CHAPTER 6

ルールとシグニチャの開発

この章では、カスタムの Web アプリケーション セキュリティ ルールおよびシグニチャを開発する方法を説明します。内容は次のとおりです。

概要

Cisco ACE Web Application Firewall には、Web アプリケーションを保護するためにそのまま使用できるルールとシグニチャの拡張ライブラリが含まれています。組み込みルールは、クロスサイト スクリプティング攻撃、Structured Query Language（SQL）および Lightweight Directory Access Protocol（LDAP）インジェクション攻撃、ならびにコマンド インジェクション攻撃など、さまざまなタイプのサーバ攻撃を防止するために存在しています。

それにもかかわらず、アプリケーションには多くの場合、セキュリティやコンテンツ プライバシに関する固有の懸念事項があります。組み込みリソースでは、お使いのアプリケーションや環境特有のすべての要件には対処できません。そのような場合に、Cisco ACE Web Application Firewall を利用して、独自のカスタム ルールおよびシグニチャを作成することができます。この拡張性により、Cisco ACE Web Application Firewall の機能を特有の要件にしっかり適応させることができます。

ここで説明するとおり、Reactor ルール言語を使用して、カスタムのルールとシグニチャを作成します。ルールとシグニチャを Manager にアップロードした後は、組み込みルールと同じ方法で、それらのルールとシグニチャを Web アプリケーションのプロファイルで適用できます。

ほとんどの場合は、新規または緊急の攻撃パターンに基づいて新しいシグニチャを適用するという要件が、ルール開発の動機となります。したがって、ルール開発プロジェクトには通常、新しいルールとともに新しいシグニチャの開発を伴います。

シグニチャとルールの開発の一般的な手順は次のとおりです。

1. ルールによって処理または保護するトラフィックを照合するために必要なシグニチャを作成します。

   シグニチャとルールの両方が作成され、テキスト ファイルとしてシステムにロードされます。ただし、Manager はルールとシグニチャを別個のリソースとして扱うため、それらを別個のテキスト ファイルに作成する必要があります。

2. カスタム ルールとルール グループを別個のテキスト ファイルに作成します。ルールは、組み込みシグニチャまたはカスタム シグニチャに依存することができます。

3. シグニチャ ファイルを（カスタム シグニチャに依存するルールをロードする前に）Manager にロードします。Manager はファイル内のシグニチャを検証し、Web コンソール ページに問題を示します。
4. カスタム ルール ファイルをアップロードします。ルールをインポートするときに、Manager は、ルールで参照されているシグニチャがシステムに存在することを確認し、ルールに対してその他の妥当性チェックを行います。

アップロードしたカスタム ルールは、プロファイルで適用できるようになります。また、Manager インターフェイスから後で直接ルールを変更することもできます。

これらの手順は、ルールとともにシグニチャを含む開発を説明したものですが、新しいシグニチャに依存しない新しいルールを開発する場合もあります。たとえば、未使用の組み込みルールのいずれかを適用したり、組み込みルールによって適用される方法とは別の方法で組み込みシグニチャを適用したりする場合があります。最も一般的なケースとして、シグニチャを使用しないルールを作成する必要がある場合があります。たとえば、データ オーバフローをチェックするルールを実装するには、次のようなルール文を使用します。

```
REQUEST_PARAMS.count() gt 100
```

もう 1 つの例として、次のような式を使用して、正規表現に対して特定のパラメータをチェックすることができます。

```
REQUEST_PARAM['username'] nre '[A-Za-z0-9]+'
```

次のセクションでは、ルール開発手順と要件の詳細情報を示します。

メッセージの正規化

ルール開発を簡素化するため、ACE Web Application Firewall はルールを評価する前にメッセージを自動的に正規化します。このため、作成するルールおよびシグニチャでさまざまなタイプのメッセージのバリエーションに対処する手間を省くことができます。たとえば、ISO-8859-1 ベース攻撃に対するシグニチャまたはルールのバージョンとは別に、同じ攻撃の UTF-8 形式に対するシグニチャまたはルールのバージョンを作成する必要はありません。また、メッセージのコンテンツを浄化することもできます。

メッセージの正規化は、ACE Web Application Firewall 経由で実際に宛先に渡されるメッセージには影響しません（NULL バイト スクリーニングやヘッダーの展開を除く）。この機能は、メッセージのコピー、さらに正確に言えば、ルール評価に使用される REQUEST_PARAMS や REQUEST_URL などの変数の入力に使用される、メッセージの値に対してだけ実行されます（「ルール変数」（P.6-19）を参照）。

正規化自体では出力メッセージは変更されませんが（ヘッダーの空白スペースの展開を除く）、カスタム エラー メッセージや HTTP ヘッダー処理などのポリシーのその他のコンテキストで変数を使用する場合は出力に影響する可能性があります。たとえば、要求の一部を正規化し、それを使用して発信ヘッダー値を入力することができます。

メッセージに適用されるメッセージの正規化には、2 つのタイプがあります。1 つは、*受動正規化*と呼ばれます。この正規化は、すべてのメッセージに対して自動的に実行されます。もう 1 つは `normalize()` 関数を使用してルールから直接呼び出される正規化です。

受動正規化

受動正規化は、メッセージに次のように影響します。

- NULL バイトを含むメッセージはすべて拒否されます（NULL バイトの存在は致命的なエラーとして処理され、ACE Web Application Firewall は、400 番のクライアント エラーを返し、変形された要求を示す警告レベルのイベントを生成します）。
- Firewall は、要求 URL（GET 引数を含む）と POST の本体で次のことを行います。
  - URL エンコードされた（つまり、パーセントエンコードされた）文字を UTF-8 エンコードされた文字にデコードします。この動作は、ルールとシグニチャの構成に対して次の意味を持ちます。

    ASCII 文字の場合、%27 などのパーセントエンコードされた文字（URL エンコードされた単引用符）を含む引数を照合するために、カスタム シグニチャでは「'」を検索するだけ済みます。

    ルールやシグニチャに Unicode 文字（特定のキリル文字など）を使用することもできます。アップロードするシグニチャまたはルール ファイルは UTF-8 形式である必要がありますが、Unicode 文字も正常に照合されます。

    シグニチャで文字以外にデコードされた値を検索するには、シグニチャまたは正規表現に 16 進数表現を入力します。たとえば、9 を表すバイトを検索するには、\x09 を入力します。
  - %u にエンコードされた文字（Microsoft で使用される非標準の Unicode エンコード）を UTF-8 にデコードします。
- 要求パスに限り（クエリー文字列、つまり、「?」の後に続く文字列は含まれません）、Firewall は自己参照（./）や後方参照（../）など、難読化している可能性のあるパス コマンドを正規化します。これは、仮想 Web アプリケーションのトラフィックを適切に照合するのに役立ちます（つまり、「/app/../path」の要求は実際、「/app」プレフィクス ハンドラを誤って照合するのではなく、「/path」ハンドラを照合します）。
- HTTP ヘッダーでは、Firewall は次のことを行います。
  - 空白スペースの展開を適用します（http://www.w3.org/Protocols/rfc822/3_Lexical.html）。ここで、複数行のヘッダーが 1 行に変換されます。

（注）　空白スペースの展開は、出力メッセージに影響する唯一の正規化機能です。

  - その他の HTTP ヘッダーは、Host ヘッダー以外は変更されません。Host ヘッダーの場合、Firewall は URL エンコードまたは %u エンコードされた値を UTF-8 に変更します。ホスト名の評価では大文字と小文字は区別されないので、ヘッダー値全体が小文字に変換されます。Host ヘッダーの正規化によって、Referer ヘッダーのチェックが容易になります。

normalize() 関数

「受動正規化」（P.6-3）で説明した正規化プロセスの目的は、評価するメッセージ コンテンツ内の多数の下位レベルのエンコード問題に対処するためのシグニチャやルールを作成する必要をなくすことにあります。ただし、パスの正規化を除き（要求をハンドラに割り当てるのに役立ちます）、このレベルの正規化は、過剰な空白スペース、エンコードされたエンティティ（<script の代わりに <script を使用するなど）、およびコメントによる攻撃文字列の分割など、より高いレベルの難読化を防止することを目的としていません。

もう 1 つの正規化レベルである*直接正規化*は、このタイプの難読化に対処することを目的としています。直接正規化を使用するには、次に示すように、メッセージの一部に対して `normalize()` 関数を呼び出します。

VARIABLE['*fieldName*'].normalize() *op testvalue*

たとえば、次のとおりです。

```
REQUEST_HEADER['My-Header'].normalize(charset|uunicode) re 'attack!'
```

この場合、HTTP ヘッダーの値 `My-Header` は正規化されてから、正規表現「`attack!`」を使用してテストされます。

GET および POST パラメータと POST の本体の場合、ここで説明する多数の正規化プロセスが自動的に適用されるので、この関数を使用して正規化プロセスを再び適用する必要はありません。ただし、HTTP ヘッダーやマルチパートのコンテンツ配置などは、受動正規化によって広範囲に正規化されることがなく、正規化関数を介して手動で正規化します。

`normalize()` 関数は、GET または POST 引数に自動的に適用されるすべての処理（URL デコードと %u デコード）に適用できます。さらに、次のことを行えます。

1. 文字を小文字に変換します。
2. すべての空白スペースの実行（あらゆるタイプの空白スペース文字）を 1 つの「スペース」文字（ASCII 32 と等価）に縮小します。
3. C 言語と CSS 形式のコメント（/* ... */ 形式）のほか、HTML/XML コメント（<!-- ... --> 形式）を除去します。これらのコメントは、正規表現タイプのスキャナでチェックされる文字のシーケンスに無関係の文字を割り込ませる攻撃を隠すために使用される場合があります。割り込ませた無関係の文字は、最終的に宛先のパーサーでは無視されます。
4. HTML、16 進数（&#xNN）、および 10 進数（&#N...N;）のエンティティをデコードします。
5. \t、\001、\xAA、\hHH、\0OOO など、JavaScript 形式および C 言語形式のエンコードをデコードします。

`normalize()` の第一の目的は難読化の試みを阻止することですが、同時にシグニチャやルールのオーサリングも容易にします。たとえば、属性名と値の間にある任意の量の空白スペースを調べる正規表現を作成する代わりに、スペース文字が 1 つ存在する、またはゼロであることを仮定することができます。

正規表現を実行するときに、値に `normalize()` を実行しても（たとえば、`REQUEST_HEADER['My-Header'].normalize(charset|uunicode) re 'attack!'`）、パラメータ `Name` の値は変更されません。正規化された値が作成され、この特定の正規表現の評価にだけ使用されます。

通常は、コンテンツベースのシグニチャ（コマンド インジェクションなど）を実行する前に値に `normalize()` を適用します。ただし、normalize() の実行によって攻撃が認識できなくなる場合があり、そのような場合には、攻撃をそのままの形で受け止めるしかありません。

また、カスタムのエラー応答メッセージ内やヘッダー内など、仮想 Web アプリケーション設定の任意の場所に `normalize()` を使用して、出力コンテンツを正規化することもできます。たとえば、HTTP ヘッダー処理の設定内で未加工のヘッダーを浄化バージョンに置き換えることができます。つまり、ヘッダー `Custom-header` の値を `REQUEST_HEADER['Custom-header'].normalize()` に置き換えます。このように、メッセージの正規化バージョンは発信要求には伝播されませんが、HTTP ヘッダー処理機能を使用して、発信メッセージに伝播されるヘッダーを正規化することができます。

（注） HTTP ヘッダー処理を使用してヘッダー リストを完全に再構築することはできますが、現在、GET および POST パラメータをリライトすることはできません。

`normalize()` 関数を使用するには、パラメータとしてメッセージに適用する正規化の機能を識別する必要があります。サポートされるパラメータは次のとおりです。パイプ演算子（「|」）を使用して複数のパラメータを渡すことができます。

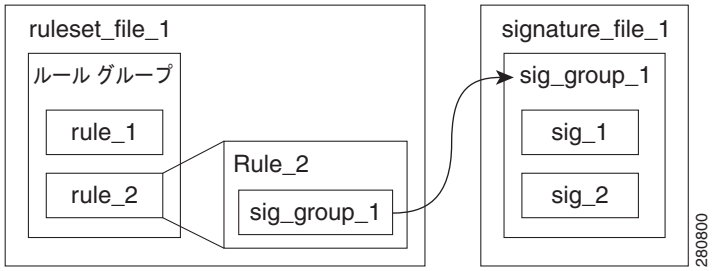
- url：URL デコードを実行します。つまり、% にエンコードされた文字を同等のバイナリに変換します。これにより、読み取り可能な通常の文字が生成される場合と、文字以外のバイトが生成される場合があります。通常、このオプションは charset オプションと一緒に使用します。
- charset：前の手順で生成された文字以外のバイトを取り、UTF-8 エンコードに変換します（元のエンコードが ISO-8859-1 であったという前提条件が使用されます）。このオプションは通常、url 引数と一緒に使用されますが、単独で使用することもできます（たとえば、HTTP ヘッダーなどの URL エンコードではなく、拡張 ASCII インジェクションの検査を必要とする値がある場合）。
- uunicode：%u でエンコードされた文字を同等の UTF-8 に変換します。
- htmlent：10 進数や 16 進数を含むすべてのタイプのエンティティ（&...;）を同等の非エンティティに変換します。
- escapes：C 形式のバックスラッシュエスケープ文字を同等の非エスケープ文字に変換します。
- nullsp：埋め込まれた NULL 文字をスペースに変換します。NULL バイトを含む要求は受動正規化により拒否されるので、通常はこの正規化機能を要求に対して実行する必要はありません。ただし、この機能は、その他のタイプのコンテンツで NULL 文字の有無をチェックするためのメカニズムとして提供されています。
- nullremove：埋め込まれた NULL 文字を除去します。NULL バイトを含む要求は受動正規化により拒否されるので、通常はこの正規化機能を要求に対して実行する必要はありません。ただし、この機能は、その他のタイプのコンテンツで NULL 文字の有無をチェックするためのメカニズムとして提供されています。
- ccomment：C-/Javascript 形式のコメントを除去します。
- xmlcomment：XML/SGML 形式のコメントを除去します。
- backslash：バックスラッシュ（\）を除去します。
- selfref：自己参照パス（./）を除去します。
- backref：後方参照パス（../）を除去します。
- trailingslash：パスの末尾のスラッシュを除去します。
- white：空白スペースの実行を正規化し、1 つのスペース文字（ASCII 32）に縮小します。
- lower：文字列内のすべての英字を小文字に変換します。

シグニチャの開発

シグニチャには、対象のメッセージとルールを照合するためのコンテンツ パターンが少なくとも 1 つ含まれます。シグニチャによって照合されたメッセージには、ルールのアクションが適用されます。システムには、多数の組み込みシグニチャがあります。また、ここで説明するとおり、独自に作成して適用することもできます。

まず、テキスト ファイルにシグニチャを作成し、そのファイルを Manager にアップロードします。テキスト ファイルには、1 つまたは複数のシグニチャを保存することができ、各シグニチャはシグニチャ グループに属する必要があります。図 6-1 に示すとおり、ルールはシグニチャ グループへの参照を介してシグニチャを適用します。

図 6-1　　ルールセットおよびシグニチャ セット ファイル



システムには、シグニチャをグループ別に整理することに関する要件はありませんが、シグニチャを合理的に整理することはパフォーマンスの最適化につながります。ACE Web Application Firewall は各シグニチャ グループを 1 つの実行可能単位にコンパイルするので、1 つのグループ内の複数のシグニチャのほうが、複数のグループ内の同数のシグニチャよりもすばやくトラフィックに適用できます。一般的には、特定の攻撃クラスをターゲットとするシグニチャを 1 つのシグニチャ グループにまとめます。グループ内の特定のシグニチャを所定の Web アプリケーションに適用できない場合、ポリシー開発者は修飾子を使用してアプリケーションに適用できないシグニチャを免除することができます。

シグニチャ文は、シグニチャの ID、そのグループ、およびその他の複数のプロパティで構成されます。プロパティの 1 つに *regex* 属性があります。この属性には、メッセージ コンテンツに適用される正規表現が含まれます。Cisco ACE Web Application Firewall の Web アプリケーション セキュリティ機能に採用されている正規表現の実装は、Perl-Compatible Regular Expressions（PCRE）です。正規表現には、文字列の終わりに一致するドル（$）や、文字列の始まりに一致するカレット（^）などの PCRE 正規表現の特殊文字を組み込むことができます。バックスラッシュによって特殊文字はエスケープされ、単純な文字として評価されます。すべての PCRE 機能がサポートされているわけではありません。たとえば、キャプチャ グループの参照はサポートされていません。

（注） PCRE 構文の詳細については、http://www.pcre.org/pcre.txt を参照してください。

既存のシグニチャの表示

独自のシグニチャまたはルールを作成するときは、通常、組み込みリソースのいずれかで提供されている例から始めると便利です。ポリシーにロードされたシグニチャのソース コードには、形式化されたビューと未加工のビューの 2 つがあります。

- 形式化されたビューについては、[Rules & Signatures] ページで [View Signatures] ボタンをクリックします。システムにロードされたすべてのシグニチャ グループがウィンドウに表示されます。所定のグループのシグニチャを表示するには、グループのエキスパンダ コントロールをクリックします。展開されると、シグニチャを構成する正規表現パターンのソースが表示されます。
- 未加工のビューについては、[Rules & Signatures] メニュー リンクをクリックしてから、[Manage Signatures] ボタンをクリックします。シグニチャ セット リソースの名前をクリックすると、別のブラウザ ウィンドウにリソース ファイルのソース コードが表示されます。

未加工のビューは、最初にシグニチャ ソース ファイルを作成するときに役立ちます。形式化されたビューは、さまざまな攻撃シグニチャに対してシステムで使用する正規表現パターンを迅速に検出するのに役立ちます。

基本的な例

シグニチャのソース コード ビューに表示されるとおり、次のような基本的なシグニチャ宣言が表示されます。

```
SQLInjection_m.drop:DROP
 regex: ;\s*\bDROP\b
 nocase: true
 name: DROP
```

このサンプルにある 1 つのシグニチャ `drop` は、シグニチャ グループ `SQLInjection_m` に属しています。シグニチャ名とグループの宣言のほか、シグニチャには次のプロパティがあります。

- 名前の宣言の後には*簡易パターン*表現が続きます。この例では、「DROP」です。正規表現で使用する際、この表現は最適化された仮のコンテンツ テストとして機能します。`regex` 属性に使用できる正規表現言語機能のサブセットを使用して、簡易パターン表現を作成します。詳細については、「簡易パターンについて」（P.6-8）を参照してください。
- `regex` は、対象のメッセージ コンテンツを照合するための正規表現です。この正規表現パターンには、ワード境界表現に囲まれた DROP 文字列が含まれます。
- `nocase` は、適用される正規表現に大文字と小文字の区別があるかどうかを示します。デフォルトでは、大文字と小文字を区別しない方法で比較が実行されます。この属性を取り入れ、`false` に設定すると、大文字と小文字を区別した比較を指定できます。
- `name` は、Web コンソール インターフェイスに表示されるシグニチャの記述名です。

ルールは、次のようにシグニチャ グループの名前を参照することによって、シグニチャをトラフィックに適用します。

```
RULEGROUPNAME.RuleName:REQUEST_ALL sig SQLInjection_m
```

`sig` キーワードに続く識別 ID `SQLInjection_m` は、シグニチャ グループを識別します。

次のセクションでは、シグニチャの各部について詳しく説明します。

簡易パターンについて

シグニチャは、正規表現を 2 段階に分けてメッセージ コンテンツに適用できます。その最初の段階が簡易パターン表現です。メッセージが簡易パターンに一致し、シグニチャに *regex* 属性も存在する場合は、regex 属性の完全な正規表現がメッセージ コンテンツに適用されます。

簡易パターンは、ACE Web Application Firewall がシグニチャの適用外であるメッセージを迅速に判断できるようにする最適化機能です。この機能では、正規表現に似た制限付きの構文を使用して、非常に迅速なメッセージの初期チェックが行われます。簡易パターンがメッセージに一致すると、ACE Web Application Firewall は完全な正規表現（存在する場合）を適用します。ルールを始動するには、両方の表現が明確に一致する必要があります。シグニチャには、1 つの簡易パターンまたは regex プロパティだけが含まれる場合があります。その場合、その表現だけがメッセージ コンテンツに一致すれば、ルールのアクションが始動します。

（注）　リライト ルールでは、簡易パターンだけを使用してコンテンツを照合できます。つまり、リライト ルールは、簡易パターンを使用するシグニチャだけを参照できます。

実際、簡易パターンは一般に、対象になり得るコンテンツを示す最小の文字シーケンスの有無をチェックし、正規表現は、DROP コマンドに対する組み込みの SQL インジェクション シグニチャが示すとおり、より徹底したコンテンツのテストを行います。

簡易パターン：DROP

Regex パターン：;\s*\bDROP\b

簡易パターン表現の構文には、PCRE 構文のサブセットが含まれます。リテラル文字の照合のほか、次の照合もサポートされます。

- すべての文字を照合する、「.」（ドット）などの単純な文字クラス ワイルドカード
- 次のように、事前に定義された文字クラス
  - \d：ディジット
  - \D：ディジット以外
  - \s：空白スペース
  - \S：空白スペース以外
  - \w：ワード（アルファベット、数字、または下線）
  - \W：ワード以外
- [a-z] などのユーザ定義の文字クラス
- 「a」には「\x61」など、16 進数にエンコードされた文字リテラル

組み込みシグニチャに関する注意事項

次に、基本設定に含まれた、ルール開発に使用できる組み込みシグニチャに関する注意事項を示します。

未使用のシグニチャ

ほとんどの組み込みシグニチャが組み込みルールのいずれかによって適用されますが、いくつか適用されないものもあります。これらのシグニチャは基本設定に含まれており、カスタム ルールに使用することができます。

- Quotes.single
- Quotes.double
- Tab.Tab
- SQLComment.dashdash

これらのシグニチャは、正規のメッセージによくある文字を照合します。たとえば、単一引用符文字は、フォーム データとして入力される苗字（O'Neil など）に含まれる場合があります。したがって、これらのシグニチャは、アプリケーション トラフィックに適していると確信できる場合、またはシグニチャを監視モードだけで適用する場合に限り使用してください。

CreditCard シグニチャ グループ

組み込みシグニチャ グループには、さまざまな長さのクレジット カード番号をすべて検索する複数のクレジット カード シグニチャ グループが含まれています。シグニチャは、適切に区切られた（スペースまたはハイフンを使用）、指定の長さの数字の集合を照合することに注意してください。疑いのあるクレジット カード番号の数字シーケンスが実際に有効なクレジット カード番号であることを確認するためのチェックサム式（Luhn アルゴリズムなど）には、シグニチャは適用されません。

カスタム シグニチャのアップロード

シグニチャの開発を終えたら、それらを Manager にアップロードしてポリシーで使用できようにします。カスタム シグニチャをアップロードするには、次の手順を実行します。

ステップ 1　[Rules & Signatures] ページで、[Manage Signatures] ボタンをクリックします。

ステップ 2　[New Signature Resource] ボタンをクリックします。

ステップ 3　シグニチャ リソースの記述名を入力します。名前は、ポリシー内のシグニチャ リソースに固有である必要があります。この記述名がシグニチャ リソース リストに表示されるので、Web コンソールの他のユーザにとってわかりやすい名前にします。

ステップ 4　[Browse] ボタンをクリックしてファイル システム上のファイルに移動することで、アップロードするシグニチャを格納するファイルを特定します。または、リソースが Web サーバから提供されている場合は、使用可能な URL を入力します。Web サーバの場所からアップロードしたリソースには、リソース リフレッシュ機能が適用され、ポリシーの導入時にリソースが自動的に再ロードされます。

ステップ 5　[Upload] をクリックします。Manager はファイル内のシグニチャを検証し、検出したエラーを表示します。完了すると、新しいリソースがリソース リストに表示されます。

シグニチャが正常にポリシーにロードされた後は、それらのシグニチャを使用して、ACE Web Application Firewall によって処理されるメッセージから対象のコンテンツを識別できます。

シグニチャ リスト内のリソースの横にある [edit] リンクをクリックし、アップデートしたファイルを [edit resource] ページにアップロードすることで、いつでもリソースをアップデートできます。

シグニチャの開発に関する一般的なガイドライン

アップロードした各シグニチャ ファイルは、ポリシー内で 1 つの名前付きリソースとなります。そのため、Manager のバージョンコントロールおよびポリシー アーカイブ機能の影響を受けます。

次に、シグニチャ ファイルの構成に関する一般的なガイドラインを示します。

- シグニチャ ファイルの先頭行は、次のようになります。

  # Cisco Signature File v.<メジャー バージョン>.<マイナー バージョン>

  バージョン番号は、シグニチャを作成した対象のシステム パーサー バージョンに一致する必要があります。現在のリリースでは 1.0 です。

- ファイルの行は、改行区切り形式になります。
- ポンド記号（#）で行を開始することにより、ファイルに 1 行コメントを追加できます。ファイルの必須の先頭行以外で、ポンド記号から始まる行はシグニチャの動作の影響を受けません。
- 各シグニチャは、「シグニチャの構文」（P.6-10）で説明する構文ルールに準拠する必要があります。
- シグニチャ ファイルに拡張（非 7 ビット ASCII）文字を使用するには、テキスト エディタから UTF-8 エンコードを使用するようにシグニチャ ソース ファイルを設定する必要があります。

シグニチャの構文

シグニチャの定義の形式は、次のとおりです。

```
X-<sig_group_id>.<sig_id>: <quick_pattern>
  <attribute-name>: <attribute-value>
  <attribute-name>: <attribute-value>
  ...
```

上記の定義で、

- <sig_group_id> は、シグニチャが属するシグニチャ グループの名前です。組み込みリソースから固有性を確保するため、作成するシグニチャ グループ名は必ず「X-」から始めます。「X-」プレフィクスを含む名前は全体で 15 文字以内であることが必要です。シグニチャ グループを別個に宣言する必要はありません。
- <sig_id> は、シグニチャの ID です。この ID は 15 文字以内にする必要があります。いずれのシグニチャ グループまたはシグニチャ ID にも、英字（大文字と小文字を区別）、数字、下線文字（_）、およびハイフン文字（-）を使用できます。ID は、同じグループ内のシグニチャ間で固有であり、シグニチャを説明するものであることが必要です。
- <quick_pattern> は、正規表現構文のサブセットを使用して照合するコンテンツを識別するパターン表現です。シグニチャには、少なくとも 1 つの簡易パターンまたは正規表現（regex 属性）が必要であり、両方を持つこともできます。両方が存在する場合は、簡易パターンが最初に評価されます。メッセージに一致した場合は、最終チェックとして regex パターンが一致部分と比較されます。
- シグニチャのすべての行で、コロンと、トークンに続く最初の空白スペース以外の文字の間の空白スペースはスキップされます。その他の改行までのすべての空白スペースは意味を持ちます。

- シグニチャには、次のオプション属性のいずれかを付けることができます。
  - regex は、メッセージ コンテンツの照合に使用される PCRE 形式の正規表現です。regex 属性はオプションですが、シグニチャには少なくとも 1 つの regex または簡易パターンが必要です。
  - name は、Web コンソールに表示されるシグニチャの記述名です。
  - cve は、シグニチャが対処する脆弱性を示すオプションの Common Vulnerabilities and Exposures（CVE）ID です。この情報は、シグニチャを文書化するのに役立ちます。
  - nocase は、正規表現の比較で大文字と小文字を区別するかどうかを決定します。デフォルトでは、大文字と小文字を区別しない方法で比較が実行されます。この属性を取り入れ、`false` に設定すると、大文字と小文字を区別した評価を行うことができます。

  各属性は専用の行に置き、先頭に 1 つまたは複数の空白スペースを付ける必要があります。属性はオプションですが、Web コンソールで使用できるようにすべてのシグニチャに name 属性を付けることをお勧めします。
- Manager と適切に相互運用するように、シグニチャの固定文字部分は 31 文字以内にする必要があります。
- シグニチャ グループ ID はポリシー上で固有でなければなりませんが、シグニチャ ID はグループのコンテキスト内でだけ固有である必要があります。ルールは完全修飾のシグニチャ ID によって個々のシグニチャを参照できます。完全修飾のシグニチャ ID は、<SigGroupID>.<SigID> のように、グループとシグニチャの両方の ID で構成されます。ただし、ほとんどの場合、ルールはシグニチャ グループ全体を参照します。

シグニチャ ファイルの例

次に、シグニチャ リスト ファイルの例を示します。

例 6-1　　シグニチャ リスト ファイル

```
# Cisco Signature File v. 1.8
# Copyright Cisco Systems, Inc. All Rights Reserved

X-NextThreat_b.exec:exec
 regex: ;\s*\bexec\b
 nocase: true
 name: exec nextthreat command attack
X-NextThreat_b.kick: kick
 nocase: true
 name: kick nextthreat command attack
X-NextThreat_m.start:start
 nocase: true
 name: start nextthreat command attack
X-NextThreat_m.cmd: cmdshell
 nocase: true
 name: cmdshell nextthreat command attack
X-PastThreat_m.cmd: dial
 nocase: true
 name: dial command attack
X-Privacy.USSocialSec:\d\d\d-\d\d-\d\d\d\d
 name: Social Security number screening
```

この例では、X-NextThreat_b、X-NextThreat_m、X-PastThreat_m、および X-Privacy の 4 つのシグニチャ グループが作成されます。組み込みシグニチャの規則に従い、シグニチャ グループには、重大度レベルの最初の文字を付けたグループ名が指定されます。たとえば、重大度が basic の場合は「b」、moderate の場合は「m」が付けられます。

この例では nocase 属性が true に設定されていますが（シグニチャが大文字と小文字を区別されない方法で適用されることを意味します）、これはデフォルトの比較モードなので、この属性を必ずしも true に設定する必要はありません。

例 6-1 のテキストをテキスト ファイルにコピーして、「カスタム シグニチャのアップロード」（P.6-9）の説明のとおりに Manager にロードすることによって、このサンプル ファイルをロードすることができます。シグニチャをロードした後は、カスタム ルールでそれらのシグニチャを適用できます。

ルールの開発

ルールは、メッセージ コンテンツに適用されるシグニチャを指定します。カスタムのメッセージ インスペクション ルールおよびメッセージ リライト ルールがサポートされています。メッセージ インスペクション ルールは要求だけに適用することができ、メッセージ リライト ルールは応答メッセージだけに適用することができます。

ルールには、ルールのアクションの条件を定義する表現が含まれます。ほとんどの場合、この表現はチェックされるメッセージの部分の ID と、チェックに使用するシグニチャの参照で構成されます。また、関数、演算子、またはその他の詳細機能を含めることもできます。

シグニチャと同様に、カスタム ルールはテキスト ファイルに作成します。Manager にアップロードされたルール ファイルは、Manager 内で名前付きリソースとなります。ルール ファイルは Manager で 1 つのリソースとして処理されるので、多数のアプリケーション用に多数のルール グループを作成する必要がある場合は、ルール グループを複数のソース ファイルに整理すると便利です。

次の例は、ルール セット ファイルの形式を示しています。

例 6-2　　ルール セット ファイルの形式

```
# Cisco Rule File v. <major_version>.<minor_version>

GROUP X-<rulegroup_id>:<rulegroup_name>

X-<rulegroup_id>.<rule_name>:<message_inspection_scope> sig <siggroup_name>
 name: <rule_descriptive_name>
 sev: <severity_level>
```

ルール セット ファイルでは、先頭行を Cisco Rule ファイル宣言から始める必要があります。バージョン番号は、ルールが作成された対象のルール セット パーサーのバージョンを反映します。ルール パーサーの現在のバージョンは 1.0 です。GROUP キーワードはルール グループを宣言します。ルール グループには、1 つまたは複数のルールを含めることができます。

組み込みルール グループから固有性を確保するため、ユーザが作成したシグニチャ グループとルール グループには先頭に「X-」文字を付ける必要があります。ルールの定義は、ルールが属するグループの宣言から始まります。次に、検査するメッセージの部分と検査に使用するシグニチャを指定します。

例 6-3　　ルール セット ファイルの例

```
# Cisco Rule File v. 1.0

# Group declarations
GROUP X-InspectRules: My Inspection Rule group
GROUP X-RewriteRules: My Message Rewrite Rule group

# rule declarations
X-InspectRules.NextThreat1:REQUEST_PARAM_ALL sig X-NextThreat_b
 name: Parameter inspection for Next Threat
 sev: 0

X-InspectRules.NextThreat2:REQUEST_HEADER sig X-NextThreat_b
 name: Header inspection for Next Threat
 sev: 0

X-InspectRules.NextThreat3:REQUEST_ALL sig X-NextThreat_m | X-PastThreat_m
 name: Closer Inspection for Next Threat
 sev: 1

# rewrite rules
X-RewriteRules.SsnMask: rewrite X-Privacy
 name: Prevents SSN leak
 rewriteChar: X
 desc: Finds and rewrites SSN
```

この例では、3 つのメッセージ インスペクション ルールとメッセージ リライト ルールを定義します。メッセージ インスペクション ルールには sev、つまり重大度プロパティが含まれていることに注意してください。プロファイルにルールの sev 値があることで、重大度に基づいてルール グループから選択的にルールを適用できるようになります。重大度の値は 0（basic）、1（moderate）、または 2（strict）です。メッセージ リライト ルールには、rewriteChar 値が含まれます。この値を使用して、メッセージ内で一致したコンテンツの各文字を置き換えます。

最初の 2 つのメッセージ インスペクション 検査ルールは、シグニチャで検査するメッセージ部分によって区別されます。

例 6-3 のテキストをテキスト ファイルにコピーして、「カスタム ルールのアップロード」（P.6-14）の説明のとおりに Manager にロードすることによって、このサンプル ルールをロードすることもできます。ルールのロードを試みる前に、例 6-1 に示されたシグニチャをアップロードする必要があります。ルールをロードした後は、プロファイルでルールを適用できます。

既存のルールの表示

シグニチャと同様に、独自のルールを作成するときは、多くの場合、組み込みルールの定義で提供されている例から始めると便利です。ルールの定義を表示するには、[Rules & Signature] ページでルールの横の [Details] リンクをクリックしてから、詳細ページの下部にある [View Source] ボタンをクリックします。ルール セットの未加工のソース コードが表示されます。

カスタム ルールのアップロード

ルールの作成が終了したら、次のセクションの説明にあるとおり、それらのルールを Manager にアップロードしてポリシーで使用できるようにする必要があります。

カスタム ルール ファイルをアップロードするには、次の手順を実行します。

ステップ 1　[Rules & Signatures] ページで [New Custom Rules] ボタンをクリックします。

ステップ 2　ルール リソースの記述名を入力します。名前は、ポリシー内のルール リソースに固有である必要があります。この記述名がルール リソース リストに表示されるので、Web コンソールの他のユーザにとってわかりやすい名前にします。

ステップ 3　外部のテキスト ファイルからルールをロードするには、[Browse] ボタンをクリックしてファイル システム上のファイルに移動することにより、ファイルを識別します。

ステップ 4　[Load File Contents] ボタンをクリックします。

ルール ソース コードが [Rule Definitions] フィールドに表示されます。ルール ソース コードをこのフィールドに直接入力したり、貼り付けたりすることができますが、ほとんどの場合は外部のテキスト ファイルからロードされます。

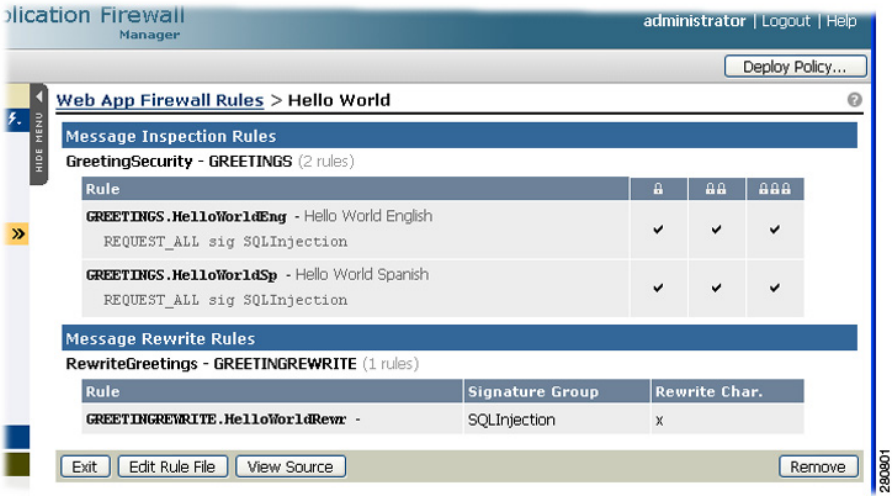
ステップ 5　必要に応じて、[Rule Definitions] フィールドのルール ソース コードに直接変更を加えます。このフィールドでの変更は、元のルール ソース ファイルには伝播されません。

ステップ 6　[Save Changes] をクリックします。

Manager によってルールが検証されます。検証エラーは、ページの上部に表示されます。たとえば、ルールがロードされていないシグニチャを参照している場合、ルールの検証エラーが発生します。

図 6-2 にあるとおり、この時点でカスタム ルールが Web コンソール ページに表示されます。カスタム ルールはプロファイルにも表示され、ここでネットワーク トラフィックに適用することが可能になります。

図 6-2　Web コンソール上のカスタム ルール



[Rules & Signatures] ページでは、カスタム ルールの名前の横にある [Details] リンクをクリックすることにより、リソースのソース コードを表示できます。ポリシーをロードした後は、詳細ページの下部にある [Edit Rule File] ボタンをクリックして、ルール ソース コードを変更することができます。ルール ファイルを編集して新しいルールの追加や、ルール セット リソースのルールの変更または削除を行うことができます。ルール グループを削除すると、ルール グループが有効であるプロファイルを含む、すべてのプロファイルからルールのアベイラビリティが削除されます。

ルールの作成に関する一般的なガイドライン

アップロードした各ルール ファイルは、ポリシー内で 1 つの名前付きリソースとなります。そのため、Manager のバージョンコントロールおよびポリシー アーカイブ機能の影響を受けます。次に、ルール ファイルの構成に関する一般的なガイドラインを示します。

- ルール ファイルの先頭行は、次のようになります。

  # Cisco Rule File v.<メジャー バージョン>.<マイナー バージョン>

  バージョンは、ルールが作成された対象のシステムのパーサー バージョンと一致する必要があります。現在のリリースでは 1.0 です。

- ポンド記号（#）で行を開始することにより、ファイルに 1 行コメントを追加できます。ファイルの必須の先頭行以外で、ポンド記号から始まる行はルール セットの動作の影響を受けません。
- ルール セットには、改行区切り形式で、1 つまたは複数のルールおよびルール グループの宣言を含めることができます。
- 管理性を保つため、リライト ルールとメッセージ インスペクション ルールを同じグループに入れることはできません。
- ルール ファイルのほとんどの要素が `desc` プロパティを持つことができます。このプロパティは、Manager ユーザ インターフェイスでの項目の文書化に役立てるためのものです。
- 重大度レベルは、sev 属性によって示されます。可能な値は、0（basic）、1（moderate）、または 2（strict）です。

ルールの構文

前述のとおり、Web アプリケーションのセキュリティ用に次の 2 タイプのルールを作成できます。

- メッセージ インスペクション ルール
- メッセージ リライト ルール

各タイプの構文には、わずかな違いがあります。ただし、次に示すとおり、どちらの構文も属するルール グループの識別、ピリオド、ルール独自の ID の順になります。

```
SQLINJECTION.SqlInjection1:REQUEST_ALL sig SQLInjection
...
```

この例では、`SqlInjection1` という新しいルールがルール グループ `SQLINJECTION` に宣言されています。名前の後にはコロン（:）が付き、その後にルール文が続きます。ルール文には、検査されるメッセージの部分と、適用されるシグニチャが含まれます。

この例は、メッセージ インスペクション ルールです。メッセージ リライト ルールは、同様の ID の配列で始まります。ただし、キーワード `rewrite` がコロンの後に続きます。

次のセクションでは、メッセージ構文について詳しく説明します。

ルール グループ宣言

グループ宣言は、ルール グループのプロパティを定義します。グループ宣言に適用できるプロパティは、説明プロパティの desc だけです。

ルール宣言内のルール グループ ID 参照に対応する GROUP 宣言がない場合、ルール グループが自動的に作成されることに注意してください。その場合、名前は ID と同じ値に設定され、説明は空になります。

ルール グループは、次の形式で宣言されます。

```
GROUP X-<rule_group_id>: <group name>
 <wsp>desc: <description><newline>
```

宣言は、次のコンポーネントで構成されます。

- GROUP：文をグループの定義として識別するキーワード。
- <rule_group_id>：ルール グループの ID。ID には、次の特徴があります。
  - 組み込みルール グループから固有性を確保するため、グループ ID は「X-」から始める必要があります。
  - ID は、必須の「X-」プレフィクスを含め、15 文字以内にする必要があります。
  - ルール グループ ID は、すべてのルール グループ間で固有でなければなりません。
  - ID に使用できる文字は、大文字または小文字の英字、数字、下線文字（_）、およびハイフン（-）文字です。
- <group_name>：Manager インターフェイスに使用するため、ユーザにわかりやすいルール グループの名前（「SQL Injection」など）にします。
- <wsp>：グループ宣言属性の前に空白スペースまたはタブを置きます（サポートされるグループ属性は desc だけです）。
- <description>：ルール グループの説明。この説明は、ルール ファイルをアップロードした後にユーザ インターフェイスに表示されます。

メッセージ インスペクション ルールの形式

メッセージ インスペクション ルールには、次の形式を使用します。

```
<rule_group_id>.<rule_id>: <rule_statement>
  <attribute-name>: <attribute-value>
  <attribute-name>: <attribute-value>
  ...
```

ルールは、次のコンポーネントで構成されます。

- <rule_group_id>：ルールが属するルール グループの ID。グループに別個のルール グループ定義が存在しない場合は、ルール グループが作成されます。組み込みルール グループから固有性を確保するため、作成するグループ名を必ず「X-」から始めます。「X-」プレフィクスを含む名前は全体で 15 文字以内であることが必要です。
- <rule_id>：ルール ID。ID には、次の特徴があります。
  - ID は、15 文字以内にする必要があります。
  - ルール ID は、グループ内のすべてのルール間で固有である必要があります。
  - ID に使用できる文字は、大文字と小文字を区別した英字、数字、下線（_）、およびハイフン（-）です。

- <rule_statement>：有効な Reactor 表現文を定義する文字列。この文の詳細については、「ルール文の形式」（P.6-18）を参照してください。
- ルール属性：ルールには、名前 - 値のペアの形式でルール属性を付けることができます。各属性は独自の行を占有します。先頭に 1 つまたは複数の空白スペースを付ける必要があります。属性の形式は、属性名、コロン、属性値、改行の順になります。

  メッセージ インスペクション ルールには、次の属性を付けることができます。

  - name：ユーザにとってわかりやすいルールの名前。
  - desc：Web コンソール ユーザ用にルールを文書化するルールの説明。
  - severity：ルールの重大度レベル。重大度レベルを使用する場合は、さまざまなレベルの厳密さまたは完全さで所定のセキュリティ タスクに対処する複数のルールを定義します。それによってプロファイルは、所定のバックエンド アプリケーションに必要な重大度レベルのルールを適用することができます。

メッセージ リライト ルールの形式

メッセージ リライト ルールは、メッセージ全体に適用するのではなく、シグニチャに一致したメッセージ コンテンツの部分を変更します。シグニチャは、1 つのメッセージで複数回一致する場合があります。最も一般的に、これらのルールは、メッセージ内の重要なコンテンツを識別し、上書きするために使用されます。

リライト ルールでは、簡易パターンだけを使用してコンテンツを照合できます。詳細については、「簡易パターンについて」（P.6-8）を参照してください。

メッセージ リライト ルールの形式は、次のとおりです。

```
<rule_group_id>.<rule_id>: rewrite <sig_group_id>
  <attribute-name>: <attribute-value>
  <attribute-name>: <attribute-value>
  ...
```

ルールは、次のコンポーネントで構成されます。

- <rule_group_id>：ルールが属するルール グループの ID。グループに別個のルール グループ定義が存在しない場合は、ルール グループが作成されます。
- <rule_id>：ルール ID。ID には、次の特徴があります。
  - ID は、15 文字以内にする必要があります。
  - ルール ID は、グループ内のすべてのルール間で固有である必要があります。
  - ID に使用できる文字は、大文字と小文字を区別した英字、数字、下線（_）、およびハイフン（-）です。
- rewrite：rewrite キーワードは、ルールをメッセージ リライト ルールとして識別します。
- <sig_group_id>：このルール内で適用するシグニチャのシグニチャ グループ ID。シグニチャは、このルールが適用されるメッセージ内でリライトされるコンテンツを決定します。

  リライト ルールは、DFA 表現を使用するシグニチャだけを適用できます。PCRE 表現は適用できません。したがって、正規表現「\d{4,6}」は使用できず、それぞれパターンの長さが異なる 3 つのシグニチャに置き換える必要があります。

- ルール属性：ルールには、名前 - 値のペアの形式でルール属性を付けることができます。各属性は独自の行を占有します。先頭に 1 つまたは複数の空白スペースを付ける必要があります。属性の形式は、属性名、コロン、属性値、改行の順になります。

  リライト ルールには、次の属性を付けることができます。

  - name：ユーザにとってわかりやすいルールの名前。
  - desc：Web コンソール ユーザ用にルールを文書化するルールの説明。
  - rewriteChar：シグニチャで一致したパターンを置換するために使用する値。rewriteChar は、一致したパターンのすべての文字に対して 1 度だけ書き込まれるので、置換文字が「x」である場合、一致した文字列 123-456-789-1011 は xxxxxxxxxxxxxxx としてリライトされます。

ルール文の形式

ルール文は通常、true に解決された場合にルールのアクションを始動する、条件付きの表現です。ルール文は、ルール ID に続くコロンの後に表示されます。

```
<rule_group_id>.<rule_id>: <rule_statement>
```

ルール文の形式は次のとおりです。

```
<variable-expr> <operator> <test-value>
```

次に例を示します。

REQUEST_PARAM_ALL sig X-NextThreat_b

この場合、`sig` は演算子です。この演算子は、シグニチャ名またはシグニチャ グループ名（-NextThreat_b）によって指定されるテスト値をテストされる値（すべてのパラメータ）に適用します。ルール文では、ルール照合以外の要素に基づいてテストを構成することができます。たとえば、次の文は、要求パラメータの数をチェックします。

```
REQUEST_PARAMS.count() gt 128
```

この例は、要求内のパラメータの数が 128 を超えるかどうかをテストします。true の場合、ルールのアクションが始動します。この例は、次のコンポーネントで構成されています。

- 変数 REQUEST_PARAMS
- 関数 count()
- 演算子 gt

以下のセクションに示すとおり、これらのルール表現機能は、カスタム ルール動作の定義に大きな柔軟性をもたらします。

- ルールの演算子
- ルール変数
- ルール関数

ルールの演算子

次の演算子は、所定のテスト値とテストされる値にバイナリ テストを適用します。テストが false に解決されると、テストされたコンテンツはルールの適用から除外されます。次の表にルールの演算子を示します。

表 6-1　　ルールの演算子

演算子	テストされる値が次の場合に true になる
eq	等しい
gt	より大きい
gte	以上
lt	より小さい
lte	以下
re	正規表現
nre	正規表現でない
sig	値が SignatureGroup の名前であるシグニチャ グループ

ルール変数

ルール変数は、ランタイム情報へのアクセスを可能にします。たとえば、要求パラメータ、ヘッダー、およびクライアント IP などのクライアント情報に対応する変数があります。変数を利用して、これらのランタイム値に基づいたルール条件を作成できます。

ルール構成のほか、ポリシーのその他のエリアでも変数を使用できます。たとえば、HTML 応答で、$(varname) の形式で変数を参照することにより、カスタム エラー応答メッセージに変数を組み込むことができます。また、ヘッダー処理フィールドでも変数を使用できるので、変数値を HTTP ヘッダーとして発信メッセージに追加できます。

（注） ルールを開発するときは、カスタム エラー応答でルール変数をテストして、トラフィックに対して各変数が返す値のタイプをすばやく検出できるようにすると便利です。

特定の変数は、次の形式で要求に関連付けられた名前付きパラメータの値へのアクセスを可能にします。

VARIABLE_NAME['param']

param は、Web フォーム変数の名前です。

次に例を示します。

REQUEST_HEADER['Referer'] sig CrossScript_m

この場合、HTTP 要求ヘッダー `Referer` は、明確に CrossScript_m 内のシグニチャに一致するコンテンツの有無をチェックします。要求パラメータ値は、POST 形式で識別されたとおりのパラメータの名前を渡すことによって、同様に取得できます。

次の表に、ルールの開発に使用できる変数を示します。戻り値の例が示されている場合、その戻り値は次の要求 URL に基づいています。

http://example.com:8080/details.do?v1=value1&v2=value2&v3=value3

表 6-2　ルール変数

変数	説明
CLIENT_IP	要求を作成したクライアントの IP。ドット付きの 4 つの数字列になります。
CLIENT_PORT	クライアントが要求を送信する宛先のリスニング ポート
SERVER_IP	ルーティングするサーバの IP。ドット付きの 4 つの数字列になります。
SERVER_PORT	アドレス指定される宛先サーバのリスニング ポート
REQUEST_ALL	URL パス、すべてのパラメータ、すべてのヘッダーの収集。適切な場合、それぞれ名前 - 値の形式になります。
REQUEST_LINE	方式と HTTP バージョンを含む完全な要求行
REQUEST_URL_RAW	次のような、正規化前の要求 URL /details.do?v1=value1&v2=value2&v3=value3
REQUEST_URL_FULL	次のような、正規化後の完全な要求 URL /details.do
REQUEST_QUERY_RAW	要求のクエリー文字列部分。クエリー文字列は、次のような、疑問符の後の要求 URL の部分です。 v1=value1&v2=value2&v3=value3
REQUEST_URL_PATH	次のような、要求 URL のパス部分 /details.do
REQUEST_URL_HOST	要求 URL のホスト部分（存在する場合）
REQUEST_HOST	URL または Host ヘッダーからの要求のホスト
REQUEST_PARAM	次のような、パラメータ名をキーとする、正規化した後のすべてのパラメータ値の収集（メッセージに GET と POST の両方の引数が含まれる場合、その両方が存在します） value1, value2, value3
REQUEST_PARAM_NAMES	すべてのパラメータ名の収集
REQUEST_PARAM_ALL	次のような、パラメータ名をキーとする、すべてのパラメータ名と値の収集 v1, v2, v3, value1, value2, value3
REQUEST_GETPARAM	次のような、パラメータ名をキーとする、正規化後のすべての GET パラメータ値の収集 value1, value2, value3
REQUEST_GETPARAM_NAMES	パラメータ名をキーとする、すべての GET パラメータ名の収集

変数	説明
REQUEST_GETPARAM_ALL	次のような、パラメータ名をキーとする、すべての GET パラメータ名と値の収集 v1, v2, v3, value1, value2, value3
REQUEST_POSTPARAM	パラメータ名をキーとする、正規化後のすべての GET パラメータ値の収集
REQUEST_POSTPARAM_NAMES	パラメータ名をキーとする、すべての GET パラメータ名の収集
REQUEST_POSTPARAM_ALL	パラメータ名をキーとする、すべての GET パラメータ名と値の収集
REQUEST_HEADER	ヘッダー名をキーとする、すべての要求ヘッダー値の収集
REQUEST_HEADER_NAMES	すべての要求ヘッダー名の収集
REQUEST_HEADER_ALL	ヘッダー名をキーとする、すべての要求ヘッダー名と値の収集
COOKIE	cookie 名をキーとする cookie 値
COOKIE_NAMES	すべての cookie 名の順序付きリスト
COOKIE_ALL	名前をキーとする、すべての cookie 名と値の収集
REQUEST_METHOD	要求方式
REQUEST_VERSION	要求の HTTP バージョン
REQUEST_LENGTH	要求の長さ
AUTH_RAW	方式および Base 64 リージョンを含む、Authorization ヘッダーの未加工のデータ
AUTH_METHOD	認証方式（Basic など、Authorization ヘッダーから取得）
AUTH_USER	Authorization ヘッダーからのユーザ名
AUTH_PASSWORD	Authorization ヘッダーからのパスワード値
AUTH_DATA	Authorization ヘッダーからのデータ。つまり、認証方式の後の値です。
SSL_CIPHER	クライアントの Secure Socket Layer（SSL）接続に使用される暗号
SSL_CERT	PEM エンコードのクライアント認証（存在する場合）
SSL_CERT_VERIFIED	クライアント認証が確認されたかどうかを示すブール値
SSL_CERT_REVOKED	クライアント認証が無効にされたかどうかを示すブール値
SSL_DIGEST_SHA1	クライアントの SSL 認証の SHA1 ダイジェスト（存在する場合）
SSL_SUBJECT_DN	クライアントの SSL 認証のサブジェクト認定者名（存在する場合）
SSL_ISSUER_DN	クライアントの SSL 認証の発行元認定者名（存在する場合）
SSL_SUBJECT_ALTNAME	クライアントの SSL 認証のサブジェクト代替名（存在する場合）

変数	説明
SSL_SUBJECT_KEYID	クライアントの SSL 認証のサブジェクト キー識別子（存在する場合）
SSL_PEER_CERT_EXTENSION	名前をキーとする、クライアントの SSL 認証の拡張値の収集
SSL_PEER_CERT_EXTENSION_OID	OID をキーとする、クライアントの SSL 認証の拡張値の収集
SSL_PEER_CERT_EXTENSION_NAMES	名前をキーとする、クライアントの SSL 認証の拡張名の収集
SSL_FINGERPRINT_CHAIN	クライアントの SSL Certificate Authority（CA; 認証局）検証チェーン内のフィンガープリントの順序付きリスト
SSL_VERSION	SSL セッションのバージョン。「TLSv1」、「SSLv2」、「SSLv3」のいずれか、または空文字列になります。
REQUEST_MULTIPART_HEADERS	名前をキーとする、マルチパート/フォームデータ メッセージの部分のすべてのヘッダーの収集。すべての部分が 1 つに集約されるので、REQUEST_MULTIPART_HEADERS['Content-type'] はすべての部分のすべてのコンテンツタイプを返します。
REQUEST_MULTIPART_DISPOSITIONS	マルチパート/フォームデータ メッセージのすべての部分の Content-Disposition ヘッダーの単純値。この値は通常、「form-data」です。
REQUEST_MULTIPART_DISPOSITION_NAMES	マルチパート/フォームデータ メッセージのすべての部分のすべての Content-Disposition ヘッダーの名前パラメータ
REQUEST_MULTIPART_DISPOSITION_FILENAMES	マルチパート/フォームデータ メッセージのすべての部分のすべての Content-Disposition ヘッダーのファイル名パラメータ
REQUEST_BODY_RAW	正規化前の要求の本体部分
REQUEST_BODY_XML	XML コンテンツとしての要求の本体部分
RESPONSE_HEADER	ヘッダー名をキーとする、すべての応答ヘッダー値の収集
RESPONSE_HEADER_NAMES	すべての応答ヘッダー名の収集
RESPONSE_HEADER_ALL	ヘッダー名をキーとする、すべての応答ヘッダー名と値の収集
REQUEST_PATH_AND_QUERY_RAW	正規化前の要求 URL のリソース パスとクエリーの部分

ルール関数

ルール関数は、ルールを評価する前にルール値に特別な処理演算を適用します。

ルール構成には、次の関数を使用できます。これらの関数は、カスタム エラー応答テキストやヘッダー処理フィールドなど、ポリシーのその他のコンテキストにも使用できます。エラー応答の変数値に関数を適用するには、$(REQUEST_PARAMS.count()) のように、$(varname.function) の形式で入力します。

表 6-3　ルール関数

関数	説明
count()	REQUEST_PARAMS.count() など、変数によって識別される項目の数を返します。
size()	変数によって参照される項目の合計サイズを返します。
maxsize()	変数によって参照される項目の最大値のサイズを返します。
urlencode()	変数によって参照されるすべての値を URL でエンコードされた同等の値に変換します。
normalize(*expr*)	expr パラメータによって指定された項目を正規化します。この関数は、値を正規化することによって値を標準形に変換し、勝手に変更されたコンテンツや有害の可能性があるコンテンツを取り除きます。詳細については、「normalize() 関数」（P.6-4）を参照してください。