Security Testing For RESTful Applications

Ofer Shezaf, HP Enterprise Security Products

ofr@hp.com
What I do for a living:
• Product Manager, Security Solutions, HP ArcSight
• Led security research and product management at Breach Security & HP Fortify

I am passionate about security after hours as well:
• OWASP leader and founder of the Israeli chapter
• Leads the Web Application Firewall Evaluation Criteria project
• Wrote the ModSecurity Core Rule Set

Fun fact: the closest airport to my house is in Damascus, Syria
In this Presentation

About RESTful Web Services
RESTful WS in the Wild
Security of RESTful WS
Pen-testing RESTful WS
Automated security testing of RESTful WS
About RESTful Web Services

- About RESTful Web Services
- RESTful WS in the Wild
- Security of RESTful WS
- Pen-testing RESTful WS
- Automated security testing of RESTful WS
Web Services

Employing web technology (i.e. HTTP) for machine to machine communication

Used for:
- Inter application communication
- Web 2.0 and Mashups
- Think client applications
- Phone applications
SOAP Web Services: example

Highly defined

Parameters are sent as a well formed XML

Isn’t this a rather complex way to send a single parameter?

```xml
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf8
Content-Length: 299

SOAPAction: "http://www.w3.org/2003/05/soap-env

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-env"
    xmlns:m="http://www.example.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/GetStockPrice.xsd">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.example.org/GetStockPrice.xsd"
        StockName="IBM">
      <m:GetStockPrice/>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```
SOAP Web Services

Commonly used protocol set for Web Services

**The theory:**
- Structures and well defined
- Robust
- Secure (?)

**However:**
- Complex and heavy, especially for phone and Web 2.0
- Not the HTTP way: Designed to work on any protocol including SMTP

* See [WS-Attacks.org](http://WS-Attacks.org) for an alternative view
The **REST design pattern**

Essentially what the Web always was

| **Client/Server** | •Clients are separated from servers by a uniform interface. |
| **Stateless** | •The client–server communication is further constrained by no client context being stored on the server between requests*. |
| **Cacheable** | •Responses must therefore, implicitly or explicitly, define themselves as cacheable or not |
| **Layered** | •A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. |
| **Uniform** | •A uniform interface between clients and servers simplifies and decouples the architecture. |
| **Code on demand** | •Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. |

* The server can be stateful; this constraint merely requires that server-side state be addressable by URL as a resource.
RESTful Web Services

Are:

• A common practice for using REST design patterns for Web Services

Are Not:

• A well defined protocol
• A set of software libraries or frameworks
RESTful Web Services: example

Isn’t this much simpler?

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-env"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-env"
                xmlns:m="http://www.example.org"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.example.org/GetStockPrice.xsd" />
    <soap:Header>
    </soap:Header>
    <soap:Body>
      <m:GetStockPrice xmlns:m="http://www.example.org"
                       xsi:schemaLocation="http://www.example.org/GetStockPrice.xsd"/>
    </soap:Body>
  </soap:Envelope>

GET /InStock/HP
Common RESTful WS Practices

Use of HTTP methods to indicate action

**CRUD:**
- Create (PUT),
- Read (GET),
- Update (POST),
- Delete (DELETE)

```
GET /InStock/HP
```

```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signature=Valuem

<AccessControlPolicy>
  <Owner>
    <ID>ID</ID>
    <DisplayName>EmailAddress</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ID>ID</ID>
        <DisplayName>EmailAddress</DisplayName>
      </Grantee>
      <Permission>Permission</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```
**Common RESTful WS Practices**

None standard parameters specifications

- As part of the URL
- None standard request parameters
- In headers
- Serialized as JSON in a parameter value of request body

**GET /InStock/HP**

**PUT /ObjectName?acl HTTP/1.1**
```
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue
```

**PUT /destinationObject HTTP/1.1**
```
Host: destinationBucket.s3.amazonaws.com
x-amz-copy-source: /source_bucket/sourceObject
x-amz-metadata-directive: metadata-directive
x-amz-copy-source-if-match: etag
x-amz-copy-source-if-none-match: etag
x-amz-copy-source-if-unmodified-since: time
x-amz-copy-source-if-modified-since: time
```

**Authorization: signatureValue**
```
Date: date
```

© Copyright 2012 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
Common RESTful WS Practices

Structured parameters and responses

- JSON and XML both widely used
- Parameter:
  - In the request body
  - Embedded in the value of a single parameter
- Response usually in the response body

http://api.geonames.org/earthquakesJSON?north=44.1&south=-9.9&east=-22.4&west=55.2&username=demo

```json
{"earthquakes": [
    ...
]}
```
Common RESTful WS Practices

Custom authentication and session management

- Commonly use security token/tickets
- While pure REST calls for URL based tokens, this is not secure and headers are often used.

```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue

<AccessControlPolicy>
<Owner>
```

```
PUT /destinationObject HTTP/1.1
Host: destinationBucket.s3.amazonaws.com
x-amz-copy-source: /source_bucket/sourceObj:
x-amz-metadata-directive: metadataDirective
x-amz-copy-source-if-match: etag
x-amz-copy-source-if-none-match: etag
x-amz-copy-source-if-unmodified-since: time
x-amz-copy-source-if-modified-since: time

<request metadata>
Authorization: signatureValue
Date: date
```
RESTful services Documentation

• No common documentation format similar to WSDL.
• WADL (Web Application Definition Languages) is a standard proposal:
  • Not approved
  • Not widely used

```xml
<resources base="http://api.search.yahoo.com/NewsSearchService">
  <resource path="newsSearch">
    <method name="GET" id="search">
      <request>
        <param name="appid" type="xsd:string" style="query" required="true"/>
        <param name="query" type="xsd:string" style="query" required="true"/>
        <param name="type" style="query" default="all" option value="all"/>
        <param name="results" style="query" type="xsd:int" optio
        <param name="start" style="query" type="xsd:int" optio
        <param name="sort" style="query" default="relevance" option value="rank"/>
        <option value="date"/>
      </request>
    </method>
  </resource>
</resources>
```
RESTful WS in the Wild

• About RESTful Web Services
• RESTful WS in the Wild
• Security of RESTful WS
• Pen-testing RESTful WS
• Automated security testing of RESTful WS
It’s Up and Coming!

restful web services 1.00 soap web services 2.70

Search Volume index

Google Trends

Everybody uses REST
Security of RESTful WS

• About RESTful Web Services
• RESTful WS in the Wild
• Security of RESTful WS
• Pen-testing RESTful WS
• Automated security testing of RESTful WS

© Copyright 2012 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.
You Already Know This Part

REST is just Web

REST Security is just Web application security
Key issues to keep in mind

No standard security mechanism similar to SOAP Web Services (WS-*).

Proprietary authentication and session management.

Some common design flaws associated with REST:
- Overreliance on SSL
- Session IDs used in the URL
- Using basic HTTP Authentication
- Bad implementation of SSO
Pen-testing RESTful WS

- About RESTful Web Services
- RESTful WS in the Wild
- Security of RESTful WS
- Pen-testing RESTful WS
- Automated security testing of RESTful WS
Detecting Attack Surface

Challenges

Inspecting the application does not reveal application attack surface

- None Web applications
- Not all Web Service functionality actually used by application
- Requests are often dynamically created, Web 2.0 style.
Challenges

Mega fuzzing

A very large number of parameters to fuzz

- None Standard Parameters
- Serialized JSON/XML parameters
Challenges

Session management

Custom authentication and session management requires adjustment in every pen test.

Need to follow custom SSO processes and session management breaks common cookie sharing practices.
Solutions

Use Documentation

Determine:
- Available services
- Use of HTTP methods
- Use of parameters

Potential Sources:
- WADL
- Programming guides
- Configuration information
- Application source
Solutions

Use Documentation

Determine:

• Available services
• Use of HTTP methods
• Use of parameters

Potential Sources:

• WADL
• Programming guides
• Configuration information
• Application source

```xml
<resources base="http://ap1.search.yahoo.com/News/">
  <resource path="newsSearch">
    <method name="GET" id="search"/>
    <request>
      <param name="appid" type="xsd:string" style="query" required="true"/>
      <param name="query" type="xsd:string" style="query" required="true"/>
      <param name="type" style="query"/>
      <option value="all"/>
      <option value="any"/>
      <option value="phrase"/>
    </request>
  </resource>
</resources>

public interface IMDbMagazineService
{
  [OperationContract]
  [WebGet(UriTemplate="/" )]
  IssueCollection GetAllIssues();
  [OperationContract]
  [WebGet(UriTemplate = "/{year}/")]
  IssueData GetIssuesByYear(string year);
  [OperationContract]
  [WebGet(UriTemplate = "/{year}/issue/{issue}" )]
  Articles GetIssue(string year, string issue);
}
**Solutions**

Use a proxy

**Determining attack surface when no documentation exists**

Useful also when documentation exists to determine initial fuzzing values

Extremely helpful for:
- None web applications
- Dynamically generated requests
Guessing parameters

Look for none standard headers

Determine if URL segments have a pattern
- Numerical values
- Well known templates such as date

Look for structures in parameter values
- JSON, XML, YAML or other

URLs with not extension
Automated security testing of RESTful WS

• About RESTful Web Services
• RESTful WS in the Wild
• Security of RESTful WS
• Pen-testing RESTful WS

Automated security testing of RESTful WS
How does automated pen-testing work?

Crawling
- Determining attack surface
- Historically only links based
- Today employ JavaScript emulation to get dynamic requests

Attacking
- Sending known attack vectors
- Fuzzing parameters
- Session based

Understanding request generation (i.e., links)
Understanding parameters
Understanding session management
RESTful WS Challenges

- Finding attack surface by crawling
- Determining what elements of the request to attack
- Optimizing fuzzing time while still addressing all potential parameters
- Getting initial values for fuzzing
- Custom authentication and session management breaks common cookie sharing practices
One: define rules

Define parameter structure for URL
Use rules when crawling and attacking

Rule can be:

• User defined
• Imported documentation, WADL or configuration files
• Proxy discovered attack surface, potentially during crawl.

Or...

Get smart!

WebInspect 9.2 REST rule editor
Two: ask the server

A server module communicating with the scanner can:

- Identify rewrites
- Send configuration and debug information
- Provide file and method structure
- Monitor server based session information
Three: Look for highly varying URL segments

http://www.44tips.com:80/svc/Grid.asmx/GetContentItems
http://www.44tips.com:80/svc/Grid.asmx/GetRelatedListItems
http://www.44tips.com:80/svc/Grid.asmx/GetUserCollectionInfo
http://www.44tips.com:80/svc/Grid.asmx/GetUserSetThumbUrls
http://www.44tips.com:80/svc/Grid.asmx/IsCollectionTitleUnique
http://www.44tips.com:80/svc/Grid.asmx/InsertCollection
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72665/Sem_Schilt
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72662/Mirko_Cro_Cop_vs_Semmy_Schilt__Video_Game_
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72661/Josh_Barnett_vs_Semmy_Schilt_II_Part_1
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72660/_Part_2__Fedor_vs_Semmy_Schilt__PRIDE_21____23_06_2002_
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72659/Fedor_Emelianenko_vs_Semmy_Schilt___Part_4_4_

svc/Grid.asmx/{param}
c/k1collection/Sem_Schilt/{param1}/{param2}
{param1}/{param2}/{param3}/{param4}/{param5}
Four: examine response codes

404 analysis

Examine if “folder” access returns 404

everything beyond the “folder” is a parameter
Thank You!
Ofer Shezaf, ofr@hp.com