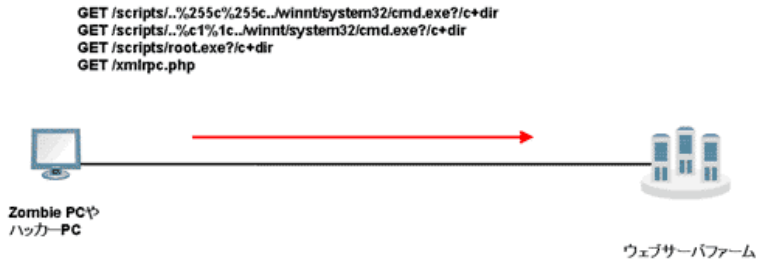
簡単なアタック防止

, 2007-01-03

iRulesの活用は、簡単なコードを適用するだけでも実にパワフルな機能となることを証明しています。
この例はコードが7行しかない短いものですが、BIG-IPのバックエンドに設置されるサーバの無駄な処理を減らします。

課題
スクリプトによるWebサーバへのアタック対策は最近の流行ではなく、何年も前から通常のセキュリティ対策の1つとなっています。図1はスクリプトによるアタックのイメージです。

図1. スクリプトによるアタック



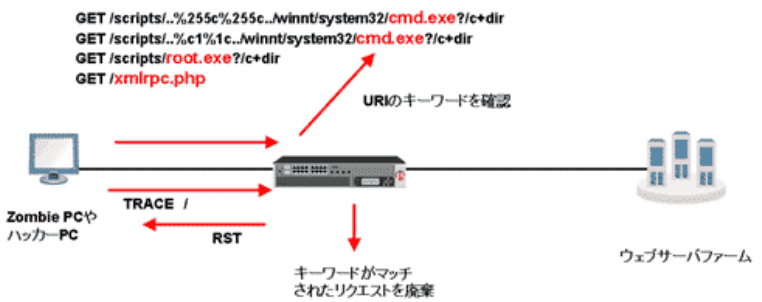
図中の「ウェブ」を「Web」へ変更

この例では、Zombie PC（ウィルスなどを感染したPC）がウェブサーバへ特定のURIをリクエストします。対策がとれていないWebサーバがそのリクエストを受けた場合、そのサーバへの管理権限が不当に与えられてしまうことがあります。もちろん、スクリプトによるアタックの大半はすでに公表されており、最新のソフトウエアにおいては対策が施されていますが、いずれにしてもこのような無駄なリクエストが頻繁にWebサーバに届けられ、サーバのリソースを無駄に消費しているサイトは少なくないでしょう。

iRuleでの対策
アタックであるリクエストそのものがWebサーバに到達しないようにすれば、Webサーバのリソースをエラーメッセージ出力ではなく、本来処理すべきリクエストにより効率的に割り当てることができます。また、仮に対策が施されていないサーバが存在する環境であれば、サーバ自体を守ることもできます。

対策方法としては、スクリプトによるアタックの特定URIを認識し、BIG-IPでリクエストを廃棄します。動作イメージは図2の通りです。

図2. iRuleによるリクエスト・スクリーニング



図中の「ウェブ」を「Web」へ変更

このように、iRuleを用いて、アタックがリクエストするURIに含まれるキーワードでスクリーニングを行います。例えばIISに対してのアタックはcmd.exeやroot.exeという実行ファイルをリクエストします。Webサイトにその文字列を含む実際のURIは存在しないため、cmd.exeとroot.exeを含むリクエストをBIG-IPで廃棄します。またxmlrpc.phpの古いバージョンに対してのセキュリティ問題が発見されたため、xmlrpc.phpをリクエストするアタックも存在します。そのファイルを利用しないWebサイトであれば、キーワードとして登録するといいでしょう。
TRACEなどのHTTPメソッドによりセキュリティ脆弱性もありますので、GET、POST以外のメソッドにはRSTを返します。
では、実際のiRuleを見てみましょう。

```
when HTTP_REQUEST {
 if { [matchclass [URI::decode [string tolower [HTTP::uri]]] contains $::bad_uris] } {
 discard
 } elseif { not [matchclass [string toupper [HTTP::method]] equals $::valid_methods] } {
 reject
 }
}
```

まず、このiRuleはクライアントからのリクエストを受信したところで発生するHTTP_REQUESTイベントのみを利用します。そのタイミングで2つのmatchclassコマンドで実際のスクリーニングを行います。

matchclass <値> <オパレータ> <クラス名>

まずはURIに対してのmatchclassを定義します。

```
if { [matchclass [URI::decode [string tolower [HTTP::uri]]] contains $::bad_uris] }
```

基本的にURIに「bad_uris」というクラスで指定するキーワードが含まれているか否かを確認しますが、アタックとして受信するURIリクエストに大文字、小文字の英数字や16進数で指定したASCII文字が混雑される場合もあります。そのため、すべてのURIを小文字で16進数としてエンコードされた文字をASCIIに置換えます。

次に、matchclassに渡す値(URI)に対して2つのコマンドを実行します。まず文字列を全体的に小文字にするために、「string tolower」コマンドを利用します。

string tolower <文字列> → string tolower [HTTP::uri]

例えばURIが「/scripts/ROOT.EXE?/c+dir」の場合、上記のコマンドで「/script/root.exe?/c+dir」になります。

また、16進数でエンコードされたURIもあります（参照：RFC 2616とRFC 2396）。例えば空白（スペース）は%20とURIに書くこともできます。しかし、「bad_uris」にキーワードがASCIIの空白を含む文字列として登録されると、matchclassのマッチングができません。そのため、URI::decodeコマンドを利用してすべてのURIをASCIIテキスト・フォーマットに戻します。

URI::decode <値> → URI::decode [HTTP::uri]

例えばURIが「/scripts/bad%20script.exe」の場合、上記のコマンドでURIが
「/scripts/bad script.exe」になります。

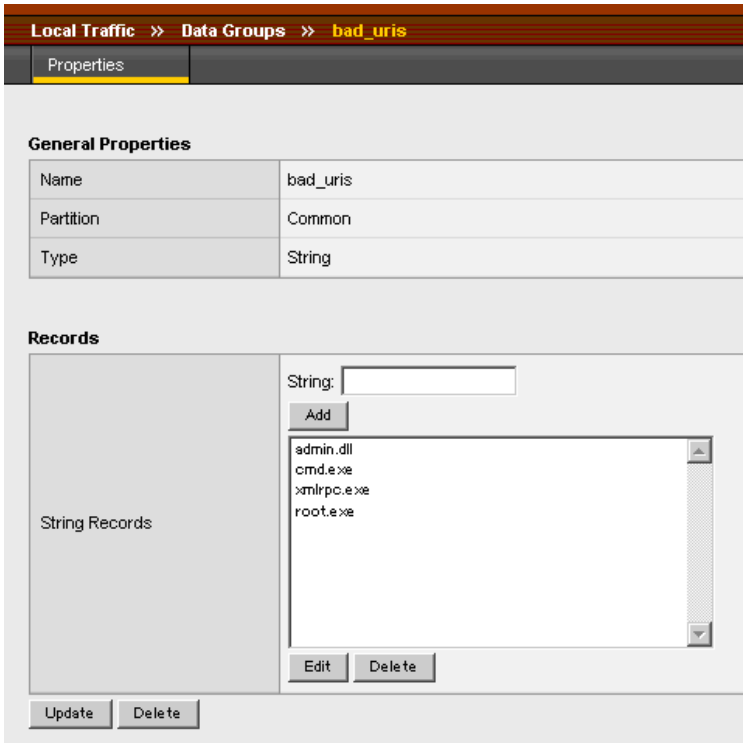
上記のコマンドでは、ユーザから受信したURIがASCIIで書かれた全小文字の文字列に変換されます。そしてその文字列をbad_urisと比較しますが、matchclassコマンドのため、これはCLIで追加するときはclassコマンドで定義します。

```
(CLIのbigip.confファイルに追加するテキスト)
class bad_uris {
```

```
 "cmd.exe"
 "root.exe"
 "admin.dll"
 "xmlrpc.php"
}
```

また、GUIではLocal Traffic >> iRules >> Data Groupsの画面でも入力できます。

図3. GUIでのclass登録（Data Group）



classでキーワードを登録することの利便性として、新たなアタックが発見された場合、classにそのキーワードを追加するだけですぐに対策が可能なことが挙げられます。

classのキーワードにマッチングしたものは、discardコマンドによりコネクションがそのまま廃棄されます。discardの場合、TCP RSTなどのパケットはユーザへ応答されません。

次に、bad_urisのキーワードを含まないURIに関しては、リクエストが利用したHTTPメソッドの確認が行われます。

```
elseif { not [matchclass [string toupper [HTTP::method]] equals $::valid_methods] }
```

同様にmatchclassのコマンドを利用します。今回はHTTP::methodコマンドでメソッドを取り出してから、string toupperコマンドでその文字列を全大文字に変換します。その文字列がvalid_methodsというクラスに登録されている文字列に比較されます。Valid_methodsのクラスの定義は以下の通りになります。

```
class valid_methods {
 "GET"
 "POST"
}
```

valid_methodsのキーワードにマッチングしない場合はrejectコマンドでコネクションがリセットされます。rejectの場合、TCP RSTがユーザへ応答されます。

F5ネットワークスジャパンでは、サンプルコードについて検証を実施していますが、お客様の使用環境における動作を保証するものではありません。実際の使用にあたっては、必ず事前にテストを実施することを推奨します。

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