corretto funzionamento del programma insieme a Notepad++.

L'esigenza di lavorare su più piattaforme contemporaneamente ha richiesto l'utilizzo di numerose macchine virtuali. La scelta è ricaduta su VMWare e sul suo prodotto più famoso, ovvero VMWare Player. Durante lo sviluppo sono stati virtualizzati due sistemi operativi Linux diversi per testare il corretto funzionamento del programma su varie piattaforme. Debian 5 ha permesso di testare l'installazione remota previa installazione del Framwork Mono. Ubuntu 10.04 è stato invece utilizzato per testare l'installazione On The Fly.

FUNZIONALITA'

Come già visto nei precedenti paragrafi lo scopo principale di Firewall Creator è quello di offrire un'interfaccia semplice e funzionale ma, allo stesso tempo, anche complessa per chi ha bisogno di funzionalità avanzate. In questo capitolo scopriremo tutte le features che fanno di Firewall Creator un programma non solo innovativo ma anche e soprattutto dinamico.

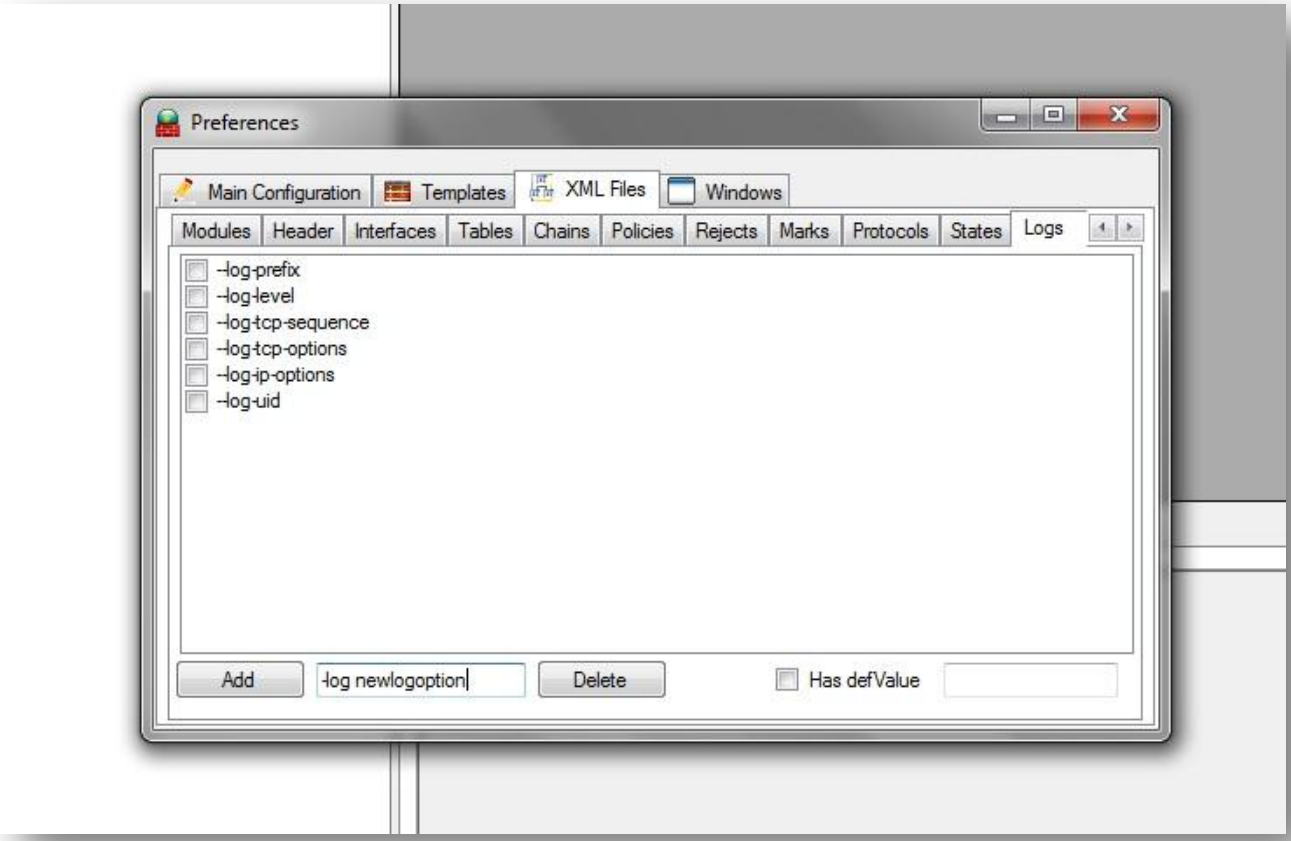
DINAMICITA' (XML UPDATE)

In un mondo IT che si evolve giorno dopo giorno è essenziale sviluppare dei software dinamici e facilmente aggiornabili. L'ambiente Linux si evolve letteralmente di ora in ora grazie ad una community di appassionati che ne ha fatto il sistema di riferimento per ambienti Network. La sua flessibilità è la chiave del suo successo, ma come può Firewall Creator adeguarsi a questa flessibilità? La risposta sta in una sola parola, XML.

Non solo i Firewall creati dal programma sono in formato XML infatti, tutta l'intera libreria del programma è a sua volta realizzata in XML. Per capire il significato di questa affermazione è meglio partire da un esempio pratico. Come abbiamo visto nel paragrafo dedicato a Iptables esistono centinaia di comandi, target e parametri diversi. Ovviamente la prima versione di Iptables non conteneva tutte queste funzionalità, col passare degli anni è stato aggiornato e presumibilmente verrà aggiornato anche in futuro. Immaginiamo che i progettisti Linux decidano di aggiungere una nuova opzione per il parametro –j LOG, chiameremo tale opzione newlogoption. La release più recente di Firewall Creator ovviamente non include tale opzione, per il sistemista sarebbe impossibile quindi scrivere una regola che la supporti senza utilizzare regole override o senza chiamare l'azienda che ha scritto il programma per commissionare una patch. Il problema non è di facile soluzione per due motivi, scrivere una regola override significa perdere completamente la potenzialità di Firewall Creator e commissionare una patch significa un dispendio di denaro non indifferente.

Firewall Creator offre però una soluzione istantanea, letteralmente istantanea. Dato che le librerie sono in XML e che tutti i parametri sono salvati in una libreria, è sufficiente aggiungere tale nuovo parametro alla lista per poterne usufruire in men che non si dica. Il programma infatti tiene conto anche di parametri speciali che possono utilizzare a loro volta ulteriori opzioni, è sufficiente un check sulla clausola HasValue per indicare al programma come dovrà comportarsi.

Detto fatto, in una manciata di click il nostro programma è aggiornato alla versione più recente di Iptables. Ovviamente il sistemista non deve essere per forza obbligato ad aggiornare la lista per conto suo, le librerie sono semplici file XML e, di conseguenza, sarà possibile eventualmente scaricarne una versione aggiornata dai server WorkTeam. Di seguito viene presentato l'esempio sopra citato, è opportuno sottolineare che ogni singola opzione può essere aggiornata sul programma, se un domani verrà approvata una nuova Policy dal nome NEWACCEPT la si potrà aggoiungere, se verrà aggiunta nuova table dal nome newmangle la si potrà aggiungere, tutto questo grazie all'enorme potenza di XML.



INSERIMENTO DI UN NUOVO TIPO DI PARAMETRO LOG

VARIABILI MULTIPLE

Molto spesso quando si scrive una regola di un Firewall è necessario applicare tale regola a decine di host o servizi. A livello di codice il sistemista potrebbe risolvere tale problema utilizzando un ciclo for che è supportato da Linux e Iptables. Non tutti i sistemisti potrebbero però essere in grado di utilizzare tale funzionalità e, inoltre, non sempre questo è possibile. Firewall Creator risolve tale problema introducendo l'uso di variabili multiple. Per capire che cosa si intende ci serviremo ancora una volta di un esempio.

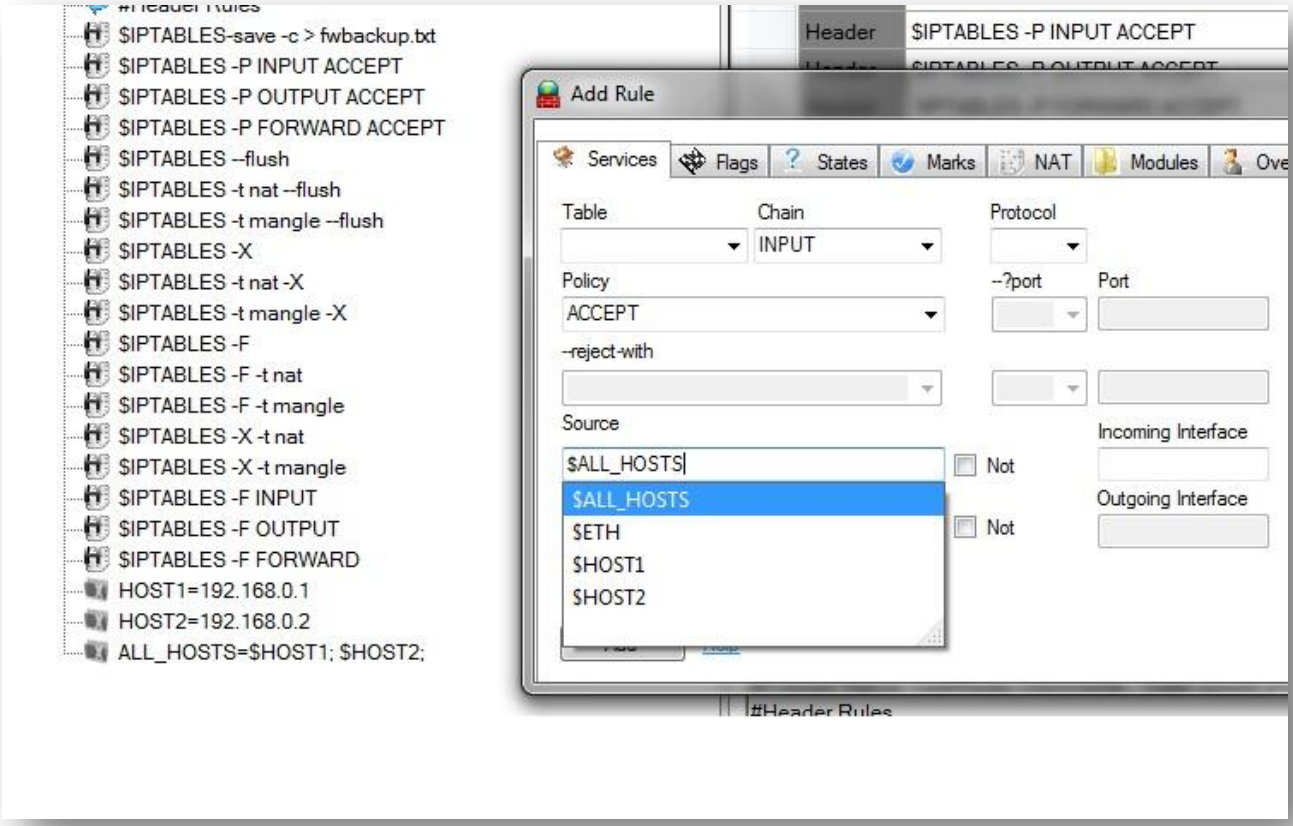
Supponiamo di avere una rete con centinaia di Host e Servizi, il nostro sistemista vuole applicare una o più regole di Policy ad alcuni PC e servizi della rete. Senza l'utilizzo di un ciclo for il sistemista dovrebbe scrivere centinaia e centinaia di regole manualmente ma, con Firewall Creator, questo non è più necessario. Una volta definiti gli Host e i Servizi è sufficiente assegnare questi valori a un'unica variabile (che per questo motivo viene chiamata variabile multipla) e la regola che si voleva definire verrà applicata su di essa.

Definiamo a titolo di esempio le seguenti variabili:

```
HOST1 = "192.168.0.1"
HOST2 = "192.168.0.2"
ssh = "22"
ftp = "21"
ALL_HOSTS = "$HOST1; $HOST2"
ALL_SERVICES = "$ssh; $ftp"
```

D'ora in poi se si vorrà applicare una regola che valga contemporaneamente per HOST1 e HOST2 basterà utilizzare ALL_HOSTS, lo stesso ragionamento vale ovviamente anche per ssh, ftp e ALL_SERVICES.

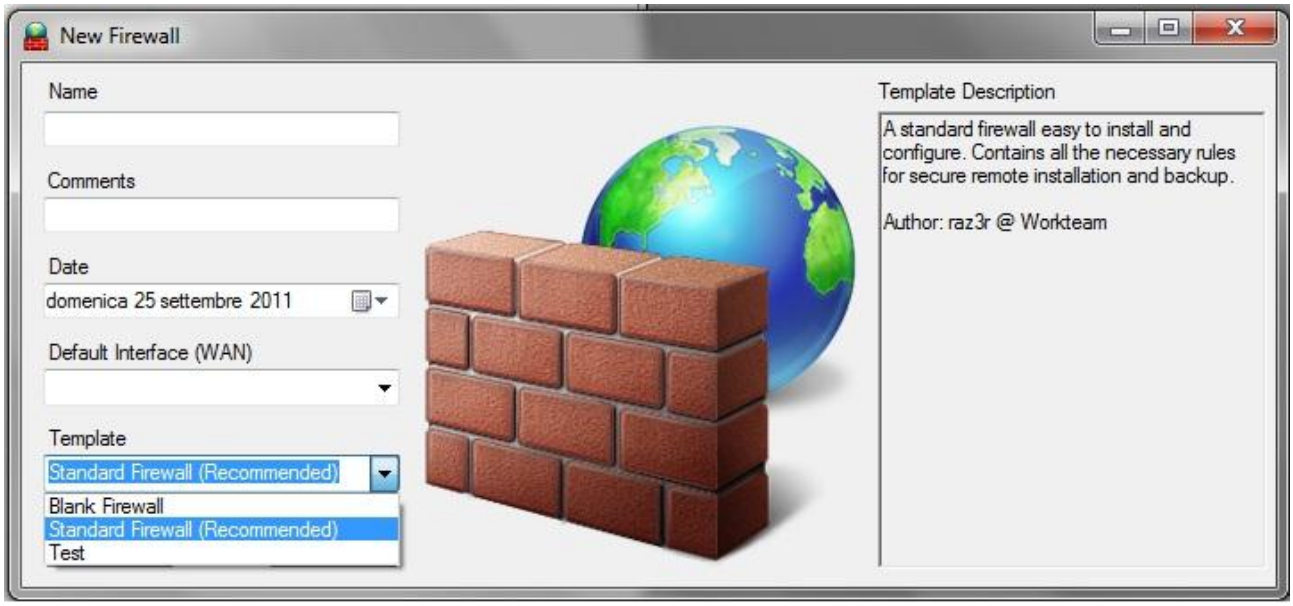
A livello organizzativo questa funzionalità è utilissima perchè è possibile creare vere e proprie sottoreti logiche e applicare a ognuna di esse delle determinate regole. Per esempio un sistemista vorrebbe poter vietare l'utilizzo di Facebook a tutti i computer del secondo piano, se prima era troppo pigro per farlo (il secondo piano ha centinaia di computer!) adesso non avrà più scuse.



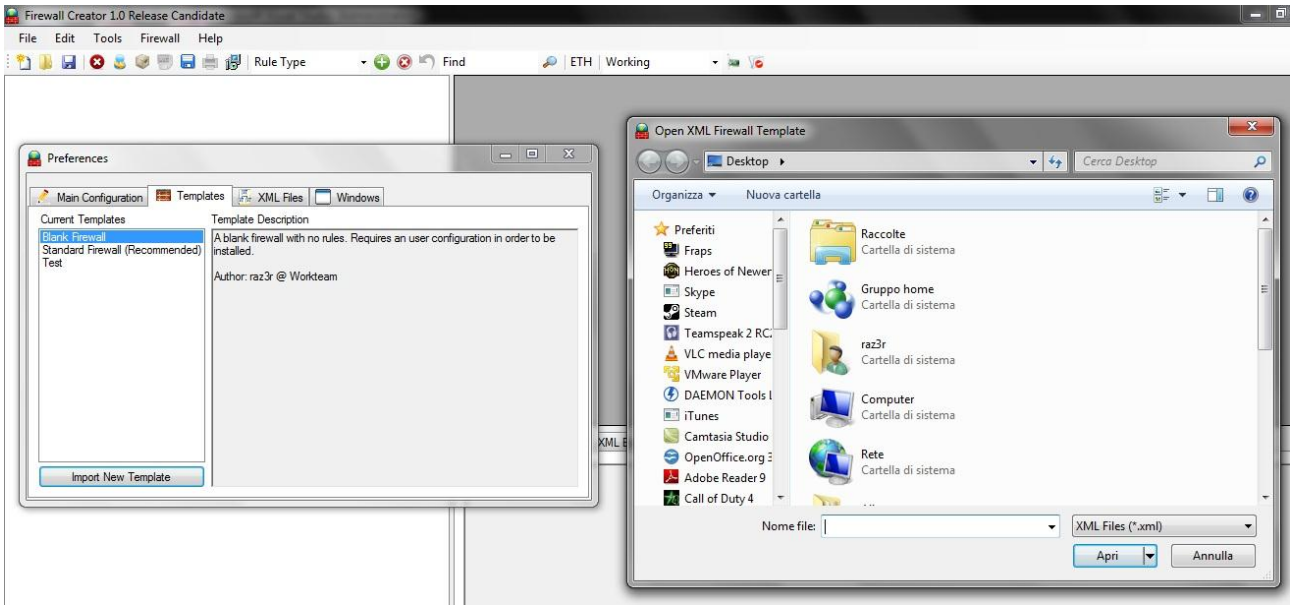
ESEMPIO DI UTILIZZO DI VARIABILI MULTIPLE

TEMPLATES

Firewall Creator supporta l'utilizzo di template per la creazione dei Firewall. E' possibile in ogni momento creare dei template per un utilizzo futuro. Ad esempio supponiamo di aver realizzato un header standard per i nostri Firewall, riscrivere ogni volta questa parte porterebbe via del tempo inutile. Creare invece un template e importarlo successivamente è la modalità più veloce per riutilizzare il lavoro già svolto in passato. Tale funzionalità può tornare utile anche a livello di community e contenuti user-generated. E' possibile infatti creare dei template funzionali (per esempio template per Firewall casalinghi, standard, avanzati etc.) e renderli disponibili su Internet per il download. Potrebbero essere un'ottima base da cui partire per poi sviluppare il Firewall secondo le relative esigenze.



E' POSSIBILE SCEGLIERE IL TEMPLATE AL MOMENTO DELLA CREAZIONE DEL FIREWALL



IN OGNI MOMENTO E POSSIBILE IMPORTARE UN NUOVO TEMPLATE

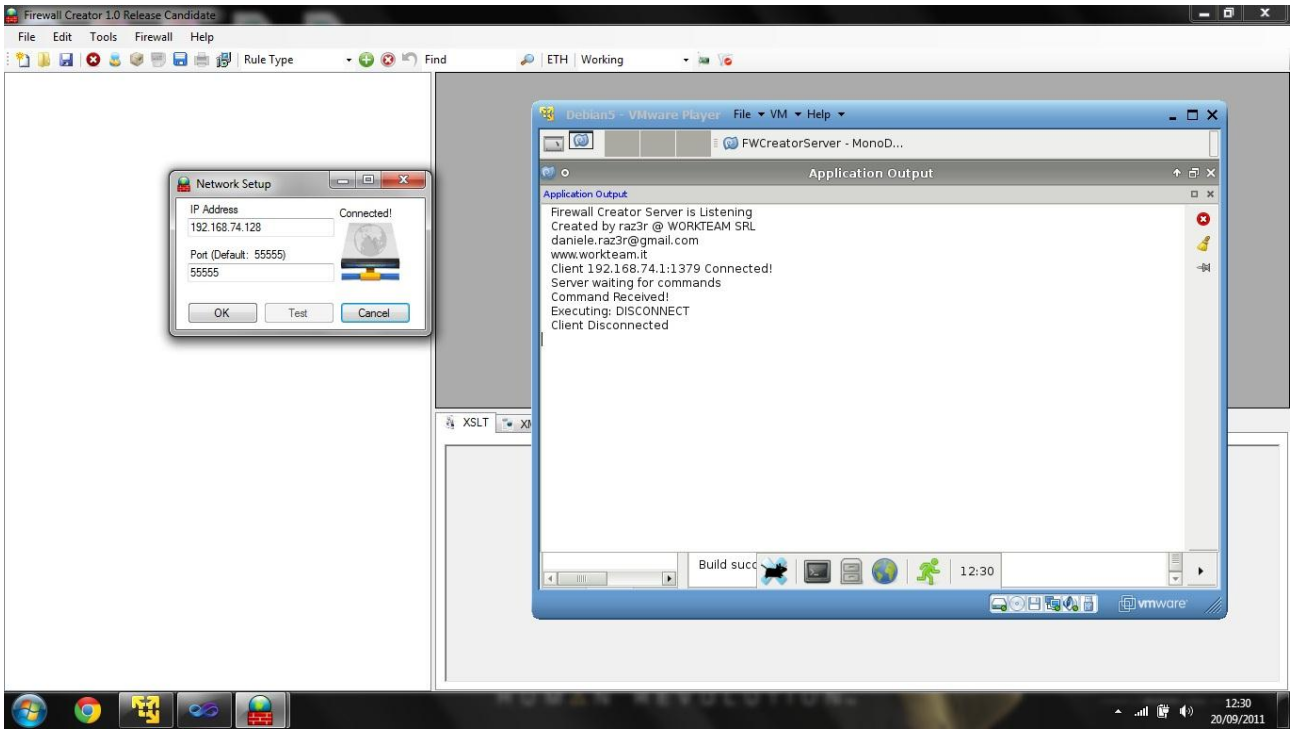
INSTALLAZIONE REMOTA E ON THE FLY

Realizzare un Firewall è importante ma effettuarne un corretto deploy lo è ancora di più. Firewall Creator permette due tipi diversi di installazione in base alle esigenze del sistemista.

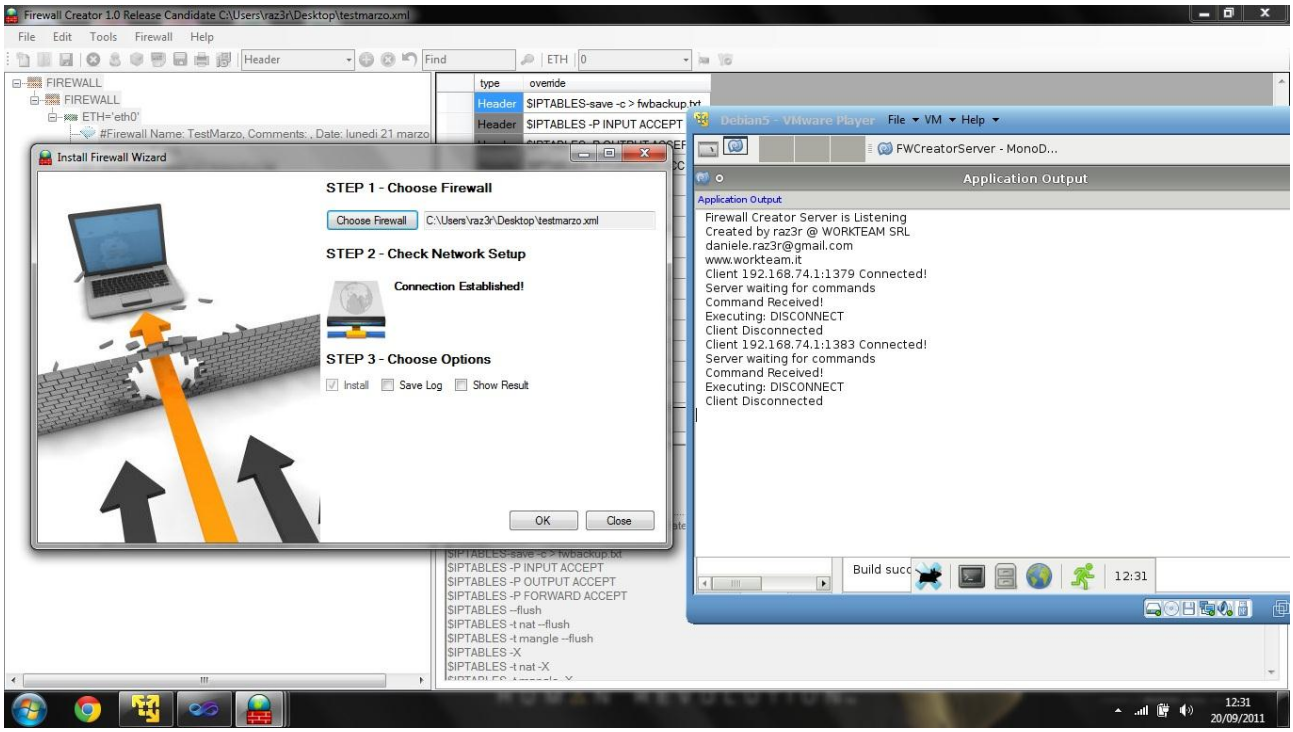
INSTALLAZIONE REMOTA

Grazie alle potenzialità offerte dal Framework .NET e da Mono è stato facile realizzare una Socket che permettesse la connessione tra infrastrutture diverse (Windows e Linux) per lo scambio di informazioni. Le informazioni di cui parliamo sono ovviamente le regole del Firewall.

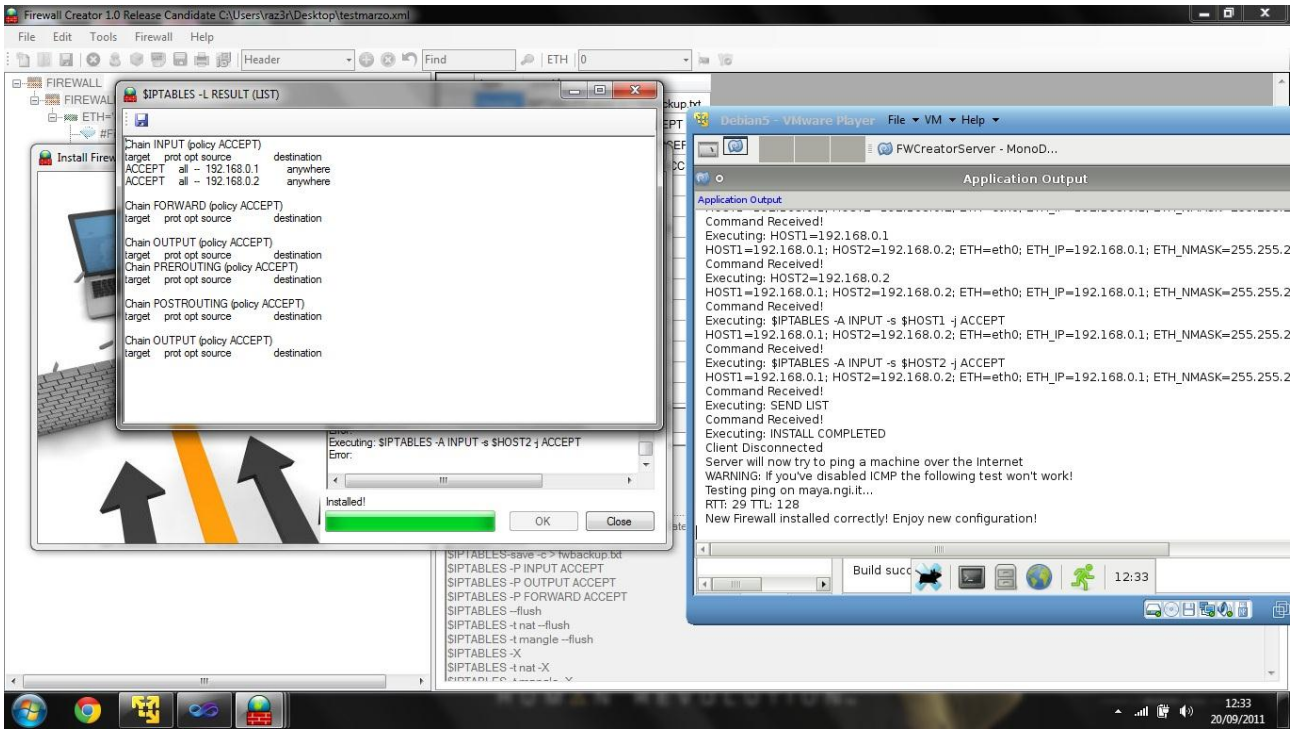
Il programma, una volta instaurata una connessione sicura, inizia a installare il Firewall, regola per regola. Ad ogni regola installata il Server (ovvero la macchina Linux) restituisce un pacchetto vuoto se tutto è andato a buon fine, altrimenti restituisce la stringa inviata dal flusso di Standard Error. Il sistemista, in caso qualcosa non sia andato a buon fine, può annullare l'installazione ripristinando il Firewall precedente. Ovviamente è necessario che chi installa il Firewall non sia un pasticcione, se ci si chiude fuori dalla macchina (per esempio inserendo come prima regola un DROP sulla catena di INPUT) sarà praticamente impossibile accedervi di nuovo, a meno che qualcuno sul Server non si renda conto che qualcosa è andato storto. In realtà Firewall Creator può gestire tale opportunità in una maniera abbastanza semplice. A meno che il sistemista non cambi le impostazioni di default, la macchina Server deve essere sempre in grado (in ogni momento, e quindi anche durante l'installazione) di pingare il PC del sistemista. Se per qualche strano motivo non ci riesce, il programma avvisa l'utente che con molta probabilità l'installazione della nuova versione del Firewall non è andata a buon fine. L'utente a questo punto può effettuare il ripristino della precedente versione (che è sicuramente funzionante) oppure telefonare al sistemista per chiedere delucidazioni. Del resto bisogna considerare che Internet e le reti LAN non sono affidabili al 100%, una congestione dei router o un fulmine potrebbero aver cancellato i nostri pacchetti in transito.



TEST DI CONNESSIONE AL SERVER REMOTO (NELL'ESEMPIO UNA MACCHINA VIRTUALE DEBIAN)



CONFIGURAZIONE DELL'INSTALLAZIONE REMOTA

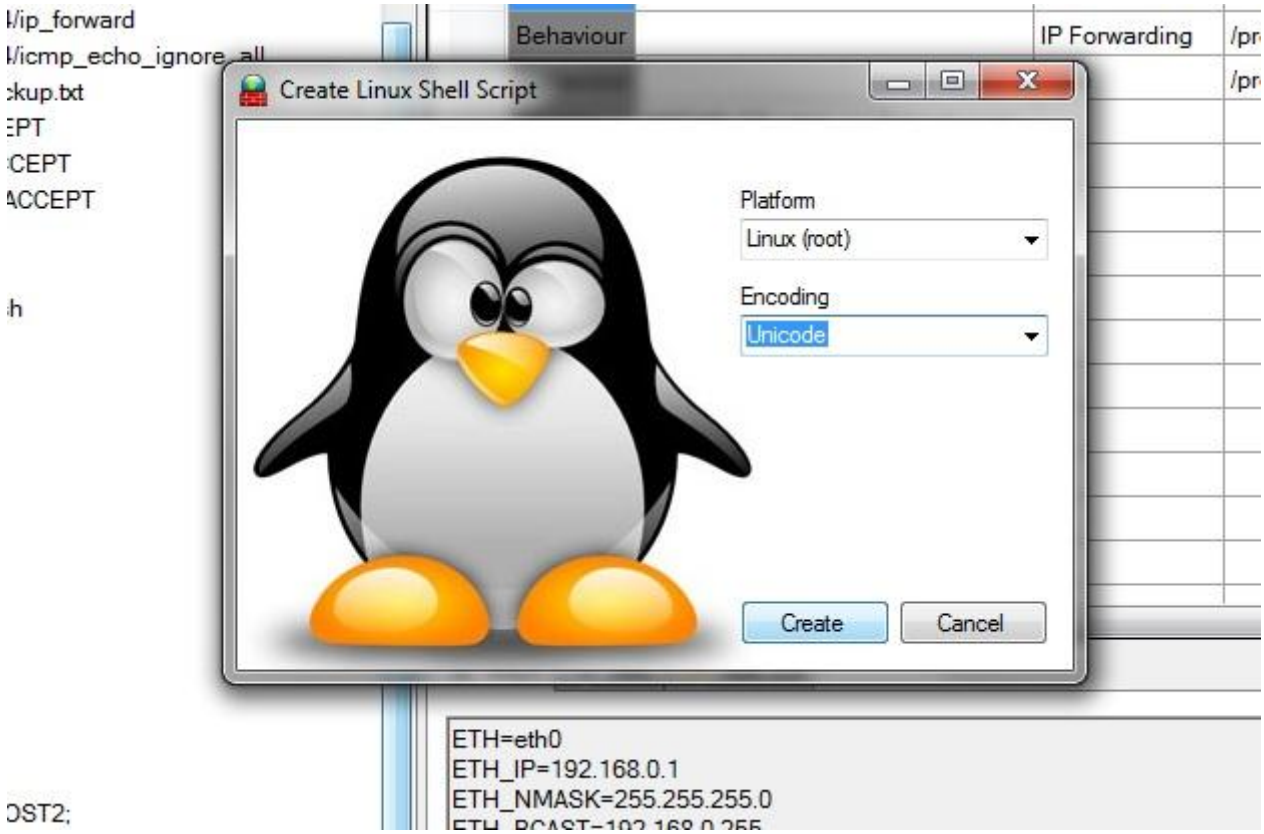


INSTALLAZIONE TERMINATA CON SUCCESSO, FIREWALL CREATOR VISUALIZZA IL RISULTATO

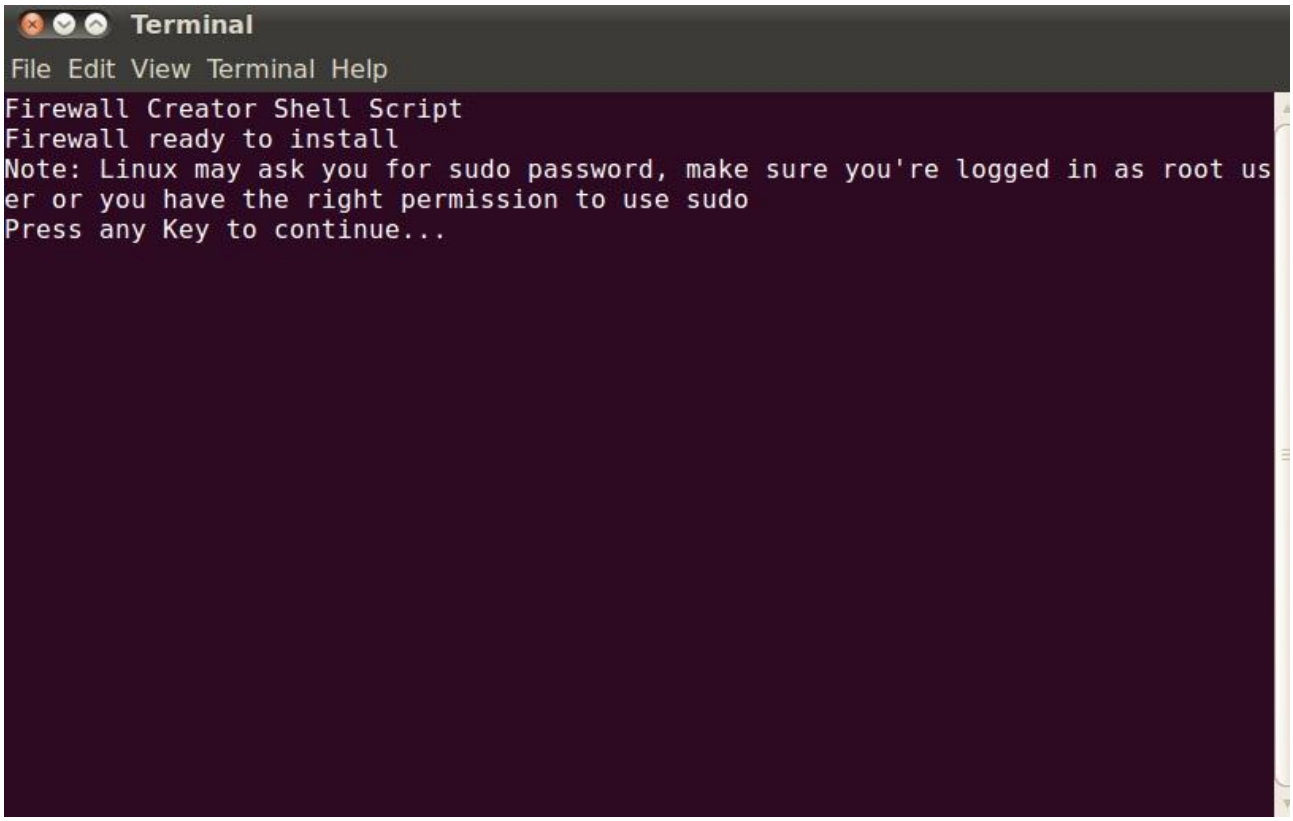
INSTALLAZIONE ON THE FLY

Non sempre il sistemista ha accesso diretto ai PC su cui dovrà installare il Firewall, anzi, nella maggior parte dei casi è anche impossibile. Il programma viene incontro a tale problema con una funzionalità avanzata che permette di salvare il Firewall su un file Script Shell di Linux. Questo altro non è che un file di testo interpretabile da una macchina Linux come codice eseguibile, è importante però definire prima della creazione la codifica di testo che utilizza la distribuzione Linux su cui verrà installato il Firewall.

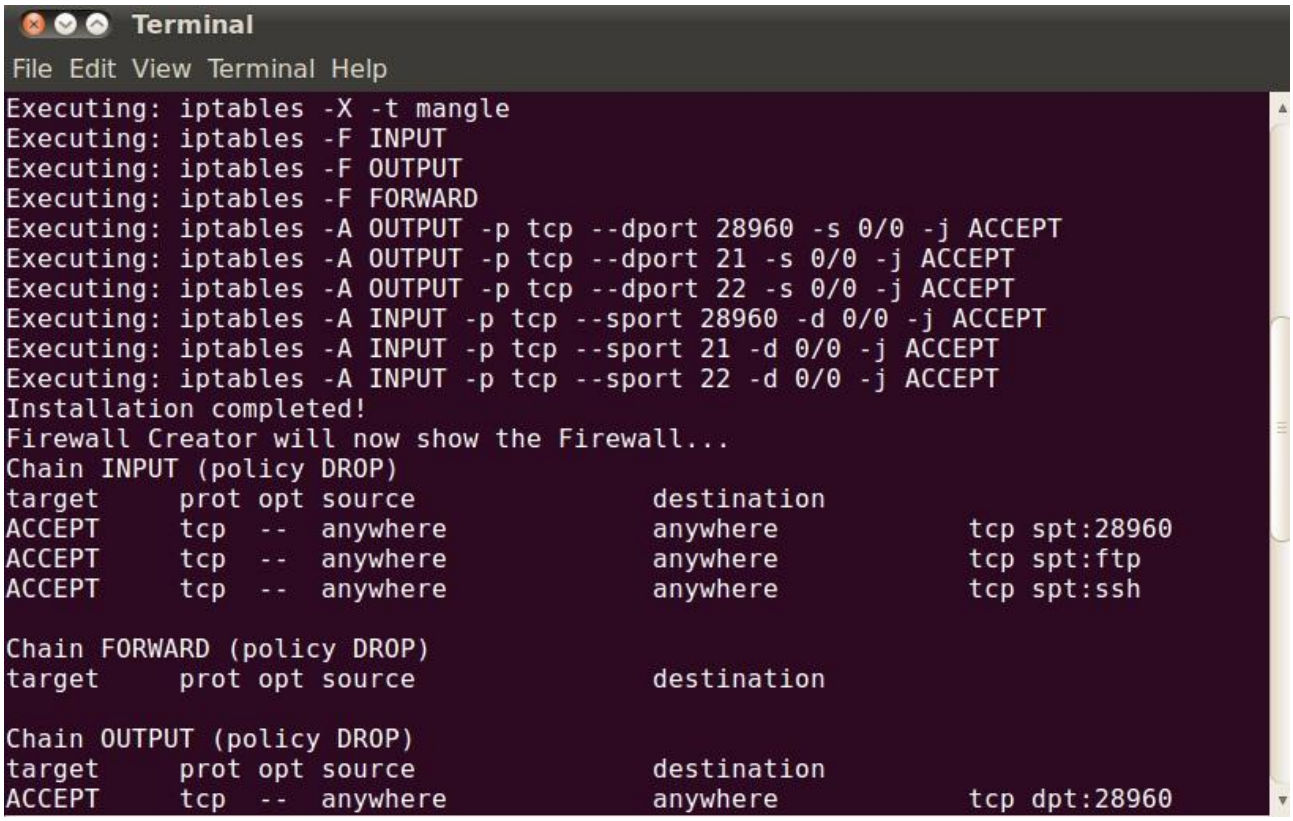
Una volta salvato tale file è possibile copiarlo su una chiavetta USB e recarsi in loco per installarlo con un semplice doppio click. Volendo è possibile inviare il file anche via email, certo il vostro datore di lavoro non sarà molto felice di sapere che dati sensibili viaggiano in maniera non protetta sulla grande Rete.



LA CREAZIONE DELLO SCRIPT SU FIREWALL CREATOR



INSTALLAZIONE SU PIATTAFORMA UBUNTU



INSTALLAZIONE TERMINATA CON SUCCESSO, VIENE VISUALIZZATO IL FIREWALL

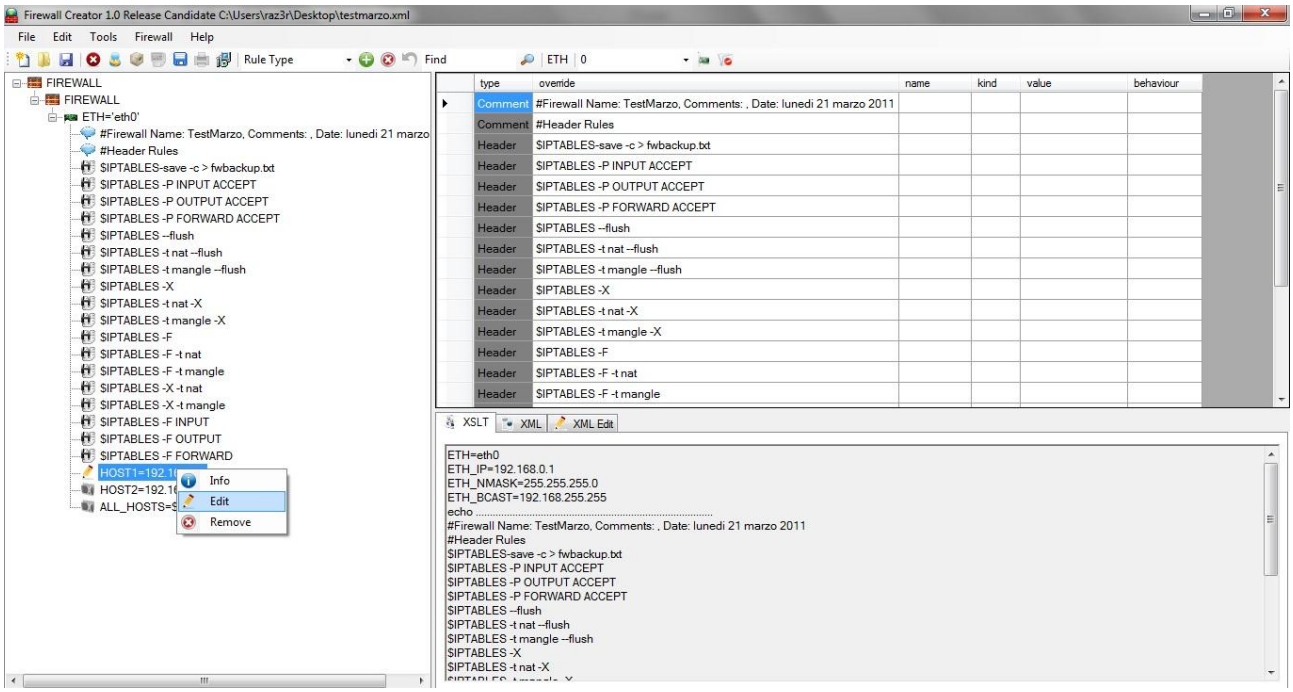
INTERFACCIA

Per poter offrire un programma semplice e dinamico spesso l'architettura non basta, è necessaria un'interfaccia user-friendly che permetta con pochi passaggi di raggiungere l'obiettivo cercato. Firewall Creator offre all'utente diverse soluzioni, alcune immediate e altre di livello avanzato per venire incontro anche alle esigenze più complesse. In questo paragrafo analizzeremo l'interfaccia principale del programma vedendo come ogni singola funzionalità ha uno scopo ben preciso e lavora insieme alle altre per offrire la massima efficienza nella scrittura e nella gestione di un Firewall.

TREE VIEW

Quando abbiamo parlato di XML nel paragrafo dedicato alle tecnologie abbiamo visto come la struttura ben ricalchi quella di un albero. In effetti è proprio così, abbiamo il Firewall che è la radice, abbiamo le interfacce di rete che sono i sottoalberi e infine abbiamo le regole, ovvero le foglie. Tale visuale è molto utile per chi progetta il Firewall perchè non solo consente di averne una visione di insieme ma permette inoltre di ordinare le regole e, come abbiamo visto in precedenza, l'ordinamento è fondamentale per il corretto funzionamento del Firewall.

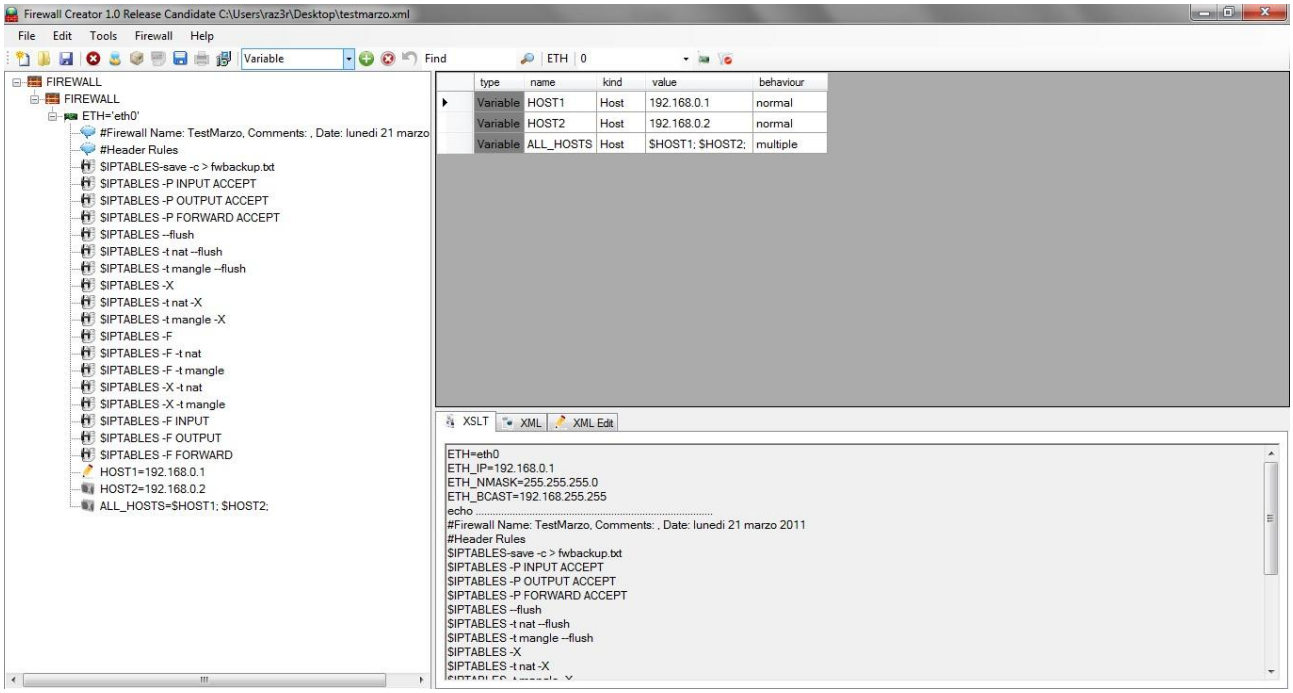
Tramite tale visuale è possibile interagire con le singole regole, è sufficiente cliccare col tasto destro su una regola per ottenere informazioni, effettuare modifiche o rimuoverla. La pressione del tasto Enter o il doppio click permettono di muoversi direttamente in modalità edit. E' possibile anche spostare o swappare le regole.



VISUALE TREEVIEW

DATAGRID VIEW

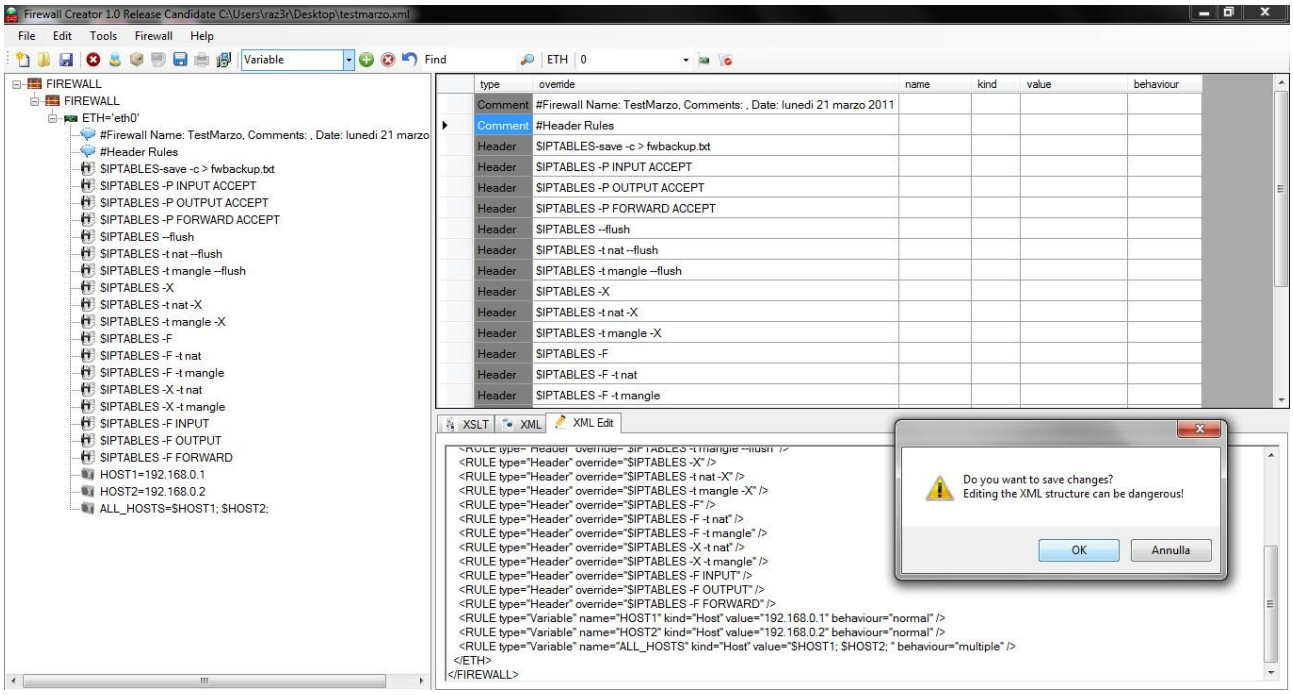
Quando è necessario modificare un singolo campo di una regola, la visuale DataGrid può tornare molto utile in quanto interagisce col Firewall come se fosse un Database. Ogni regola viene interpretata come una serie di valori di una tabella, le varie colonne sono i vari campi disponibili. La modifica a tale livello è però consigliata solo ad un'utenza esperta, il programma effettua controlli ad ogni inserimento ma non è in grado di rilevare determinati tipi di errori. In poche parole una regola può essere sintatticamente corretta ma non è detto che il risultato ottenuto sia quello voluto dal sistemista. Ecco perchè è fondamentale agire con cautela ed essere certi di quello che si fa.



VISUALE DATAGRID VIEW

Quando una modifica visuale non è sufficiente si può sempre ricorrere alla forza bruta, la visuale XML Edit serve proprio a questo. Come dice il nome è possibile andare a modificare il file XML come se fosse un semplice file di testo. Tale modalità è stata aggiunta a scopo di debug durante l'implementazione di Firewall Creator e si è decisa comunque di lasciarla nella versione finale. Modificare i Firewall a questo livello però è assolutamente rischioso e, se non si conosce la sintassi di XML e la struttura dei Firewall creati dal programma, c'è il rischio di perdere completamente i dati. Per fortuna il software viene incontro all'utente e, se le modifiche portano a degli errori nella successiva lettura, permette di ripristinare la versione precedentemente funzionante.

Le visuali XML e XSLT servono soltanto ad avere un'idea del risultato complessivo, la prima non fa altro che visualizzare il Firewall come un vero e proprio XML, la seconda invece mostra il risultato a livello di sintassi Iptables. In ogni momento quindi è possibile vedere cosa andrà effettivamente in esecuzione quando, eventualmente, installeremo il Firewall.

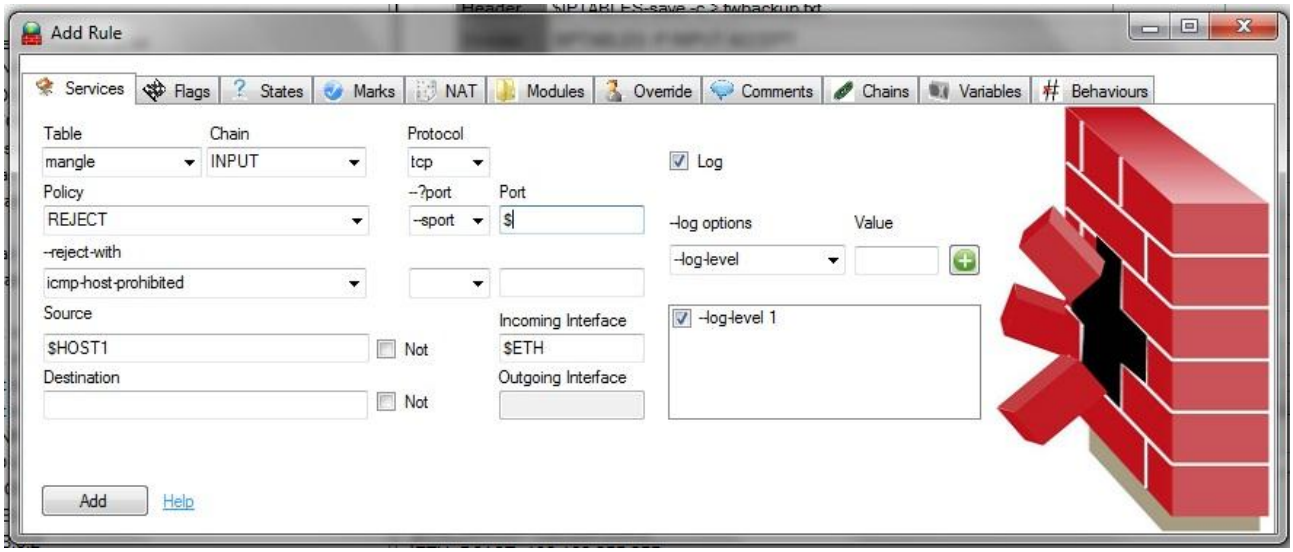


VISUALE XML EDIT

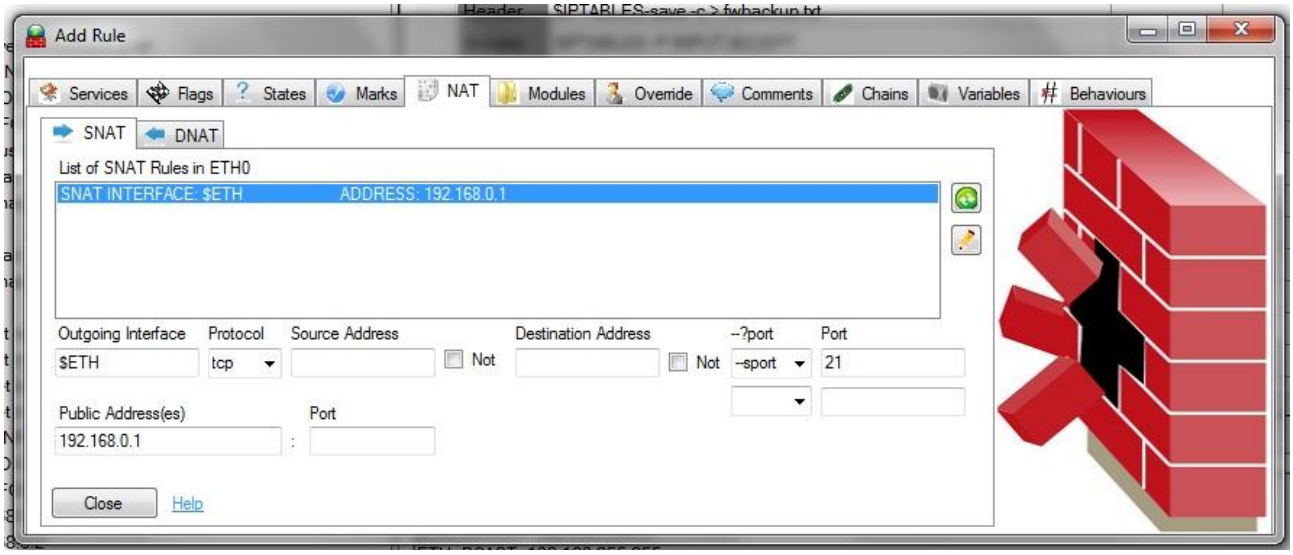
ADD RULE

Tramite questa feature è possibile aggiungere delle regole a un Firewall esistente. Firewall Creator utilizza una suddivisione logica delle regole in base al loro tipo e al loro scopo. Nella release attuale ci sono numerosi tipi di regola e di seguito viene elencato il loro scopo.

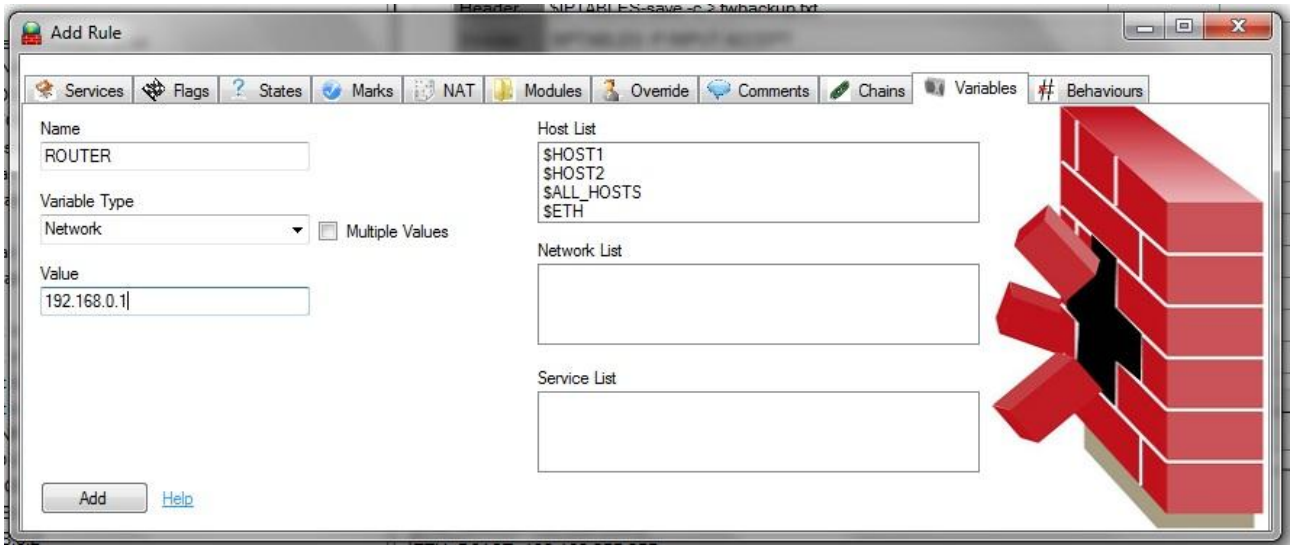
- *Services* – Sono le regole di Policy, quelle che permettono di filtrare in maniera vera e propria i pacchetti. Grazie alle numerosi opzioni disponibili è possibile scrivere regole di complessità elevatissima.
- *Flags* - Permette di scrivere regole relative al parametro –tcp-flags di TCP.
- *States* - Permette di scrivere regole basate sullo stato del connection tracking dei pacchetti in transito.
- *Marks* - Sono le regole relative al controllo del traffico.
- *NAT* - Permette di creare regole di tipo SNAT e DNAT.
- *Modules* – I moduli sono delle regole, solitamente molto lunghe, che vengono utilizzate anche in più Firewall. Tali regole per loro natura sono difficilmente implementabili da Firewall Creator e vengono quindi gestite come vere e proprie liberie da utilizzare all'occorrenza.
- *Override* - E' uno strumento molto potente di Firewall Creator ma anche il più complesso. Sono regole scritte a mano dall'utente, Firewall Creator le interpreta come semplice testo e non effettua modifiche su di esse. E' importante capire che tale funzione può essere utilizzata anche per eseguire comandi Linux altrimenti non implementabili tramite il programma. Per esempio il sistemista potrebbe decidere di riavviare il sistema una volta installato il Firewall, sarà quindi sufficiente aggiungere una regola Override in fondo al Firewall con scritto *reboot*.
- *Comments* - Sono semplici commenti, possono essere utili in caso di regole particolarmente complesse o per organizzare meglio il Firewall.
- *Chains* - Permette di definire nuove catene.
- *Variables* - Vengono chiamate impropriamente regole e in effetti il programma le interpreta come regole vere e proprie. In realtà sono variabili e, come tali, esistono a prescindere da Iptables.
- *Behaviours* – Il loro funzionamento è simile ai moduli ma sono regole parametrizzate. Uno dei più importanti è l'abilitazione dell'IP Forwarding ma la libreria può contenere centinaia di comportamenti diversi.



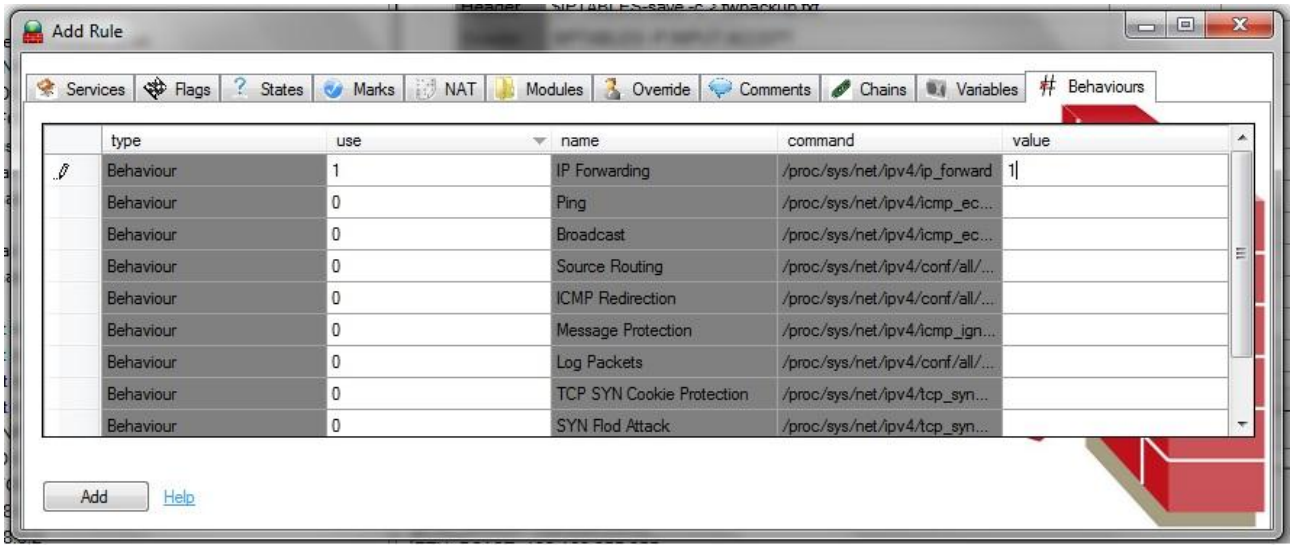
INSERIMENTO DI UNA REGOLA DI TIPO SERVICES



INSERIMENTO DI UAN REGOLA DI TIPO SNAT



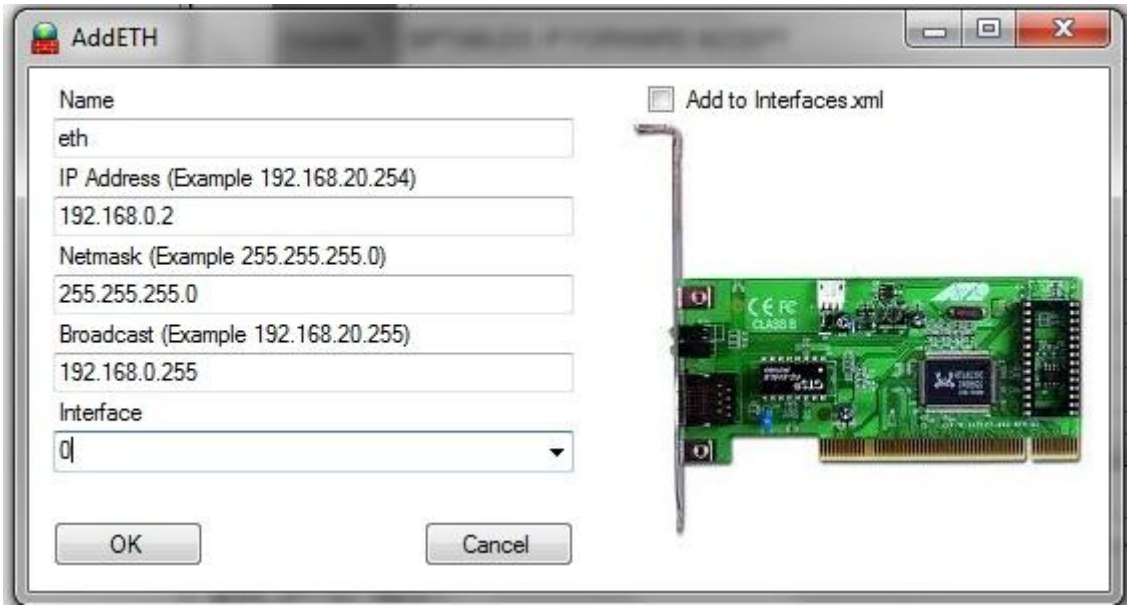
INSERIMENTO DI UNA NUOVA VARIABILE



INSERIMENTO DI UNA REGOLA DI TIPO BEHAVIOURS

ADD ETH INTERFACE

In ogni momento è possibile aggiungere nuove interfacce di rete al Firewall, è possibile definirne il nome, l'indirizzo IP, la Netmask e il Broadcast, oltre ovviamente all'interfaccia vera e propria che può essere un numero, la local (lo), o un parametro definito dall'utente nel file interfaces.xml. Il programma raggruppa automaticamente le regole in base all'interfaccia di provenienza (se disponibile) e permette quindi di avere una visione che distingua in maniera semplice il traffico in entrata e in uscita da una singola scheda di rete.



L'AGGIUNTA DI UNA NUOVA SCHEDA DI RETE AL FIREWALL

CODICE SORGENTE

In questo paragrafo vedremo le sezioni di codice sorgente più importanti, la realizzazione di Firewall Creator ha richiesto la scrittura di decine di migliaia di righe di codice C# e XSL. Riproporre, per esempio, l'intero foglio di stile XSLT richiederebbe centinaia di pagine. Verranno di conseguenza proposti alcuni pezzi relativi alle funzionalità più importanti e i metodi principali. E' importante considerare inoltre che il software non è open source ed essendo destinato alla vendita non può essere rivelato al 100%.

NEW BLANK FIREWALL

Viene creato un nuovo Firewall a partire dal template Blank Firewall che, come si può immaginare dal nome, non è altro che un Firewall vuoto con un semplice commento iniziale. La classe *XmlTextWriter* di *System.Xml* contiene tutti i metodi necessari per la realizzazione di questo metodo.

```
//...
else if (check.CompareTo("Blank Firewall") == 0)
{
      try
      {
            XmlTextWriter newXML = new XmlTextWriter(filename, encoding);
            newXML.WriteStartDocument();
            newXML.WriteComment("XML Firewall made with "+version);
            newXML.WriteComment("Template: " + check);
            newXML.WriteStartElement("FIREWALL");
            newXML.WriteStartElement("ETH");
            newXML.WriteStartAttribute("name");
            newXML.WriteString("ETH");
            newXML.WriteStartAttribute("number");
            newXML.WriteString(ethInterface);
            newXML.WriteStartAttribute("ip");
            newXML.WriteString("");
            newXML.WriteStartAttribute("netmask");
            newXML.WriteString("");
            newXML.WriteStartAttribute("broadcast");
            newXML.WriteString("");
            newXML.WriteStartElement("RULE");
            newXML.WriteStartAttribute("type");
            newXML.WriteString("Comment");
            newXML.WriteStartAttribute("override");
            newXML.WriteString("#Firewall Name: " + firewallname + ", Comments:
" + comment + ", Date: " + date);
            newXML.WriteEndElement();
            newXML.WriteEndElement();
            newXML.WriteEndElement();
            newXML.Close();
      }
      catch (Exception err)
      {
            MessageBox.Show("URL File not valid!");
      }
}
//...
```

ADD NODE

Come visto nel paragrafo relativo all'interfaccia, la modalità principale di gestione del Firewall è la TreeView, il metodo seguente popola tale TreeView in maniera ricorsiva utilizzando le classi di supporto *TreeNode* e *XmlNode*. L'oggetto doc a cui si fa riferimento più volte è un'istanza di *XmlDocument*. E' importante notare il brillante utilizzo della clausola try catch per risolvere il problema dei log. Quando una regola viene loggata infatti il programma crea in automatico due regole Iptables, la regola di base e la regola che ne permette il log. Quest'ultima non deve essere visualizzata nel Firewall (e di conseguenza nel TreeView) perchè è semplicemente un parametro della regola precedente. Quindi quando si incontra una regola di log bisogna saltare a quella successiva. Il programma con un try prova a recuperare l'attributo log, se viene catturata un eccezione è sicuro che si tratti di una regola normale.

```
private void AddNode(XmlNode doc, TreeNode tn, string[] lines)
{
      TreeNode tn1 = null;
      if (doc.HasChildNodes)
      {
            if (doc.Name.CompareTo("ETH") == 0)
            {
                  tn1 =
tn.Nodes.Add(doc.Attributes["name"].Value+"='eth"+doc.Attributes["number"].Value
+"'");
                  tn1.ToolTipText = doc.OuterXml;
                  tn1.ImageIndex = 1;
                  toolStripComboBox2.SelectedItem =
doc.Attributes["number"].Value.ToString();
            }
            if (doc.Name.CompareTo("FIREWALL") == 0)
            {
                  tn1 = tn.Nodes.Add(doc.Name);
                  tn1.ToolTipText = doc.Name;
                  tn1.ImageIndex = 0;
            }
      }
      else
      {
            tn1 = tn.Nodes.Add("", (doc.OuterXml), 1);
            tn1.ToolTipText = doc.OuterXml;
            try
            {
                  string s = doc.Attributes["log"].Value;
                  tn1.Text = lines[i++];
                  i++;
            }
            catch
            {
                  tn1.Text = lines[i++];
            }
            if (doc.Attributes[0].Value.ToString().CompareTo("Header") == 0)
                  tn1.ImageIndex = 2;
            if (doc.Attributes[0].Value.ToString().CompareTo("Service") == 0)
                  tn1.ImageIndex = 3;
            if (doc.Attributes[0].Value.ToString().CompareTo("Comment") == 0)
                  tn1.ImageIndex = 4;
            if (doc.Attributes[0].Value.ToString().CompareTo("Chain") == 0)
                  tn1.ImageIndex = 5;
```

```
            if (doc.Attributes[0].Value.ToString().CompareTo("Override") == 0)
                  tn1.ImageIndex = 6;
            if (doc.Attributes[0].Value.ToString().CompareTo("Module") == 0)
                  tn1.ImageIndex = 8;
            if (doc.Attributes[0].Value.ToString().CompareTo("Flag") == 0)
                  tn1.ImageIndex = 9;
            if (doc.Attributes[0].Value.ToString().CompareTo("State") == 0)
                  tn1.ImageIndex = 10;
            if (doc.Attributes[0].Value.ToString().CompareTo("Variable") == 0)
                  tn1.ImageIndex = 11;
            if (doc.Attributes[0].Value.ToString().CompareTo("SNAT") == 0)
                  tn1.ImageIndex = 13;
            if (doc.Attributes[0].Value.ToString().CompareTo("DNAT") == 0)
                  tn1.ImageIndex = 12;
            if (doc.Attributes[0].Value.ToString().CompareTo("Mark") == 0)
                  tn1.ImageIndex = 14;
            if (doc.Attributes[0].Value.ToString().CompareTo("Behaviour") == 0)
                  tn1.ImageIndex = 15;
      }
      foreach(XmlNode node in doc.ChildNodes)
      {
            AddNode(node, tn1, lines);
      }
}
```

LOAD OPTIONS

All'avvio del programma vengono immediatamente controllate le variabili di ambiente. Queste riguardano principalmente i path delle librerie e del foglio di stile XSL. Il primo avvio permette all'utente di definire questi percorsi (il programma ovviamente non li conosce) e crea un relativo file (come al solito, XML) per salvarli e ricordarli. L'utente, in ogni momento, può comunque modificare tali parametri nelle opzioni.

```
private void loadOptions()
{
      path = System.IO.Directory.GetCurrentDirectory();
      XmlDocument doc = new XmlDocument();
      try
      {
            doc.Load(path + "\\Program.xml");
            XmlNode nodexmlpath = doc.SelectSingleNode("/PROGRAM/XMLPATH");
            XMLPath = nodexmlpath.InnerText;
            XmlNode nodexsltpath = doc.SelectSingleNode("/PROGRAM/XSLTPATH");
            XSLTPath = nodexsltpath.InnerText;
            XmlNode nodetemppath = doc.SelectSingleNode("/PROGRAM/TEMPPATH");
            tempPath = nodetemppath.InnerText;
            XmlNode xmldgview = doc.SelectSingleNode("/PROGRAM/DGVIEW");
            dgviewstyle = xmldgview.InnerText;
            changeDGView(dgviewstyle);
            XmlNode xmlfont = doc.SelectSingleNode("/PROGRAM/FONT");
            fontname = xmlfont.InnerText;
            changeFont(new System.Drawing.Font(fontname, 9));
      }
      catch
      {
            MessageBox.Show("PLEASE READ
CAREFULLY!"+Environment.NewLine+Environment.NewLine+"Since this is the first
time you execute the program you need to specify the XML, XSLT and Temporary
File folders."+Environment.NewLine+Environment.NewLine+"Be sure you have read
and write access permissions to those
folders."+Environment.NewLine+Environment.NewLine+"You can even edit those
folders later through the Preferences Menu.", "", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            FolderBrowserDialog folder = new FolderBrowserDialog();
            folder.Description = "XML Folder";
            folder.ShowDialog();
            XMLPath = folder.SelectedPath;
            FolderBrowserDialog folder2 = new FolderBrowserDialog();
            folder2.Description = "XSLT Folder";
            folder2.ShowDialog();
            XSLTPath = folder2.SelectedPath;
            FolderBrowserDialog folder3 = new FolderBrowserDialog();
            folder3.Description = "Temporary File Folder";
            folder3.ShowDialog();
            tempPath = folder3.SelectedPath;
            createVariables(path);
      }
}

private void createVariables(string path)
{
      XmlTextWriter prog = new XmlTextWriter(path + "\\Program.xml",
System.Text.Encoding.UTF8);
```

```
        prog.WriteStartDocument();
        prog.WriteStartElement("PROGRAM");
        prog.WriteStartElement("XMLPATH");
        prog.WriteString(XMLPath);
        prog.WriteEndElement();
        prog.WriteStartElement("XSLTPATH");
        prog.WriteString(XSLTPath);
        prog.WriteEndElement();
        prog.WriteStartElement("TEMPPATH");
        prog.WriteString(tempPath);
        prog.WriteEndElement();
        prog.WriteStartElement("FONT");
        prog.WriteString("MS Sans Serif");
        prog.WriteEndElement();
        prog.WriteStartElement("DGVIEW");
        prog.WriteString("Fill");
        prog.WriteEndElement();
        prog.WriteEndElement();
        prog.Close();
        fontname = "MS Sans Serif";
        changeFont(new System.Drawing.Font(fontname, 9));
        dgviewstyle = "Fill";
        changeDGView(dgviewstyle);
}
```

UPDATE VARIABLES

Il metodo successivo aggiorna il Firewall una volta che una variabile viene modificata dall'utente. Tramite una query Xpath è possibile ottenere tutti i nodi XML che contengono la variabile modificata e poi, con un semplice ciclo foreach, si effettua l'effettiva modifica.

```
public void updateVariables(XmlDocument doc, string oldname, string name)
{
      string newOld = "$" + oldname;
      string newNew = "$" + name;
      string xpathquery = "/FIREWALL/ETH/RULE/@*[contains(.,'" + newOld +
"')]/..";
      XmlNodeList list = doc.SelectNodes(xpathquery);
      foreach (XmlNode n in list)
      {
            for (int i = 0; i < n.Attributes.Count; i++)
            {
                  if (n.Attributes[i].Value.ToString().CompareTo(newOld) == 0)
                  {
                        n.Attributes[i].Value = newNew;
                        continue;
                  }
                  if (n.Attributes[i].Value.ToString().Contains(newOld))
                  {
                        string oldString = n.Attributes[i].Value.ToString();
                        string newString = oldString.Replace(newOld, newNew);
                        n.Attributes[i].Value = newString;
                        continue;
                  }
            }
      }
}
```

ADD SERVICE

A titolo di esempio viene proposto il codice relativo all'aggiunta di una regola di servizio al Firewall, come abbiamo visto ci sono decine di tipi di regola diversi, questo è uno dei più complessi in quanto tiene conto di numerosi parametri. Come primo problema bisogna quindi gestire i vari controlli sui campi inseriti dall'utente per far si che la regola creata abbia un senso e possa essere di conseguenza generata dal programma. Fin qui il discorso sarebbe abbastanza semplice se solo non fosse che a complicare le cose ci sono le variabili multiple.

Se l'utente inserisce variabili multiple nelle varie opzioni della regola bisogna gestire tale problema. Supponiamo che siano state definite le seguenti variabili come nell'esempio presentato nel precedente paragrafo:

```
HOST1 = "192.168.0.1"
HOST2 = "192.168.0.2"
ssh = "22"
ftp = "21"
ALL_HOSTS = "$HOST1; $HOST2"
ALL_SERVICES = "$ssh; $ftp"
```

Ora l'utente decide di scrivere una regola di servizio che utilizza sia ALL_HOSTS, sia ALL_SERVICES. Osservando attentamente ci si rende conto che l'utilizzo combinato di queste due variabili deve dar luogo alla creazione di ben quattro regole diverse che tengano conto delle relative quattro combinazioni:

```
$HOST1 e $ssh
$HOST1 e $ftp
$HOST2 e $ssh
$HOST2 e $ftp
```

Risolvere tale problema non è stato molto semplice anche perchè le combinazioni possono essere più o meno di quattro. Fortunatamente con un pò di intuito e con l'introduzione di due metodi di supporto si è riusciti brillantemente a trovare un modo per gestire tutte le combinazioni possibili. Di seguito vediamo proprio questa parte con la chiamata ai due metodi di supporto (findBehaviours e createRules) e la chiamata al metodo vero e proprio che crea la regola, ovvero AddService. Per lavorare meglio con i vari campi si è implementata una nuova classe chiamata MyField. MyFields invece è un'istanza della classe ArrayList dichiarata a livello globale.

```
private void buttonService_Click(object sender, EventArgs e)
{
      string logoptions = "";
      string rejectoptions = "";
      string protocol = "";
      string porttype = "";
      string porttype2 = "";
      string table = "";
      if ((comboBox1.SelectedItem != null) && (comboBox3.SelectedItem != null))
      {
            if (checkBox1.Checked)
            {
                  CheckedListBox.CheckedItemCollection coll =
checkedListBox1.CheckedItems;
                  int i = coll.Count;
```

```
                for (int j = 0; j < i; j++)
                {
                        logoptions = logoptions + " " + coll[j];
                }
            }
            if (comboBoxServiceTable.SelectedItem == null)
                table = "";
            else
                table = comboBoxServiceTable.SelectedItem.ToString();
            if (comboBox7.SelectedItem == null)
                rejectoptions = "";
            else
                rejectoptions = comboBox7.SelectedItem.ToString();
            if (comboBoxServiceProtocol.SelectedItem == null)
            {
                protocol = "";
            }
            else
            {
                protocol = comboBoxServiceProtocol.SelectedItem.ToString();
                if (comboBoxServicePort1.SelectedItem == null)
                    porttype = "";
                else
                    porttype = comboBoxServicePort1.SelectedItem.ToString();
                if (comboBoxServicePort2.SelectedItem == null)
                    porttype2 = "";
                else
                    porttype2 =
comboBoxServicePort2.SelectedItem.ToString();
            }
            ArrayList rules = new ArrayList();
            ArrayList rule = new ArrayList(5);
            rule.Add(null);
            rule.Add(null);
            rule.Add(null);
            rule.Add(null);
            rule.Add(null);
            rule.Add(null);
            MyFields = new ArrayList();
            bool[] behaviours = findBehaviours(new string[] {
textBoxSource.Text, textBoxDestination.Text, textBoxServiceI.Text,
textBoxServiceO.Text, textBoxServicePort1.Text, textBoxServicePort2.Text });
            createRules(behaviours, new string[] { textBoxSource.Text,
textBoxDestination.Text, textBoxServiceI.Text, textBoxServiceO.Text,
textBoxServicePort1.Text, textBoxServicePort2.Text });
            MyField source = (MyField)MyFields[0];
            MyField destination = (MyField)MyFields[1];
            MyField incoming = (MyField)MyFields[2];
            MyField outgoing = (MyField)MyFields[3];
            MyField port1 = (MyField)MyFields[4];
            MyField port2 = (MyField)MyFields[5];
            string sstuck = null;
            string dstuck = null;
            string istuck = null;
            string ostuck = null;
            string p1stuck = null;
            string p2stuck = null;
            foreach (string s in source.getChilds() ?? new ArrayList { null })
            {
```

```
                    if ((source.getChilds() == null))
                    {
                          if (source.getName() == null)
                          {
                                rule[0] = null;
                          }
                          else
                          {
                                rule[0] = source.getName();
                                sstuck = (string)rule[0];
                          }
                    }
                    else
                    {
                          rule[0] = s;
                    }
                    foreach (string d in destination.getChilds() ?? new ArrayList
{ null })
                    {
                          if ((destination.getChilds() == null))
                          {
                                if (destination.getName() == null)
                                {
                                      rule[1] = null;
                                }
                                else
                                {
                                      rule[1] = destination.getName();
                                      dstuck = (string)rule[1];
                                }
                          }
                          else
                          {
                                rule[1] = d;
                          }
                          foreach (string i in incoming.getChilds() ?? new
ArrayList { null })
                          {
                                if ((incoming.getChilds() == null))
                                {
                                      if (incoming.getName() == null)
                                      {
                                            rule[2] = null;
                                      }
                                      else
                                      {
                                            rule[2] = incoming.getName();
                                            istuck = (string)rule[2];
                                      }
                                }
                                else
                                {
                                      rule[2] = i;
                                }
                                foreach (string o in outgoing.getChilds() ?? new
ArrayList { null })
                                {
                                      if ((outgoing.getChilds() == null))
                                      {
```

```
                                        if (outgoing.getName() == null)
                                        {
                                              rule[3] = null;
                                        }
                                        else
                                        {
                                              rule[3] = outgoing.getName();
                                              ostuck = (string)rule[3];
                                        }
                                  }
                                  else
                                  {
                                        rule[3] = o;
                                  }
                                  foreach (string p1 in port1.getChilds() ??
new ArrayList { null })
                                  {
                                        if ((port1.getChilds() == null))
                                        {
                                              if (port1.getName() == null)
                                              {
                                                    rule[4] = null;
                                              }
                                              else
                                              {
                                                    rule[4] = port1.getName();
                                                    p1stuck = (string)rule[4];
                                              }
                                        }
                                        else
                                        {
                                              rule[4] = p1;
                                        }
                                        foreach (string p2 in
port2.getChilds() ?? new ArrayList { null })
                                        {
                                              if ((port2.getChilds() == null))
                                              {
                                                    if (port2.getName() ==
null)
                                                    {
                                                          rule[5] = null;
                                                    }
                                                    else
                                                    {
                                                          rule[5] =
port2.getName();
                                                          p2stuck =
(string)rule[5];
                                                    }
                                              }
                                              else
                                              {
                                                    rule[5] = p2;
                                              }
                                              rules.Add(rule);
                                              rule = new ArrayList(5);
                                              if (sstuck != null)
                                                    rule.Add(sstuck);
```

```
                                                else
                                                      rule.Add(s);
                                                if (dstuck != null)
                                                      rule.Add(dstuck);
                                                else
                                                      rule.Add(d);
                                                if (istuck != null)
                                                      rule.Add(istuck);
                                                else
                                                      rule.Add(i);
                                                if (ostuck != null)
                                                      rule.Add(ostuck);
                                                else
                                                      rule.Add(o);
                                                if (p1stuck != null)
                                                      rule.Add(p1stuck);
                                                else
                                                      rule.Add(p1);
                                                if (p2stuck != null)
                                                      rule.Add(p2stuck);
                                                else
                                                      rule.Add(p2);
                                          }
                                    }
                              }
                        }
                  }
            }
            string chain = comboBox1.SelectedItem.ToString();
            string policy = comboBox3.SelectedItem.ToString();
            bool tolog = checkBox1.Checked;
            bool denys = checkBoxSourceD.Checked;
            bool denyd = checkBoxDestionationD.Checked;
            foreach (ArrayList r in rules)
            {
                  AddService(table, chain, policy, tolog, (string)r[0],
(string)r[1], denys, denyd, logoptions, rejectoptions, protocol, (string)r[2],
(string)r[3], porttype, (string)r[4], porttype2, (string)r[5]);
            }
            test.refreshAll();
            MyFields = null;
      }
      else
      {
            MessageBox.Show("Missing Parameter!");
      }
}
```

FIND BEHAVIOURS

Questo metodo restituisce un array di booleani dove ogni campo indica il comportamento della relativa variabile, ovvero se è una variabile normale o di tipo multiplo. Questo è molto importante per capire quali saranno i vari accoppiamenti che si dovranno generare. E' chiaro che se non ci sono variabili multiple l'accoppiamento è uno solo e, di conseguenza, verrà generata una e una sola regola.

```
private bool[] findBehaviours(string[] array)
{
	bool[] behaviours = new bool[6];
	int i = 0;
	foreach (string v in array)
	{
		bool find = false;
		if ((v == null) || (v.CompareTo("") == 0))
		{
			behaviours[i] = false;
			i++;
			continue;
		}
		else
		{
			string temp = "";
			foreach (char c in v)
			{
				if (c.CompareTo('$') == 0)
				{
				}
				else
				{
					temp = temp + c;
				}
			}
			XmlDocument doc = new XmlDocument();
			doc.Load(test.getCurrentFilename());
			XmlNodeList vars =
doc.SelectNodes("/FIREWALL/ETH/RULE[@type='Variable']");
			foreach (XmlNode n in vars)
			{
				if
(n.Attributes["name"].Value.ToString().CompareTo(temp) == 0)
				{
					if
(n.Attributes["behaviour"].Value.ToString().CompareTo("multiple") == 0)
					{
						behaviours[i] = true;
						i++;
						find = true;
						break;
					}
					else
					{
						behaviours[i] = false;
						i++;
						find = true;
						break;
					}
				}
			}
```

```
                    if (!find)
                          i++;
              }
        }
        return behaviours;
}
```

CREATE RULES (GETVALUES)

Ora che conosciamo le variabili in gioco e il loro comportamento è il momento di creare le combinazioni, prima però bisogna non solo organizzare queste variabili ma anche ricavare le variabili singole a partire da quelle multiple. Il separatore di default è il punto e virgola quindi tutto sta nell'elaborare delle stringhe. Ci serviremo di un ulteriore metodo di supporto, ovvero getValues.

```
private void createRules(bool[] behaviours, string[] array)
{
	int i = 0;
	foreach (string v in array)
	{
		if ((v == null) || (v.CompareTo("") == 0))
		{
			MyField newfield = new MyField(null, null);
			MyFields.Add(newfield);
			i++;
		}
		else
		{
			if (behaviours[i] == true)
			{
				MyField newfield = new MyField(v, new ArrayList());
				ArrayList vars = getValues(v);
				foreach (string w in vars)
				{
					newfield.AddChild(w);
				}
				MyFields.Add(newfield);
				i++;
			}
			else
			{
				MyFields.Add(new MyField(v, null));
				i++;
			}
		}
	}
}
```

```
private ArrayList getValues(string varToCheck)
{
	ArrayList toReturn = new ArrayList();
	string temp = "";
	foreach (char c in varToCheck)
	{
		if (c.CompareTo('$') == 0)
		{
		}
		else
		{
			temp = temp + c;
		}
	}
	varToCheck = temp;
	string value = "";
	XmlDocument doc = new XmlDocument();
```

```
        try
        {
                doc.Load(test.getCurrentFilename());
                XmlNodeList chains =
doc.SelectNodes("/FIREWALL/ETH/RULE[@type='Variable']");
                foreach (XmlNode CHAIN in chains)
                {
                        if (CHAIN.Attributes["name"].Value.CompareTo(varToCheck) ==
0)
                        {
                                value = CHAIN.Attributes["value"].Value.ToString();
                        }
                }
                string varToAdd = "";
                foreach (char c in value)
                {
                        if (c.CompareTo(';') == 0)
                        {
                                toReturn.Add(varToAdd);
                                varToAdd = "";
                                continue;
                        }
                        if (c.CompareTo(' ') == 0)
                        {
                                continue;
                        }
                        else
                        {
                                varToAdd = varToAdd + c;
                        }
                }
                return toReturn;
        }
        catch (Exception err)
        {
                MessageBox.Show(err.Message);
                return null;
        }
}
```

ADD SERVICE (EFFETTIVO)

Ora che abbiamo tutte le varie combinazioni è il momento di creare, finalmente, le regole XML. Faremo uso di molte classi della libreria *System.Xml* come *XmlElement*, *XmlNode* e *XmlDocument*. Come si può notare all'inizio del codice, il programma mantiene sempre una copia di backup del Firewall in caso si verifichi qualche problema o in caso l'utente prema CTRL+Z per annullare l'operazione appena effettuata.

Una volta caricato il Firewall attuale creiamo al suo interno un nuovo elemento e cominciamo ad inserire i vari valori con il metodo SetAttribute. Tale valore viene aggiunto all'interfaccia di rete attualmente in utilizzo all'interno del Firewall, se l'interfaccia selezionata non è presente nel Firewall la regola viene aggiunta all'interfaccia di default (solitamente eth0).

Alla fine il metodo salva il Firewall e chiama il metodo refreshAll per aggiornare le varie visuali con la nuova regola inserita.

```
private void AddService(string table, string chain, string policy, bool log,
string source, string destination, bool sourceD, bool destinationD, string
logoptions, string rejectwith, string protocol, string incoming, string
outgoing, string porttype1, string port1, string porttype2, string port2)
{
      if (!test.getWorking())
      {
            MessageBox.Show("You must create or open a new Firewall in order to
add Rules");
      }
      else
      {
      XmlDocument toBackup = new XmlDocument();
      toBackup.Load(test.getCurrentFilename());
      File.SetAttributes(test.getBackupFileName(), FileAttributes.Normal);
      toBackup.Save(test.getBackupFileName());
      File.SetAttributes(test.getBackupFileName(), FileAttributes.Hidden);
      test.setButtonUndo();
      bool foundETH = false;
      XmlDocument doc = new XmlDocument();
      doc.Load(test.getCurrentFilename());
      XmlNode newNode = doc.DocumentElement;
      XmlElement rule = doc.CreateElement("RULE");
      rule.SetAttribute("type", "Service");
      rule.SetAttribute("chain", chain);
            if ((table != "") && (table != null))
            {
                  rule.SetAttribute("table", table);
            }
            if (policy.CompareTo("REJECT") == 0)
            {
                  rule.SetAttribute("policy", policy);
                  if (rejectwith != "")
                        rule.SetAttribute("rejectwith", rejectwith);
            }
            else
                  rule.SetAttribute("policy", policy);
            if (log == true)
            {
                  rule.SetAttribute("log", "to log");
                  rule.SetAttribute("logoptions", logoptions);
```

```
            }
            if ((source != null) && (source != ""))
            {
                  rule.SetAttribute("source", source);
                  if (sourceD)
                        rule.SetAttribute("denysource", "to deny");
            }
            if ((destination != null) && (destination != ""))
            {
                  rule.SetAttribute("destination", destination);
                  if (destinationD)
                        rule.SetAttribute("denydestination", "to deny");
            }
            if ((protocol != null) && (protocol != ""))
            {
                  rule.SetAttribute("protocol", protocol);
            }
            if ((incoming != null) && (incoming != ""))
            {
                  rule.SetAttribute("input", incoming);
            }
            if ((outgoing != null) && (outgoing != ""))
            {
                  rule.SetAttribute("output", outgoing);
            }
            if ((porttype1 != null) && (porttype1 != ""))
            {
                  rule.SetAttribute("porttype", porttype1);
                  rule.SetAttribute("port", port1);
            }
            if ((porttype2 != null) && (porttype2 != ""))
            {
                  rule.SetAttribute("porttype2", porttype2);
                  rule.SetAttribute("port2", port2);
            }
            XmlNodeList list = doc.SelectNodes("/FIREWALL/ETH");
            foreach (XmlNode node in list)
            {
                  if
(node.Attributes["number"].Value.ToString().CompareTo(test.getWorkingETH()) ==
0)
                  {
                        newNode = node;
                        foundETH = true;
                        break;
                  }
            }
            if (edit)
            {
                  doEditNode(doc, rule);
            }
            else
            {
                  if (foundETH)
                  {
                        newNode.AppendChild(rule);
                  }
                  else
                  {
```

```
                        MessageBox.Show("Current firewall does not have ETH" +
test.getWorkingETH() + " , Rule will be added to default ETH");
                        newNode.LastChild.AppendChild(rule);
                  }
                  doc.Save(test.getCurrentFilename());
            }
            test.refreshAll();
            if (edit)
                  this.Close();
      }
}
```

XSL TRANSFORM

Il breve frammento di codice qui proposto mostra il momento in cui il Firewall XML viene tradotto in linguaggio Iptables Linux vero e proprio. Il risultato viene salvato su un file di testo. La classe *XslTransform* è considerata obsoleta dal Framework .NET ma, stranamente, risulta decisamente più veloce della nuova versione proposta. Per questo motivo si è scelto di utilizzare la vecchia versione della classe.

```
XslTransform transformation = new XslTransform();
transformation.Load(XSLTPath + "\\firewallxslt.xslt");
transformation.Transform(filename, XSLTPath + "\\transformation.txt");
```

XSLT STYLESHEET

Per completare la panoramica sul codice sorgente del programma ci manca da analizzare solo il foglio di stile XSLT. Purtroppo tale foglio di stile è letteralmente enorme, basti pensare che uno solo dei suoi template occuperebbe oltre 250 pagine. La complessità di questo foglio di stile è alla base della potenza di Firewall Creator del resto. Come abbiamo visto il programma crea dei Firewall XML e, quando richiesto, effettua una trasformazione in linguaggio Iptables Linux utilizzando proprio tale foglio di stile. Ma come avviene la trasformazione? Il programma ha generato solo righe di codice XML che non sono nient'altro che un insieme di attributi.

La soluzione è semplice ma, dovendo tenere conto di migliaia di combinazioni diverse, è molto lunga e ha richiesto numerose ore di lavoro. In pratica il foglio di stile contiene svariati template, uno per ogni tipo di regola (Services, NAT, Behaviours...) e ogni template trasforma gli attributi in una riga di testo interpretabile da un sistema Linux. Prima di vedere qualcosa di più complesso partiamo dall'esempio più banale in assoluto.

```
<xsl:if test="@type='Chain'">
      <!--REGOLA DI TIPO CHAIN-->$IPTABLES -N <xsl:value-of
select="@chain"/><xsl:text>&#13;&#10;</xsl:text>
</xsl:if>
```

Il template qui sopra trasforma una regola XML di tipo Chain (ovvero le catene) in linguaggio Iptables Linux. Vediamo come funziona nel dettaglio.

Le regole di tipo Chain hanno un solo parametro che altro non è che il nome che si vuole dare alla catena, in questo caso tale valore è contenuto nel campo chain del nodo XML. Tramite value-of è possibile estrapolare questo valore e aggiungerlo a un piccolo prefisso che è in questo caso il comando iptables seguito da –N. La cosa non dovrebbe stupirci, come visto nel paragrafo dedicato a Iptables il comando –N serve proprio a creare una nuova regola.

Se il nodo in questione contiene il valore NEWCHAIN il foglio di stile non farà altro che salvare sul risultato la seguente riga di codice.

```
$IPTABLES –N NEWCHAIN
```

L'esempio presentato sembra stupido ma all'aumentare dei parametri la complessità del template aumenta in maniera esponenziale. In realtà questo si traduce poi in centinaia di if annidati, uno per ogni combinazione diversa.

Di seguito è possibile vedere il foglio di stile "compresso", ovvero senza ingrandire i vari template.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
      <xsl:output method="text"/>
      <xsl:template match="text()"/>
      <xsl:template match="/">
            <xsl:for-each select="/FIREWALL/ETH">...</xsl:for-each>
            echo
.....
....
            <xsl:text></xsl:text>
            <xsl:for-each select="/FIREWALL/ETH/RULE">
                  <xsl:if test="@type='Header'">...</xsl:if>
                  <xsl:if test="@type='Override'">...</xsl:if>
                  <xsl:if test="@type='Module'">...</xsl:if>
```

```
                    <xsl:if test="@type='Behaviour'">...</xsl:if>
                    <xsl:if test="@type='Variable'">...</xsl:if>
                    <xsl:if test="@type='Mark'">...</xsl:if>
                    <xsl:if test="@type='Comment'">...</xsl:if>
                    <xsl:if test="@type='DNAT'">...</xsl:if>
                    <xsl:if test="@type='SNAT'">...</xsl:if>
                    <xsl:if test="@type='Chain'">...</xsl:if>
                    <xsl:if test="@type='Service'">...</xsl:if>
                    <xsl:if test="@type='Flag'">...</xsl:if>
                    <xsl:if test="@type='State'">...</xsl:if>
              </xsl:for-each>
       </xsl:template>
</xsl:stylesheet>
```

Nell'esempio qui sotto si è selezionato uno solo delle centinaia di if che compongono il template SNAT. In questo caso particolare è presente il parametro protocol e non sono presenti nè il campo source, nè il campo destination. Nemmeno il campo porttype è presente,

```
<xsl:if test="@type='SNAT'">
       <xsl:if test="boolean(string (@protocol))">
              <xsl:if test="not(boolean(string (@destination))) and
not(boolean(string (@source)))">
                    <xsl:if test="not(boolean(string (@porttype)))">
                           $IPTABLES -t nat -A POSTROUTING
                           -o <xsl:value-of select="@output"/>
                           -p <xsl:value-of select="@protocol"/>
                           -j SNAT --to-source <xsl:value-of
select="@rangeaddresses"/>
                           <xsl:text>&#13;&#10;</xsl:text>
                    </xsl:if>
              </xsl:if>
       </xsl:if>
</xsl:if>
```

In quest'altro esempio invece sono presenti sia source che destination ed è presente anche la clausola not (deny). Il campo porttype è presente, il che aumenta ulteriormente la complessità. E' interessante notare come l'aggiunta e la rimozione di parametri abbia cambiato totalmente la sintassi finale della regola.

```
<xsl:if test="@type='SNAT'">
       <xsl:if test="boolean(string (@protocol))">
              <xsl:if test="boolean(string (@destination)) and boolean(string
(@source))">
                    <xsl:if test="boolean(string (@denydestination)) and
boolean(string (@denysource))">
                           <xsl:if test="boolean(string (@porttype))">
                           $IPTABLES -t nat -A POSTROUTING -o <xsl:value-of
select="@output"/> -p <xsl:value-of
select="@protocol"/><xsl:text>&#32;</xsl:text><xsl:value-of
select="@porttype"/><xsl:text>&#32;</xsl:text><xsl:value-of
select="@port"/><xsl:if test="boolean(string (@porttype2))">
                                  <xsl:text>&#32;</xsl:text>
                                  <xsl:value-of select="@porttype2"/>
                                  <xsl:text>&#32;</xsl:text>
                                  <xsl:value-of select="@port2"/>
                                  </xsl:if> -s ! <xsl:value-of select="@source"/> -d
! <xsl:value-of select="@destination"/> -j SNAT --to-source <xsl:value-of
```

```
select="@rangeaddresses"/><xsl:text>&#13;&#10;</xsl:text>
                        </xsl:if>
                    </xsl:if>
                </xsl:if>
            </xsl:if>
        </xsl:if>
```

FIREWALL DI ESEMPIO

Dopo aver visto tutti i particolari relativi alla creazione di un Firewall è ora di vedere finalmente il risultato. Di seguito viene proposto uno dei tanti Firewall che è possibile creare con questo applicativo, la struttura come si vede rispecchia quanto detto finora e, anche per un sistemista alle prime armi, è molto facile vedere all'interno lo scheletro di un Firewall Iptables Linux.

```
<?xml version="1.0" standalone="yes" ?>
<FIREWALL>
  <ETH name="ETH" number="0" ip="192.168.0.1" netmask="255.255.255.0"
broadcast="192.168.0.255">
     <RULE type="Comment" override="#My Firewall" />
     <RULE type="Behaviour" name="IP Forwarding"
command="/proc/sys/net/ipv4/ip_forward" value="0" />
     <RULE type="Behaviour" name="Ping"
command="/proc/sys/net/ipv4/icmp_echo_ignore_all" value="0" />
     <RULE type="Header" override="$IPTABLES-save -c > fwbackup.txt" />
     <RULE type="Header" override="$IPTABLES -P INPUT ACCEPT" />
     <RULE type="Header" override="$IPTABLES -P OUTPUT ACCEPT" />
     <RULE type="Header" override="$IPTABLES -P FORWARD ACCEPT" />
     <RULE type="Header" override="$IPTABLES --flush" />
     <RULE type="Header" override="$IPTABLES -t nat --flush" />
     <RULE type="Header" override="$IPTABLES -t mangle --flush" />
     <RULE type="Header" override="$IPTABLES -X" />
     <RULE type="Header" override="$IPTABLES -t nat -X" />
     <RULE type="Header" override="$IPTABLES -t mangle -X" />
     <RULE type="Header" override="$IPTABLES -F" />
     <RULE type="Header" override="$IPTABLES -F -t nat" />
     <RULE type="Header" override="$IPTABLES -F -t mangle" />
     <RULE type="Header" override="$IPTABLES -X -t nat" />
     <RULE type="Header" override="$IPTABLES -X -t mangle" />
     <RULE type="Header" override="$IPTABLES -F INPUT" />
     <RULE type="Header" override="$IPTABLES -F OUTPUT" />
     <RULE type="Header" override="$IPTABLES -F FORWARD" />
     <RULE type="Variable" name="HOST1" value="192.168.0.2" kind="Host"
behaviour="normal" />
     <RULE type="Variable" name="HOST2" value="192.168.0.3" kind="Host"
behaviour="normal" />
     <RULE type="Variable" name="ALL_HOSTS" value="$HOST1; $HOST2;" kind="Host"
behaviour="multiple" />
     <RULE type="Variable" name="ssh" value="22" kind="Service"
behaviour="normal" />
     <RULE type="Variable" name="ftp" value="21" kind="Service"
behaviour="normal" />
     <RULE type="Variable" name="ALL_SERVICES" value="$ssh; $ftp;"
kind="Service" behaviour="multiple" />
     <RULE type="Service" chain="INPUT" policy="ACCEPT" source="$HOST1"
protocol="tcp" porttype="--sport" port="$ssh" />
     <RULE type="Service" chain="INPUT" policy="ACCEPT" source="$HOST1"
protocol="tcp" porttype="--sport" port="$ftp" />
     <RULE type="Service" chain="INPUT" policy="ACCEPT" source="$HOST2"
protocol="tcp" porttype="--sport" port="$ssh" />
     <RULE type="Service" chain="INPUT" policy="ACCEPT" source="$HOST2"
protocol="tcp" porttype="--sport" port="$ftp" />
  </ETH>
</FIREWALL>
```

CODICE LATO SERVER

Il programma, per gestire l'installazione remota, necessita di un piccolo demone lato server che permetta la connessione del programma principale e l'installazione del Firewall. Come descritto in precedenza il programma lato server gira, ovviamente, su piattaforma Linux ed è stato realizzato in linguaggio C# tramite Mono Develop. Viene riproposto il codice integrale in quanto relativamente breve.

Una volta avviato, il server configura i parametri e si mette in ascolto sulla porta 55555 creando un nuovo thread. E' importante che tale porta resti abilitata prima durante e dopo l'installazione remota come visto nel precedente paragrafo, per assicurare il corretto svolgimento delle operazioni.

Il server risponde a numerosi comandi come ad esempio DISCONNECT o SEND LIST che permettono rispettivamente di chiudere la connessione e inviare al client la configurazione del Firewall corrente e/o appena installato.

Una volta terminate le operazioni di installazione (solo in questo caso), il programma di default effettua un tentativo di Ping alla macchina client e, in caso l'esito sia negativo, richiede all'utente se vuole effettuare il Restore del Firewall precedente.

NOTE PROGETTUALI

- La porta di default è attualmente quella utilizzata per testare il corretto funzionamento del programma e potrebbe non rispecchiare quella della release commerciale.
- Nel codice evidenziato qui sotto viene pingato a titolo di esempio il server maya.ngi.it in quanto non è ancora possibile (per ovvi motivi) conoscere l'indirizzo IP del sistemista.

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Text;
using System.Threading;

namespace FWCreatorServer
{
	class MainClass
	{
		private TcpListener tcpListener;
		private Thread listenThread;
		private string username;
		private IPEndPoint ip;
		private bool install;

		public MainClass()
		{
			install = false;
			tcpListener = new TcpListener(IPAddress.Any, 55555);
			listenThread = new Thread(new ThreadStart(listenForClients));
			listenThread.Start();
		}

		public static void Main (string[] args)
		{
			MainClass man = new MainClass();
			man.Configure();
		}

		public void Configure()
		{
			Console.WriteLine ("Firewall Creator Server is Listening");
			Console.WriteLine ("Created by raz3r @ WORKTEAM SRL");
			Console.WriteLine ("daniele.raz3r@gmail.com");
			Console.WriteLine ("www.workteam.it");
		}

		public void listenForClients ()
		{
			tcpListener.Start();
			while (true)
			{
				TcpClient client = tcpListener.AcceptTcpClient();
				ip = (IPEndPoint)client.Client.RemoteEndPoint;
				Console.WriteLine("Client "+ip.ToString()+"
Connected!");
				Console.WriteLine("Server waiting for commands");
				Thread clientThread = new Thread(new
ParameterizedThreadStart(HandleClient));
				clientThread.Start(client);
			}
		}

		public void HandleClient(object client)
		{
```

```
                    bool toExecute = false;
                    string vars = "";
                    string[] arrayVars = new string[] {};
                    vars = vars + "";
                    TcpClient tcpClient = (TcpClient)client;
                    NetworkStream clientStream = tcpClient.GetStream();
                    clientStream.ReadTimeout = 50000;
                    clientStream.WriteTimeout = 50000;
                    byte[] message = new byte[32767];
                    int bytesread;
                    while (true)
                    {
                          bytesread = 0;
                          try
                          {
                                bytesread = clientStream.Read(message, 0, 4096);
                          }
                          catch
                          {
                                break;
                          }
                          if (bytesread == 0)
                                continue; //No variables
                          install = true;
                          ASCIIEncoding encoding = new ASCIIEncoding();
                          Console.WriteLine("Command Received!");
                          Console.WriteLine("Executing:
"+encoding.GetString(message, 0, bytesread));
                          System.Diagnostics.Process proc = new
System.Diagnostics.Process();
                          proc.StartInfo.UseShellExecute = false;
                          proc.StartInfo.RedirectStandardError = true;
                          proc.StartInfo.RedirectStandardOutput = true;
                          proc.StartInfo.FileName = "sh";
                          string toWrite = encoding.GetString(message, 0,
bytesread);
                          //bool found = findIPTables(toWrite);
                          if (toWrite.CompareTo("DISCONNECT") == 0)
                          {
                                install = false;
                                break;
                          }
                          else if (toWrite.CompareTo("GET FW") == 0)
                          {
                                File.WriteAllText("/tmp/temp.sh", "iptables-
save");
                                proc.StartInfo.Arguments = "/tmp/temp.sh";
                                proc.Start();
                                string toSendList =
proc.StandardOutput.ReadToEnd();
                                byte[] bufferList;
                                bufferList = encoding.GetBytes(toSendList);
                                clientStream.Write(bufferList, 0,
bufferList.Length);
                                proc.Close();
                                continue;
                          }
                          else if (toWrite.CompareTo("INSTALL COMPLETED") == 0)
                          {
```

```
                                    install = true;
                                    break;
                              }
                              else if (toWrite.CompareTo("VARIABLES") == 0)
                              {
                                    byte[] cmdOK = encoding.GetBytes("COMMAND
RECEIVED");
                                    clientStream.WriteTimeout = 50000;
                                    clientStream.Write(cmdOK, 0, cmdOK.Length);
                                    clientStream.Flush();
                                    byte[] buffervar = new byte[32767];
                                    clientStream.ReadTimeout = 50000;
                                    clientStream.Read(buffervar, 0, 32767);
                                  vars = encoding.GetString(buffervar);
                                    arrayVars = vars.Split('/');
                                    byte[] varOK = encoding.GetBytes("VARIABLES
RECEIVED");
                                    clientStream.WriteTimeout = 50000;
                                    clientStream.Write(varOK, 0, varOK.Length);
                                    clientStream.Flush();
                                    continue;
                              }
                              else if (toWrite.CompareTo("SEND LIST") == 0) //List
case!!!
                              {
                                    File.WriteAllText("/tmp/temp.sh",
"IPTABLES=iptables; $IPTABLES -L; sudo $IPTABLES -L -t nat");
                                    proc.StartInfo.Arguments = "/tmp/temp.sh";
                                    proc.Start();
                                    string toSendList =
proc.StandardOutput.ReadToEnd();
                                    byte[] bufferList;
                                    bufferList = encoding.GetBytes(toSendList);
                                    clientStream.Write(bufferList, 0,
bufferList.Length);
                                    proc.Close();
                                    continue;
                              }
                              if (install)
                              {
                                    string varsToWrite = "";
                                    toExecute = true;
                                    for (int i = 0; i < arrayVars.Length-1; i++)
                                    {
                                          varsToWrite = varsToWrite + arrayVars[i] + "
";
                                    }
                                    Console.WriteLine(varsToWrite);
                                    if (varsToWrite.CompareTo("") == 0)
                                          File.WriteAllText("/tmp/temp.sh",
"IPTABLES=iptables; "+toWrite);
                                    if (varsToWrite.CompareTo("") != 0)
                                          File.WriteAllText("/tmp/temp.sh",
varsToWrite + "IPTABLES=iptables;"+toWrite);
                              }
                              if (toExecute)
                              {
                                    proc.StartInfo.Arguments = "/tmp/temp.sh";
                                    proc.Start(); //EXECUTE COMMAND
```

```
                                    string toSend = proc.StandardError.ReadToEnd();
//REDIRECT STANDARDERROR
                                    byte[] buffer;
                                    buffer = encoding.GetBytes("Error: "+toSend);
                                    clientStream.Write(buffer, 0, buffer.Length);
//SEND STANDARDERROR
                                    proc.Close();
                                    toExecute = false;
                                    continue;
                              }
                              Console.WriteLine("Command not recognized! Network
error?");
                        }
                        clientStream.Close();
                        tcpClient.Close(); //Close current connection and disconnect
or maybe connection LOST
                        Console.WriteLine("Client Disconnected");
                        if (install)
                        {
                              if (!tryPing())
                              {
                                    //ASK USER TO RESTORE FIREWALL
                                    Console.WriteLine("Firewall didn't install
correctly, do you want to restore your previous firewall? Yes (y) / No (n)");
                                    string answer = Console.ReadLine();
                                    if (answer.CompareTo("y") == 0)
                                    {
                                          //DO RESTORE
                                          Console.WriteLine("Restoring Firewall...");
                                          File.WriteAllText("/tmp/temp.sh", "sudo
iptables-restore -c < fwbackup.txt");
                                          System.Diagnostics.Process proc = new
System.Diagnostics.Process();
                                          proc.StartInfo.UseShellExecute = false;
                                          proc.StartInfo.FileName = "sh";
                                          proc.StartInfo.Arguments = "/tmp/temp.sh";
                                          if (proc.Start())
                                                Console.WriteLine("Firewall
Restored!");
                                          else
                                                Console.WriteLine("Program was unable
to restore previous Firewall (maybe there is no available backup?)");
                                    }
                              }
                              else
                              {
                                    Console.WriteLine("New Firewall installed
correctly! Enjoy new configuration!");
                              }
                        }
                  }

                  private bool tryPing()
                  {
                        //If I can ping client machine everything should be fine
                        Ping p = new Ping();
                        Console.WriteLine("Server will now try to ping a machine over
the Internet");
                        Console.WriteLine("WARNING: If you've disabled ICMP the
```

```
following test won't work!");
                  Console.WriteLine("Testing ping on maya.ngi.it...");
            PingReply reply = p.Send("88.149.128.3");
                  if (reply.Status == IPStatus.Success)
                  {
                        Console.WriteLine("RTT:
"+reply.RoundtripTime.ToString()+" TTL: "+reply.Options.Ttl);
                        return true;
                  }
                  else
                  {
                        Console.WriteLine("Cannot ping client, reason:
"+reply.Status.ToString());
                        return false;
                  }
            }
      }
}
```


CONCLUSIONI E CONSIDERAZIONI

La realizzazione del programma si è dimostrata essere un vero e proprio successo. Tutti gli obiettivi sono stati raggiunti e si è andati ben oltre le aspettative iniziali. E' sufficiente considerare il fatto che la release attuale del software permette di creare migliaia di regole Iptables diverse, un risultato assolutamente inaspettato. Quello che doveva essere un semplice tirocinio di apprendimento si è dimostrato essere in realtà una solida base per progetti assai più importanti. Personalmente spero che Firewall Creator sia solo il primo di una lunga serie di applicativi, realizzare questo progetto infatti ha confermato le mie reali capacità e mi ha insegnato a spingermi sempre oltre. Nell'informatica niente è impossibile, è sufficiente la passione e la voglia di apprendere, i risultati poi arrivano di conseguenza. In un mondo in continua evoluzione però è importante restare al passo, non bisogna solo aggiornare noi stessi, bisogna aggiornare anche i software ed è in questo che Firewall Creator riesce brillantemente. La sua struttura dinamica lo rende pronto per i Firewall del futuro e, in un mondo in cui la sicurezza informatica è essenziale, questo non può che essere un eccellente traguardo per il programma. E' opportuno infine menzionare che il software ha partecipato all'edizione 2010 del concorso STAGE IT organizzato da Confindustria Padova. Anche se non ha vinto il primo premio è stata un'esperienza eccezionale confrontarsi con altre aziende e ragazzi che condividono la mia stessa passione.

OBIETTIVI FUTURI

Nonostante il programma sia pronto per un utilizzo professionale saranno necessari alcuni piccoli accorgimenti per renderlo commerciabile. Il lavoro è attualmente concentrato sul fix di vari piccoli bug riscontrati durante la fase di beta testing ma l'obiettivo più importante è l'inserimento di un sistema anticopia. L'ETA (*Estimated Time of Arrival*) per la commercializzazione si aggira intorno al prossimo autunno. Ovviamente una volta in vendita sarà necessario continuare il supporto per inserire via via nuove features che possano rendere il programma sempre più avanzato e aggiornato.

Ci sono due idee, nate in fase di progettazione, che non sono però mai state realizzate. La prima era una versione Cisco del progetto, in poche parole si trattava di realizzare lo stesso e identico programma ma con Firewall Cisco invece che Linux. Il linguaggio è molto simile ma il tempo disponibile non era sufficiente per implementare entrambe le versioni.

La seconda idea invece è molto più complessa e sarebbe assai interessante realizzarla in futuro. Si tratta di una sorta di Compiler che sia in grado di correggere eventuali errori nel Firewall e di realizzare anche un Reverse Engineering dei Firewall realizzati col programma. In poche parole, a partire dal linguaggio Iptables Linux, ricostruire la struttura XML della regola.

CONFINDUSTRIA PADOVA
Servizi Innovativi e Tecnologici ICT Lab

Camera di Commercio Padova

UNIVERSITÀ DEGLI STUDI DI PADOVA

Galileo
PARCO SCIENTIFICO E TECNOLOGICO

WORKTEAM SRL
E
DANIELE DE MARCO

HANNO PARTECIPATO A

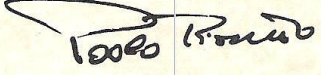
PREMIO stage it 2010

CON IL PROGETTO

FIREWALL CREATOR

Padova 15 Marzo 2011

Il Presidente di ICT Lab
Paolo Fioretto



RINGRAZIAMENTI

Intendo rivolgere un sentito ringraziamento a tutte le persone che mi hanno aiutato nella realizzazione di Firewall Creator e di questa tesi. Ringrazio il mio relatore, il professor Sergio Congiu che mi ha seguito per la tesi. Ringrazio Enrico Conte, direttore dell'azienda WorkTeam che mi ha insegnato a non fermarmi mai davanti a un problema ma ad andare sempre oltre e a trovare soluzioni innovative. Ringrazio infine la mia famiglia per avermi dato l'occasione di studiare e tutti i miei amici e compagni di corso per il loro supporto.

Un ringraziamento speciale va a Gianmarco Bonaiti, professore di informatica presso l'istituto A. Greppi di Monticello Brianza recentemente scomparso. E' grazie a lui e ai suoi insegnamenti se oggi sono riuscito a concludere questo progetto, ed è un peccato che non possa essere con me a condividere questo risultato. L'informatica non è una scienza, è una passione e come tale va coltivata per poter essere espressa al meglio. Il professor Bonaiti è stato il primo a farmelo capire perchè era in grado di trasmettere la sua passione. Grazie ancora, ciao Gianmarco.

BIBLIOGRAFIA

Andrew S. Tanenbaum, *Computer Networks 4th Edition*, Prentice Hall PTR

Linux Man Iptables, http://ipset.netfilter.org/iptables.man.html

Wikipedia the Free Encyclopedia, http://en.wikipedia.org/wiki/Main_Page

CONTATTI

WORKTEAM SRL

SEDE LEGALE E AMMINISTRATIVA
Via Vanoni 11, San Donà di Piave (VE)
TEL +39 0421 220467
FAX +39 0421 482213
MAIL: info@workteam.it

DANIELE DE MARCO

TEL +39 3333474391
MAIL daniele.raz3r@gmail.com