Protection Against Content Based Attacks

Protection against content based attacks

- Message content is a significant attack vector used by malicious API consumers.
  - JSON threat protection
  - XML threat protection
  - general content protection
- API services provides a set of policy types to mitigate the potential for your backend services to be compromised by attackers or by malformed request payloads.

Content based security

JSON threat protection

JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.

JSONThreatProtection Policy

XML threat protection

XML attacks attempt to use structures that overwhelm XML parsers to crash a service and induce application-level denial-of-service attacks.

XMLThreatProtection Policy

General content protection

Some content-based attacks use specific constructs in HTTP headers, query parameters, or payload content to attempt to execute code.

An example is SQL-injection attacks.

RegularExpressionProtection Policy

JSON threat protection policy

- APIs that support JavaScript object notation (JSON) are vulnerable to content-level attacks.
- Simple JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.
- JSONThreatProtection policy minimizes the risk posed by content-level attacks by enabling you to specify limits on various JSON structures, such as arrays and strings.
- All settings are optional and should be tuned to optimize your service requirements against potential vulnerabilities.

```
<JSONThreatProtection async="false" continueOnError="false"
enabled="true" name="JSON-Threat-Protection-1">
   <DisplayName>JSON Threat Protection 1</DisplayName>
   <ArrayElementCount>20</ArrayElementCount>
   <ContainerDepth>10</ContainerDepth>
   <ObjectEntryCount>15</ObjectEntryCount>
   <ObjectEntryNameLength>50</ObjectEntryNameLength>
   <Source>request</Source>
   <StringValueLength>500</StringValueLength>
</JSONThreatProtection>
```

XML threat protection policy

- XMLThreatProtection policy minimizes the risk posed by content-level attacks on XML payload.
- Optionally, detect XML payload attacks based on configured limits.
- Screen against XML threats using the following approaches:
  - Validate messages against an XML schema (.xsd)
  - Evaluate message content for specific blacklisted keywords or patterns
  - Detect corrupt or malformed messages before those messages are parsed

```
<XMLThreatProtection async="false" continueOnError="false"
enabled="true" name="XML-Threat-Protection-1">
   <DisplayName>XML Threat Protection 1</DisplayName>
   <NameLimits>
      <Element>10</Element>
      <Attribute>10</Attribute>
      <NamespacePrefix>10</NamespacePrefix>

<ProcessingInstructionTarget>10</ProcessingInstructionTarget
>
   </NameLimits>
   <Source>request</Source>
   <StructureLimits>
      <NodeDepth>5</NodeDepth>
      <AttributeCountPerElement>2</AttributeCountPerElement>
      <NamespaceCountPerElement>3</NamespaceCountPerElement>
      <ChildCount includeComment="true"
includeElement="true" includeProcessingInstruction="true"
includeText="true">3</ChildCount>
   </StructureLimits>
   <ValueLimits>
      <Text>15</Text>
      <Attribute>10</Attribute>
      <NamespaceURI>10</NamespaceURI>
      <Comment>10</Comment>

<ProcessingInstructionData>10</ProcessingInstructionData>
   </ValueLimits>
</XMLThreatProtection>
```

apigee

Regular expression protection policy

- Some content-based attacks use specific constructs in HTTP headers, query parameters, or payload content to attempt to execute code.
- An example is SQL injection attacks.
- Such attacks can be mitigated using the RegularExpressionProtection Policy type
- Extracts information from a message (for example, URI Path, Query Param, Header, Form Param, Variable, XML Payload, or JSON Payload) and evaluates that content against predefined regular expressions.
- If any specified regular expressions evaluate to true, the message is considered a threat and is rejected.

```xml
<RegularExpressionProtection async="false" continueOnError="false"
enabled="true" name="Regular-Expression-Protection-1">
     <DisplayName>Regular Expression Protection 1</DisplayName>
     <Source>response</Source>
     <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
     <URIPath>
         <Pattern>REGEX PATTERN</Pattern>
     </URIPath>
     <QueryParam name="a-query-param">
         <Pattern>REGEX PATTERN</Pattern>
     </QueryParam>
     <Header name="a-header">
         <Pattern>REGEX PATTERN</Pattern>
     </Header>
     <FormParam name="a-form-param">
         <Pattern>REGEX PATTERN</Pattern>
     </FormParam>
     <Variable name="request.content">
         <Pattern>REGEX PATTERN</Pattern>
     </Variable>
     <XMLPayload>
         <Namespaces>
             <Namespace prefix="apigee">http://www.apigee.com</Namespace>
         </Namespaces>
         <XPath>
             <Expression>/apigee:Greeting/apigee:User</Expression>
             <Type>string</Type>
             <Pattern>REGEX PATTERN</Pattern>
         </XPath>
     </XMLPayload>
     <JSONPayload>
         <JSONPath>
             <Expression>$.store.book[*].author</Expression>
             <Pattern>REGEX PATTERN</Pattern>
         </JSONPath>
        </JSONPayload>
</RegularExpressionProtection>
```

apigee

Example blacklist patterns

Because we configure policies in XML, your regular expressions must be URL Encoded

Name	Regular Expression
SQL Injection	`[\s]*((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b))`

Server-Side Include Injection	`<!--\s*<!--(include|exec|echo|config|printenv)\s+.*`
	`XML encoded: &lt;!--\s*&lt;!--(include|exec|echo|config|printenv)\s+.*`

XPath Abbreviated Syntax Injection	`(/(@?[\w_?\w:\*]+(\+\])*)?)+`
XPath Expanded Syntax Injection	`/?(ancestor(-or-self)?|descendant(-or-self)?|following(-sibling))`

JavaScript Injection	`<\s*script\b[^>]*>[^<]+<\s*/\s*script\s*>`
	`XML encoded: &lt;\s*script\b[^&gt;]*&gt;[^&lt;]+&lt;\s*/\s*script\s*&gt;`

Java Exception Injection	`.*Exception in thread.*`

apigee

Thank You

Google Cloud