DNS Flood対策

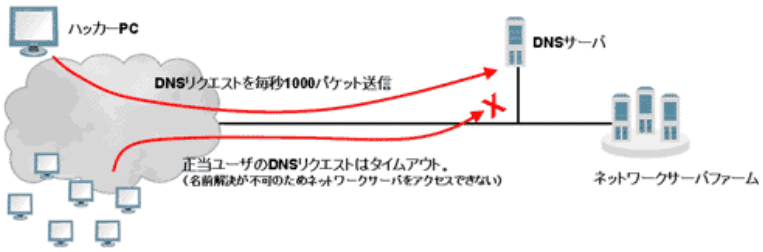
, 2007-02-03

iRulesはTCL（ティクル）というインタープリタ型言語をベースにしています。データ内容による単純なアクセス制限の他に、プログラミング言語だからこそ可能な複雑な処理も行います。この例ではIPアドレスごとのDNSリクエストの回数を、カウンタを用いて計算してリクエスト数を制限します。

課題

Denial of Service （DoS）アタックは、悪質なハッカーが正当なユーザを妨害し、ネットワーク上のサービスを利用できないようにします。DoSアタックの方法、手段は様々ありますが、よく見られるパターンはFloodと呼ばれる、ターゲットとなるサーバへコネクションを大量に開くパターンです。その中の一つの種類としてはDNSサーバへのFloodアタックがあります。DNSはUDPを利用するため、TCPのようなオーバヘッドがなく、OSによる制限がほとんどなく一般のパソコンでも可能なアタック手段です。また、DNSはほとんどのネットワークサービスが利用しているため、1つのアタックでWeb、映像、FTPなど、あらゆるネットワークサービスを妨害します。DNS Floodのイメージを図1で示します。

図1. DNS Floodアタックの例

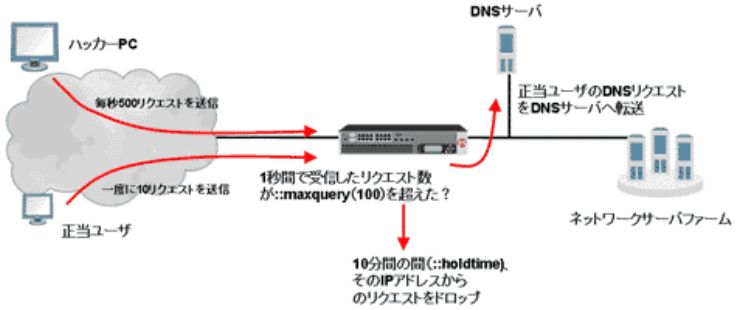


iRuleでの対策

対策の一つとして、大量のDNSリクエストを受信した際、そのIPアドレスからのパケットをドロップすることが考えられます。しかし、アタックではない通常のアクセスの場合でも、ネットワーク・アプリケーションが一定の時間で複数のリクエストを送信することがあります。また、大量にDNSリクエストを受けたことによってあるIPアドレスからのパケットをドロップすることにしたとしても、DHCPによる動的にアサインされるアドレスの存在も考えられます。そのため、永遠にそのIPアドレスをブロックすることではなく、そのIPアドレスを一時的にブロックする方が対策としてふさわしいと考えられます。

iRuleによる対策では、ある一定の時間に受信したリクエスト数があるリミット（::maxqueryと呼びます）を越えた場合に、送信元のIPアドレスからのリクエストをブロックします。そしてある一定の時間（::holdtimeと呼びます）が経過したらそのIPアドレスからのアクセスを再び許可します。動作のイメージを図2に示します。

図2. DNS Floodの対策



実際のiRuleは以下の通りになります。

```
when RULE_INIT {
 set ::maxquery 100
 set ::holdtime 600
```

```
 set ::maxarraysize 400
 array set ::usertable { }
 array set ::blacklist { }
}

when CLIENT_ACCEPTED {
 set srcip [IP::remote_addr]
 set currtime [clock seconds]
 if { [info exists ::blacklist($srcip)] } {
 if { $::holdtime > [expr {$currtime - $::blacklist($srcip)}] } {
 #set ::blacklist($srcip) $currtime
 #log local0. "BL now [array get ::blacklist]"
 drop
 #log local0. "Drop $srcip"
 return
 } else {
 unset ::blacklist($srcip)
 #log local0. "remove $srcip from blacklist"
 }
 }
 if { [info exists ::usertable(time,$srcip)] and $currtime == $::usertable(time,$srcip) } {
 incr ::usertable(freq,$srcip)
 if { $::usertable(freq,$srcip) > $::maxquery } {
 #log local0. "New blacklist member <$srcip> with $::usertable(freq,$srcip) times"
 set ::blacklist($srcip) $currtime
 unset ::usertable(freq,$srcip)
 unset ::usertable(time,$srcip)
 drop
 return
 }
 } else {
 set ::usertable(freq,$srcip) 1
 set ::usertable(time,$srcip) $currtime
 #log local0. "New member <$srcip><$currtime>"
 }
}

when CLIENT_CLOSED {
 if { [array size ::usertable] > $::maxarraysize } {
 set usertablelist [array get ::usertable]
 foreach { x y } $usertablelist {
 if { $x contains "time," and $currtime ne $y } {
 set recip [string trimleft $x "time,"]
 unset ::usertable(time,$recip)
 unset ::usertable(freq,$recip)
 }
 }
 }
 #log local0. "Usertable is now [array get ::usertable]"
}
```

RULE_INITのイベント
RULE_INITイベントでは、それぞれのグローバル変数を定義し初期化します。このiRuleでは次の4つのグローバル変数を利用します。

① : : maxquery - 1秒間で許可する最大のリクエスト数。この例では100と設定します。
② ::holdtime - IPアドレスをブロックする場合、リクエストを再び許可するまでの時間。

この例では600秒（10分）と設定します。
③ ::usertable { } - あるIPアドレスがアクセスした時間と1秒間に受信したリクエスト数を
保存するArray。
④ ::blacklist { } - ブロックするIPアドレスとアクセスした時間を保存するArray。

なお、このiRuleはArray（配列）を活用していますので、特にTCLにおけるArrayの動作を理解しなければなりません。まず、一般的なプログラミング言語で可能な複雑な構造のArrayと違い、TCLではArrayが基本的に2列ある「2次元Array」となります：

```
Array（2列のテーブル）

ip_accesses {
192.168.0.1 50
192.168.0.50 250
192.168.0.20 23
}
```

上記のように、TCLのArrayを定義するとき、{ }で内容を囲みます。そしてArray内のデータをアクセスするには、左側の列の値のどれかを指定し、右側の列の値を利用する、という方法になります。例えば上記のArrayはそれぞれのIPアドレスからのアクセス回数を記録するものです。192.168.0.50のアクセス回数を出力するには下記のようなコマンドを利用します。

```
[ip_accessess(192.168.0.50)] ⇒ 250
```

Array内のデータの更新、変更、置換も同様のフォーマットで実施可能です。

```
set ip_accessess(192.168.0.50) 100 192.168.0.50のデータを100に設定
incr ip_accessess(192.168.0.50) 192.168.0.50のデータに1を足す
```

さらに新たなデータをArrayに追加するには、setコマンドを利用できます。

```
set ip_accessess(192.168.0.220) 500 ⇒ ip_accesses {
 192.168.0.1 50
 192.168.0.50 250
 192.168.0.20 23
 192.168.0.220 500
 }
```

もちろん、Arrayのアクセスエレメント（利用するデータを指定する値、上記の「(192.168.0.50)」）は、静的な文字列だけでなく、変数として指定することも可能です。

```
set ipaddress [IP::remote_addr]
set ip_accessess($ipaddress) 1 IPアドレスをArrayに追加しデータを1に設定
```

今回のiRuleに戻ると、定義したArrayは下記のようなリストとして利用することになります。

フォーマット

```
::usertable { ::usertable {
freq, アクセス回数 freq,192.168.0.1 200
time, アクセスした時刻 time,192.168.0.1 11234499
. freq,192.168.0.50 1000
. time,192.168.0.50 11234499
.
}

::blacklist { ::blacklist {
 アクセスした時刻 192.168.0.1 11234499
. 192.168.0.50 11234499
.
.
}
```

::usertableでは、各IPアドレスに対して、time及びfreqという二つのデータを記録します。::blacklistに対しては、IPアドレスと最後にアクセスした時間だけを記録します。詳細については、後ほど説明します。

CLIENT_ACCPETEDイベント
iRuleの残りの処理はすべてCLIENT_ACCEPTEDイベントで行います。処理として、まずアクセスしているクライアントのIPアドレスとアクセスする時間を変数に保存します。そのコマンドは以下の通りです。

```
 set srcip [IP::remote_addr]
 set currtime [clock seconds]
```

上記のようにsrcipという変数でクライアントのIPアドレスを設定します。また、TCLのclockコマンドを利用し、現在時刻の秒をcurrtimeという変数に保存します。clockのコマンドは下記のフォーマットで利用します。

```
clock <add | clicks | format | microseconds | milliseconds | scan |seconds>
```

今回は現在時刻を秒で換算した値を返すsecondsというパラメータを利用します。なお、返される値はUNIX Time形式となります。UNIX Timeは、日付、時、分などの区別がなく、単に1970年1月1日 00:00:00から経過した秒数だけが記録されます。例えば、2007年1月1日の00:00:00はUNIX Timeで1167609600秒となります。単なる数字になりますので、今回のiRuleに適しています。

では、記録したsrcipとcurrtimeはどのように利用されるでしょうか。このイベントではifコマンドを利用し3つの条件によってそれぞれの処理を実施します。その条件としては以下の通りです。

(1)IPアドレスがブラックリストに未登録の場合
(2)IPアドレスが確認中の場合
(3)IPアドレスがブラックリストに登録済みの場合

以下のセクションにて各条件で行う処理を説明します。

(1)IPアドレスがブラックリストに未登録の場合
各クライアントからDNSリクエストのパケットを受信したとき、::usertableも::blacklistに載っていないクライアントがまず::usertableに登録されます。ここは最後のifコマンドのelseの部分で実施します。ロジックは下記のようになります。

```
if { <::blacklistに載っていない> } { ··· }
if { <::usertableに載っていない> } { ··· }
 else {
 set ::usertable(freq,$srcip) 1
 set ::usertable(time,$srcip) $currtime
 #log local0. "New member <$srcip><$currtime>"
}
```

つまりアクセス時点で::blacklistにも::usertableにも載っていない場合は、そのIPアドレスとアクセスした回数、時刻を::usertableに登録します。動作のイメージを図3に示します。

図3. ::usertableの内容について（ハッカーPCがアクセスする5秒前、正当ユーザが1秒間で10回リクエストを実行）

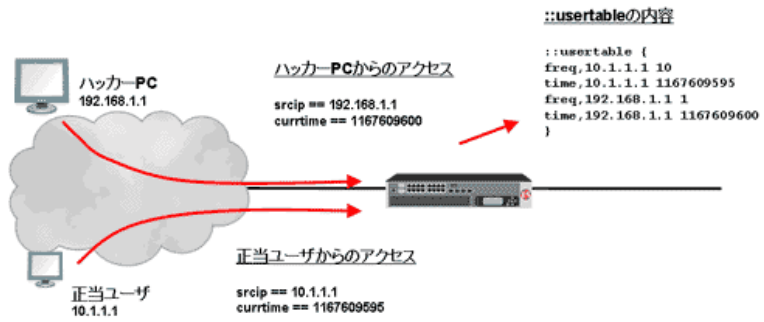


図3では、ハッカーPCが1167609600（UNIX Time）にアクセスした時点のデータです。::usertableの最後にfreq,192.168.1.1とtime,192.168.1.1のエントリが追加されます。ここで重要なポイントは、::usertableはグローバル変数ですので、他のIPアドレスについての情報も記録されています。理由は後ほど説明しますが、ブラックリストに登録されないIPアドレスはすべて一旦登録されます。よって図3の例では、ハッカーPCがアクセスした5秒前に、正当ユーザのアクセスした10回の記録が残っています。

(2)IPアドレスが確認中の場合
1秒間にあるIPから複数のアクセスがあった場合、2回目以降のアクセスでそのIPのアクセス回数を確認する処理に入ります。処理はCLIENT_ACCEPTEDイベントの2つ目のifコマンドで実施されます。

```
if { <::blacklistに載っていない> } { ··· }
if { [ info exists ::usertable(time,$srcip)] and $currtime == $::usertable(time,$srcip) } {
 incr ::usertable(freq,$srcip)
 if { $::usertable(freq,$srcip) > $::maxquery } {
 #log local0. "New blacklist member <$srcip> with $::usertable(freq,$srcip) times"
 set ::blacklist($srcip) $currtime
 unset ::usertable(freq,$srcip)
 unset ::usertable(time,$srcip)
 drop
 return
 }
 else {  }
```

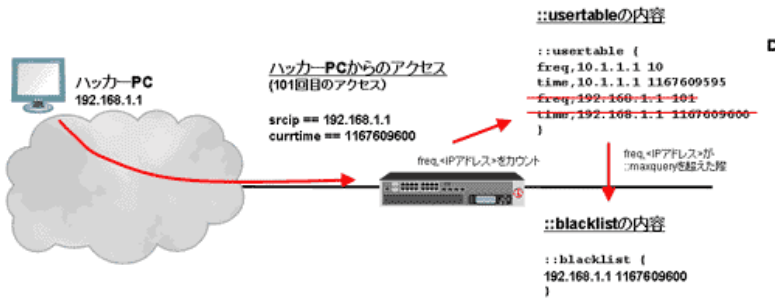
このifコマンドの中では、下記のTCLコマンドを利用します。
info exists <変数名> - 変数名に値が設定された場合に1を返す
incr <変数名> - 変数名の値に1を足して同じ変数に保存
unset <変数名> - 指定した変数を削除
drop - パケット（またはコネクション）を廃棄する
return - 当パケット（コネクション）に対しての処理を終了

まず、info existsコマンドでは、::usertableにtime,のデータが存在するか否かを確認します。存在する場合、次に記録済みの時刻と現在時刻を比較します。記録済みの時刻と現在時刻が秒単位で同じという条件を満たした場合、アクセス回数をカウントする必要があります。

カウントは::usertableのfreq,のエントリで記録しますので、incrのコマンドを利用してアクセスごとにそのエントリを1個ずつ足していきます。そしてfreq,として記録した回数を::maxqueryに比較します。::maxqueryを超えた場合、そのIPアドレスとIPアドレスのアクセスした時刻を::blacklistに追加します。最後にblacklistに載ったIPアドレスは::usertableで記録する必要がなくなりますので、そのIPアドレスと関連するエントリを::usertableから削除します。そしてその時点でdropとreturnコマンドでパケットを廃棄し当パケットに対してのiRule処理を停止します。

動作のイメージを図4に示します。

図4. 1秒間にリクエストを101回送信したIPアドレスは::blacklistに登録



このように、::blacklistに載せたもののみが::usertableから削除されます。そのため、以前にアクセスしたIPアドレスかつ正当なユーザでblacklistに載らなかったユーザは::usertableに残ります。そのため、::usertableが非常に長くなる可能性もあると考えられますが、対策としてはCLIENT__CLOSEDイベントで処理を行います。

(3)IPアドレスがブラックリストに登録済みの場合
CLIENT_ACCEPTEDの最後の処理は実は最初に実施される部分です。

```
 if { [ info exists ::blacklist($srcip) ] } {

 if { $::holdtime > [expr {$currtime - $::blacklist($srcip)} ] } {
 drop
 #log local0. "Drop $srcip"
 return
 } else {
 unset ::blacklist($srcip)
 #log local0. "remove $srcip from blacklist"
 }
 }
if { <::usertableに載っていない> } { ··· }
 else { ··· }
```

このifコマンドでは、上記と同様のdropやreturnコマンドを利用します。その他に下記のコマンドで現在時刻と::blacklistに記録した時刻の時間差を計算します。

```
expr { 数式 } - 指定した数式を実施
```

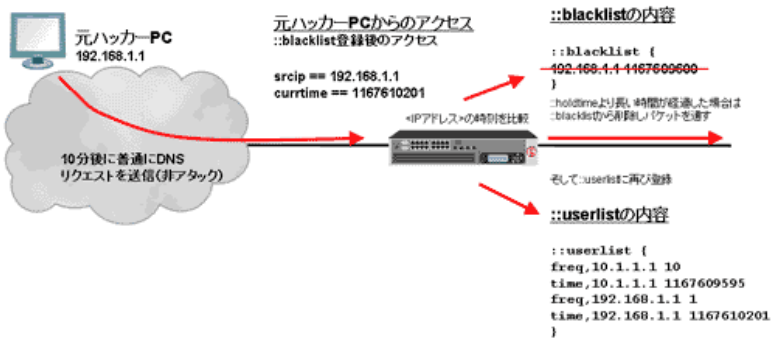
上記の数式では、現在時刻（currtime）から::blacklistに記録された時間を引きます。その結果を::holdtimeに比較し、::holdtimeほどの時間が経過していない場合はコネクションをdropしreturnでiRule処理を停止します。

なお、::holdtime以上の時間が経過した場合、そのIPアドレスをblacklistより削除します。しかし、その後に次のifコマンドのelseの部分

でIPアドレスが再び::usertableに載り、そのIPアドレスに対してのアクセス数確認が再度実施されます。

動作イメージは以下の通りです。

図5. ::holdtimeが経過して::blacklistから削除され、再び::userlistでアクセス回数を確認



上記のように、このifコマンドでは::blacklistに最初に登録した時刻から::holdtimeが経過したところで再度アクセスを許可するようになります。長く続くアタックの場合再度::userlistにおいて::maxqueryとの比較などが必要になります、その結果::holdtimeの間隔で::maxqueryほどのパケット数を通すことになります。それも防ぎたい場合は、ifコマンドに次の一行を追加します。

```
if { [ info exists ::blacklist($srcip) ] } {

if { $::holdtime > [expr {$currtime - $::blacklist($srcip)}] } {
set ::blacklist($srcip) $currtime
drop
#log local0. "Drop $srcip"
return
} else {
unset ::blacklist($srcip)
#log local0. "remove $srcip from blacklist"
}
}
```

上記のsetコマンドでは、::holdtimeが経過していない状態で::blacklistに載っているIPアドレスからのアクセスが入ったら、::blacklistに記録する時刻を更新します。これでアクセスがまったくなくなるまで::blacklistから削除されなくなります。

CLIENT_CLOSEDイベント
上記にも説明しましたが、::usertableに登録する情報はCLIENT_ACCEPTEDでの削除は（::blacklistに登録するとき以外）特に行っていません。その処理はCLIENT_CLOSEDイベントで以下のコードで行います。

```
if { [array size ::usertable] > $::maxarraysize } {
set usertablelist [array get ::usertable]
foreach { x y } $usertablelist {
if { $x contains "time," and $currtime ne $y } {
set recip [string trimleft $x "time,"]
unset ::usertable(time,$recip)
unset ::usertable(freq,$recip)
}
}
}
```

まず、RULE_INITで定義した::usertableの最大のエントリ数に比較し、超えた場合は実際に削除を実施します。これでCLIENT__CLOSEDによるCPU負荷をある程度コントロールできます。つまり::maxarraysizeの値により::usertableの削除する頻度が確定します

ので、これはサイトへのDNSアクセスするIPアドレスの数や受信レートにより決めると良いでしょう。また、各IPアドレスに対して::usertableに2つのエントリーが作成されますので、記録するIPアドレスの数は::maxarraysizeの半分となります。

::usertableが::maxarraysizeを超えた場合、次にforeachコマンドで各エントリを見ながら、現在時刻と異なるもの（つまり古いデータ）のみを::usertableから削除します。foreachコマンドの使い方は以下の通りになります。

```
foreach <エントリ変数> <リスト変数> {
コマンド
...
}
```

foreachでは、リスト内の各エントリを一旦指定したエントリ変数に保存し、{}の中のコマンドを実行します。そしてエントリ変数は単なる変数だけでなく、こちらもリストでも利用可能です。そのため、以下のような使い方の場合、リスト内のエントリを2つずつ、それぞれにxとyという変数に保存します。

```
foreach { x y } $usertablelist { ... }
```

また、foreachではリスト変数は利用しますが、Array変数は利用不可となります。そのため、2次元であるArrayを一旦1次元のリストに変換する必要があります。この処理はforeachの前に、以下のコマンドで実施します。

```
set usertablelist [array get ::usertable]
```

array get のコマンドでは、::usertableのすべての内容が空白で区切られた1つの文字列として返されます。それをusertablelistに保存すると、usertablelistがリスト変数になります。例は以下の通りです。

```
2次元のArray変数 1次元のリスト変数
::usertable { ::usertablelist {
freq,192.168.0.1 200 freq,192.168.0.1
time,192.168.0.1 11234499 200
freq,192.168.0.50 1000 time,192.168.0.1
time,192.168.0.50 11234499 11234499
} freq,192.168.0.50
 1000
 time,192.168.0.50
 11234499
 }


foreach内のコマンドとしては、次の処理を行います。

 if { $x contains "time," and $currtime ne $y } {
 set recip [string trimleft $x "time,"]
 unset ::usertable(time,$recip)
 unset ::usertable(freq,$recip)

foreachのxとyのエントリ変数、それぞれに::usertablelistのエントリが入ります。
```

リスト変数 foreachのx y 変数

```
::usertablelist { (1回目)
freq,192.168.0.1 x==freq,192.168.0.1
200 y==200
time,192.168.0.1 (2回目)
11234499 x==time,192.168.0.1
freq,192.168.0.50 y==11234499
1000 ・
time,192.168.0.50 ・
11234499 ・
}
```

そしてforeach内では、まずxが"time,"を含む文字列となっているか否かを確認します。含む文字列となっている場合はxに保存されている文字列(つまり"time,")から"time,"の部分を削除し、IPアドレスのみを取得します。string trimleftのコマンドを利用します。

```
string trimleft <文字列または文字列を保存する変数> <削除文字列>
```

string trimleftコマンドが指定した文字列から、左側から「削除文字列」を消します。例えばtime,192.168.0.1の文字列の場合、このiRuleで使うstring trimleftコマンドの実行後は192.168.0.1のみです。そのIPアドレスを使って、実際の::usertableのArray変数からエントリを削除します。

その結果、usertableが::maxarraysizeより大きくなった場合、CLIENT_CLOSEDイベントで(現在時刻より)古いエントリを全部削除します。

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com