

Security Testing For RESTful Applications

Ofer Shezaf, HP Enterprise Security Products
ofr@hp.com



About Myself

**I create
security
products**

Currently, chief architect for Risk and Vulnerability Management at HP.

Security research and Product Management at Breach Security, a Web Application Firewalls vendor.

Electric Cars security...

**I am an
application
security
veteran**

Director, Web Application Security Consortium.

Founded the OWASP Israeli chapter.

ModSecurity Core Rule Set Project, WASC Web Hacking Incident Database.

**I really try to
learn what
information
security is**

Read my blog at <http://www.xiom.com>

Be ready to some philosophy of science and cognitive psychology

**I live in
Kibbutz
Yiftah, Israel**



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

Research

Test

Remediate



About RESTful Web Services

About RESTful Web Services

RESTful WS in the Wild

Security of RESTful WS

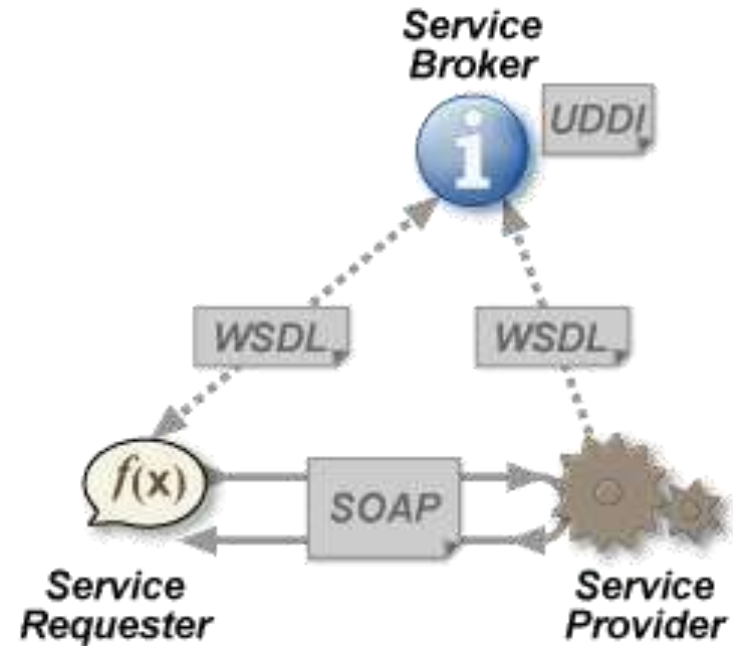
Pen-testing RESTful WS

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

```
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/20
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.exempl
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```



GET /InStock/GetStockPrice?StockName=HP

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

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 (optional)

- 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

Common RESTful WS Practices

GET /InStock/**HP**

Use of HTTP methods to indicate action

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

Embedding parameters in the request

- As part of the URL
- In headers
- Serialized as JSON in a parameter value of request body

Structured output

- Using JSON or XML for information serialization

Custom authentication and session management

- Use of the security token concept
- Often use headers



RESTful WS Example

More

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



```
{"earthquakes":  
[  
  {"eqid":"c0001xgp","magnitude":8.8,"lng":142.369,"src":"us","datetime":"2011-03-11 04:46:23","depth":24.4,"lat":38.322},  
  {"eqid":"2007hear","magnitude":8.4,"lng":101.3815,"src":"us","datetime":"2007-09-12 09:10:26","depth":30,"lat":-4.5172},  
  {"eqid":"2007aqbk","magnitude":8,"lng":156.9567,"src":"us","datetime":"2007-04-01 18:39:56","depth":10,"lat":-8.4528},  
  {"eqid":"2007hec6","magnitude":7.8,"lng":100.9638,"src":"us","datetime":"2007-09-12 21:49:01","depth":10,"lat":-2.5265},  
  {"eqid":"a00043nx","magnitude":7.7,"lng":100.1139,"src":"us","datetime":"2010-10-25 12:42:22","depth":20.6,"lat":-3.4841},  
  ...  
]
```

More Examples

It often doesn't look like your typical Web (1 or 2) application

```
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_stamp
x-amz-copy-source-if-modified-since: time_stamp
<request metadata>
Authorization: signatureValue
Date: date
```

Parameters in Headers

None Standard
Parameters/Method

None Standard Authentication and
Authorization

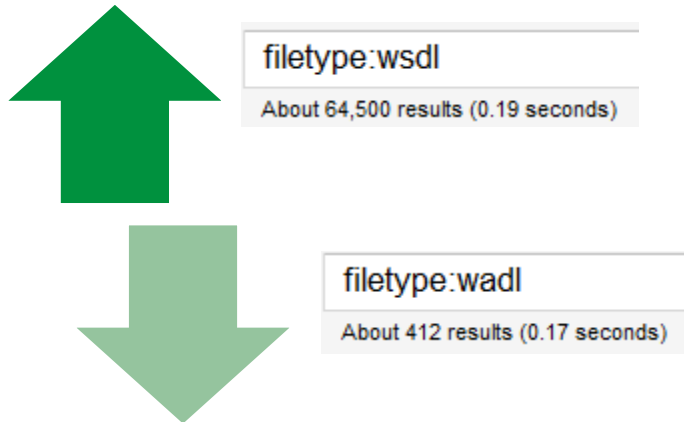
```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue

<AccessControlPolicy>
  <Owner>
    <ID>ID</ID>
    <DisplayName>EmailAddress</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org
        <ID>ID</ID>
        <DisplayName>EmailAddress</DisplayN
      </Grantee>
      <Permission>Permission</Permission>
```



RESTful services Documentation

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



```
<resources base="http://api.search.yahoo.com/NewsSearchServi
  <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="al
          <option value="all"/>
          <option value="any"/>
          <option value="phrase"/>
        </param>
        <param name="results" style="query" type="xs
        <param name="start" style="query" type="xsd:
        <param name="sort" style="query" default="ra
          <option value="rank"/>
          <option value="date"/>
        </param>
        <param name="language" style="query" type="x
      </request>
```

RESTful Web Services 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!

Google trends

RESTful services, SOAP services

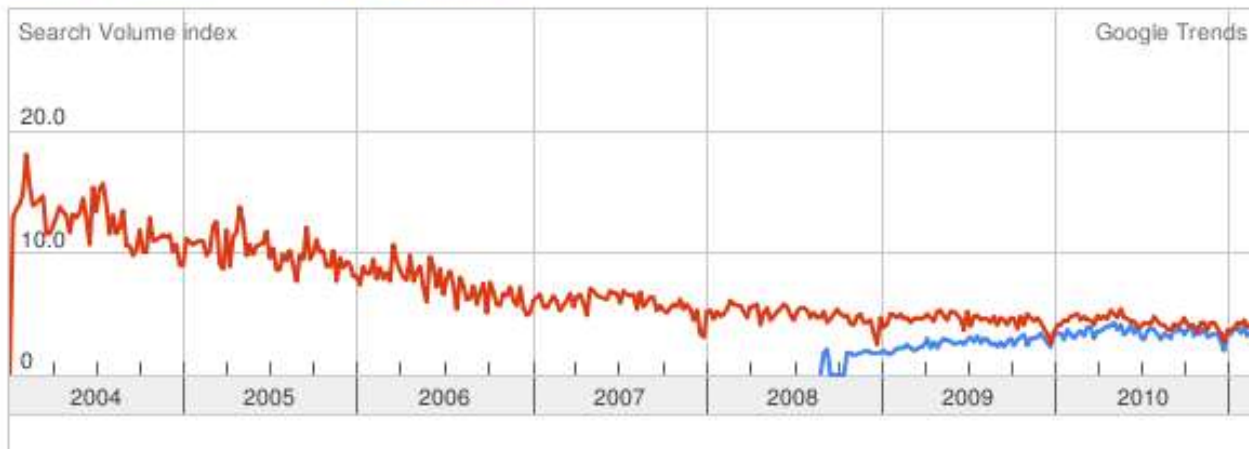
Search Trends

Tip: Use commas to compare multiple search terms.

Searches [Websites](#)

Scale is based on the average worldwide traffic of [restful services](#) in all years. [Learn more](#)

[restful services](#) 1.00 [soap services](#) 7.00



No news articles were found.



Who Uses REST?



RESTful Web Services Security

About RESTful Web Services

RESTful WS in the Wild

Security of RESTful WS

Pen-testing RESTful WS

Automated security testing of RESTful WS



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 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



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.

Fuzzing standard parameters not sufficient anymore

- Uses none standard parameters.
- Serialized inputs as JSON or XML

Guidelines for fuzzing are missing

- Determining initial values for fuzzing is hard for unused features
- Potentially large number of parameters is inhibitive in terms of time and requires selection

Custom authentication and session management breaks common cookie sharing practices.



Use Documentation

Determine:

Available services

Use of HTTP methods

Use of parameters

Potential Sources:

WADL

Programming guides

Configuration information

Application source

Programming Guides

GET /admin/user/{user}/role

Get all roles assigned for a user.

```
GET /rest/admin/user/{user}/role
```

- Parameters:

Name	Type	Description
user	userByLogin	Login name of a user.

Apache Mod_rewrite configuration

```
<Directory /var/www/example.com>  
  RewriteEngine on  
  RewriteBase /  
  RewriteCond %{REQUEST_FILENAME} !-f  
  RewriteCond %{REQUEST_FILENAME} !-d  
  RewriteRule ^(.*)$ index.php?q=$1 [L,QSA]  
</Directory>
```



Additional Documentation Examples

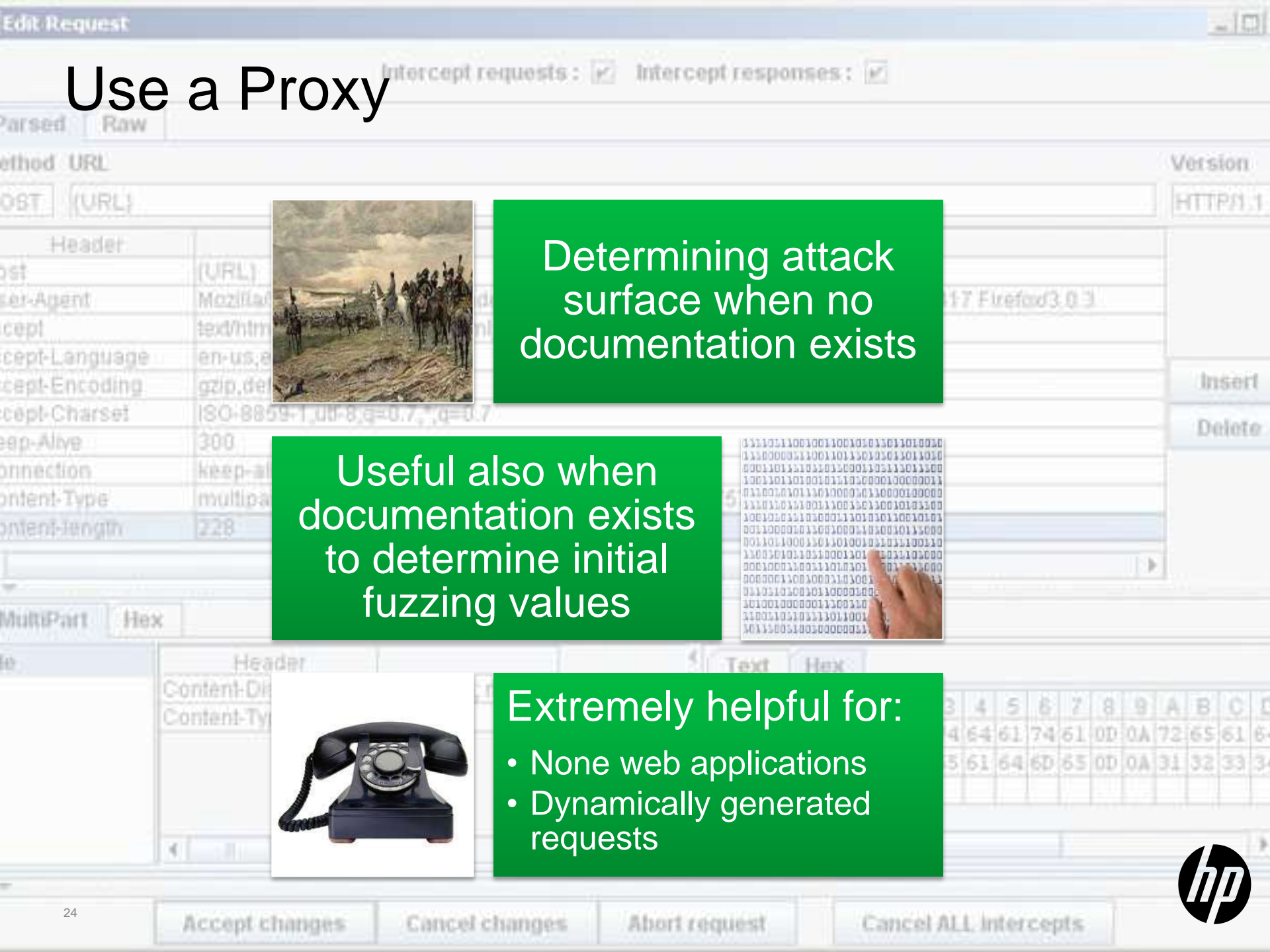
WADL

```
<resources base="http://api.search.yahoo.com/NewsS
  <resource path="newsSearch">
    <method name="GET" id="search">
      <request>
        <param name="appid" type="xsd:stri
          style="query" required="true"/>
        <param name="query" type="xsd:stri
          style="query" required="true"/>
        <param name="type" style="query" c
          <option value="all"/>
          <option value="any"/>
          <option value="phrase"/>
        </param>
        <param name="results" style="query
        <param name="start" style="query"
        <param name="sort" style="query" c
          <option value="rank"/>
          <option value="date"/>
        </param>
        <param name="language" style="quer
      </request>
```

WCF

```
[ServiceContract]
public interface IMSDNMagazineService
{
    [OperationContract]
    [WebGet(UriTemplate="/")]
    IssuesCollection GetAllIssues();
    [OperationContract]
    [WebGet(UriTemplate =("/{year}"))]
    IssuesData GetIssuesByYear(string year);
    [OperationContract]
    [WebGet(UriTemplate =("/{year}/{issue}"))]
    Articles GetIssue(string year, string issue);
    [OperationContract]
    [WebGet(UriTemplate =("/{year}/{issue}/{article}"))]
    Article GetArticle(string year, string issue, string article);
    [OperationContract]
    [WebInvoke(UriTemplate =("/{year}/{issue}", Method="POST")]
    Article AddArticle(string year, string issue, Article article);
}
```





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



Determining Parameters



Look for none standard headers



Determine if URL segments have a pattern

- Numerical values
- Well known templates



Look for structures in parameter values

- JSON, XML, YAML or other



Look for irregular 404 responses

- Including site specific “file not found” messages.



Brute force

- Change methods
- Attack any URL segment

```
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_s
x-amz-copy-source-if-modified-since: time_sta
<request metadata>
Authorization: signatureValue
Date: date
```



Automated RESTful Pen Testing

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 works?

Crawling

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

Attacking

- Parameter based:
 - Sending known attack vectors
 - Fuzzing parameters
- Comparing behavior for different users or before and after login

Pre-requisites

- 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 attacking time while still addressing all potential parameters.

Getting initial values for fuzzing

Custom authentication and session management breaks common cookie sharing practices.



Defining Rules

- Define parameter structure for URLs.
- Enable tool to use documentation & proxy discovered attack surface.
- Automated discovery of rules:
 - Automated import of documentation sources.
 - Applying parameter determination techniques.

The image shows two screenshots from a web application security tool. The top screenshot displays the 'Resource Parameters' configuration for a request. The 'Resource Path' is set to `/search.{format}`. The 'Resource Parameters' table is as follows:

Name	Default value	Style
q	soapui	QUERY
format	json	TEMPLATE

The 'Required' checkbox is checked, and the 'Type' is set to `{http://www.w3.org/2001/XMLSchema:string}`. The bottom screenshot shows the 'URL Rewrite Editor' with the following rules:

Rule (Regular Expression)	Replace	Options
Hostname : <code>http://testphp.vulnweb.com</code>	1 global rules	
<code>/artist.php/subsection/(\d+)/details/(\d+)</code>	<code>/artist.php?subsection=\$1&details=\$2</code>	L,NC

Thank You!

Ofer Shezaf, ofr@hp.com

