Security Testing For RESTful Applications

Ofer Shezaf, HP Enterprise Security Products

ofr@hp.com

About the speaker

<u>What I do for a living?</u>

- Product Manager, Security Solutions, HP ArcSight
- Led security research and product management at Breach Security & HP Fortify

<u>I am passionate about security after hours as well:</u>

- 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

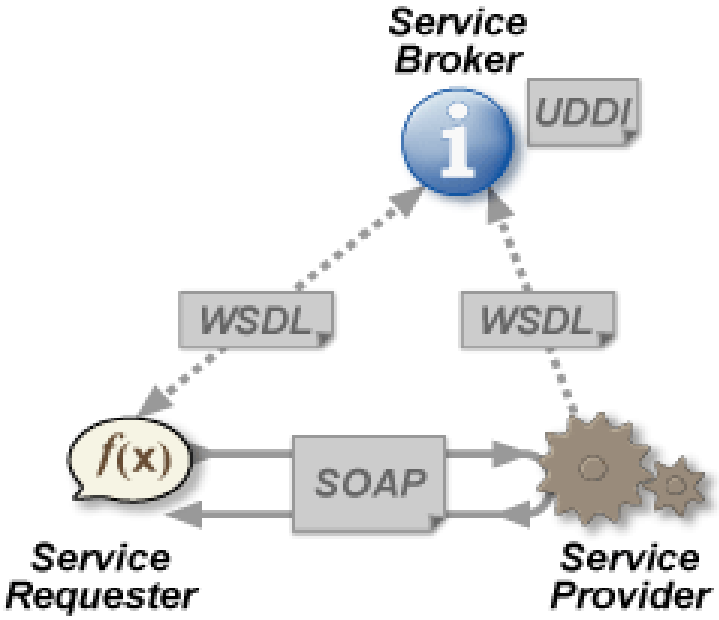
- About RESTful Web Services
- RESTful WS in the Wild
- Security of RESTful WS
- Pen-testing RESTful WS
- Automated security testing of RESTful WS

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?

```
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.exampl
      <m:StockName>IBM</m:StockName>
    </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 for an alternative view

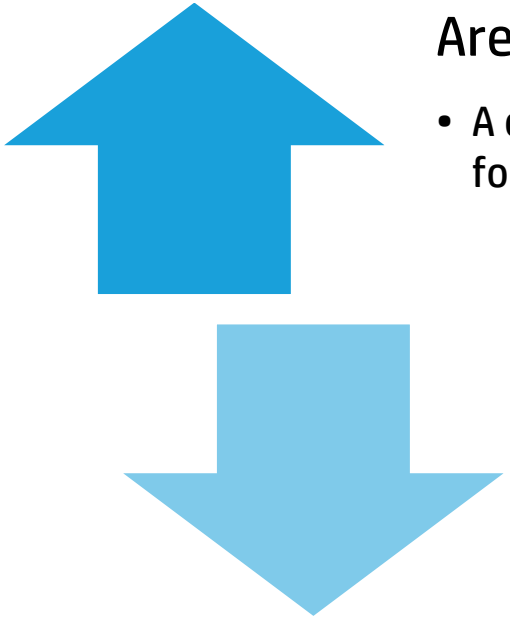
The REST design pattern

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

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

<AccessControlPolicy>
  <Owner>
```

```
PUT /destinationObject HTTP/1.1
Host: destinationBucket.s3.amazonaws.com
x-amz-copy-source: /source_bucket/sourceObj
x-amz-metadata-directive: metadata_directi
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
```


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


```
{"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},
        ...
```


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.

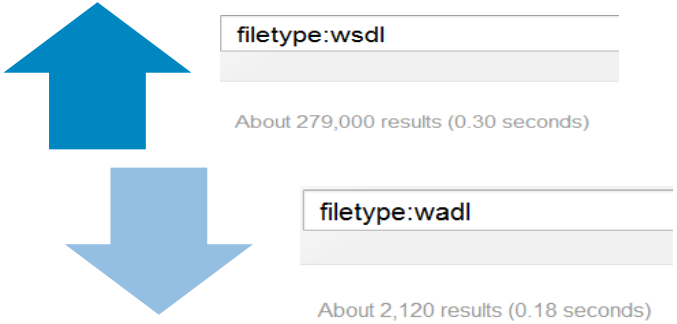
```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: signatureValue

<AccessControlPolicy>
  <Owner>
```

```
PUT /destinationObject HTTP/1.1
Host: destinationBucket.s3.amazonaws.com
x-amz-copy-source: /source_bucket/sourceOb
x-amz-metadata-directive: metadata_directi
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




```
<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 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!

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

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

Challenges

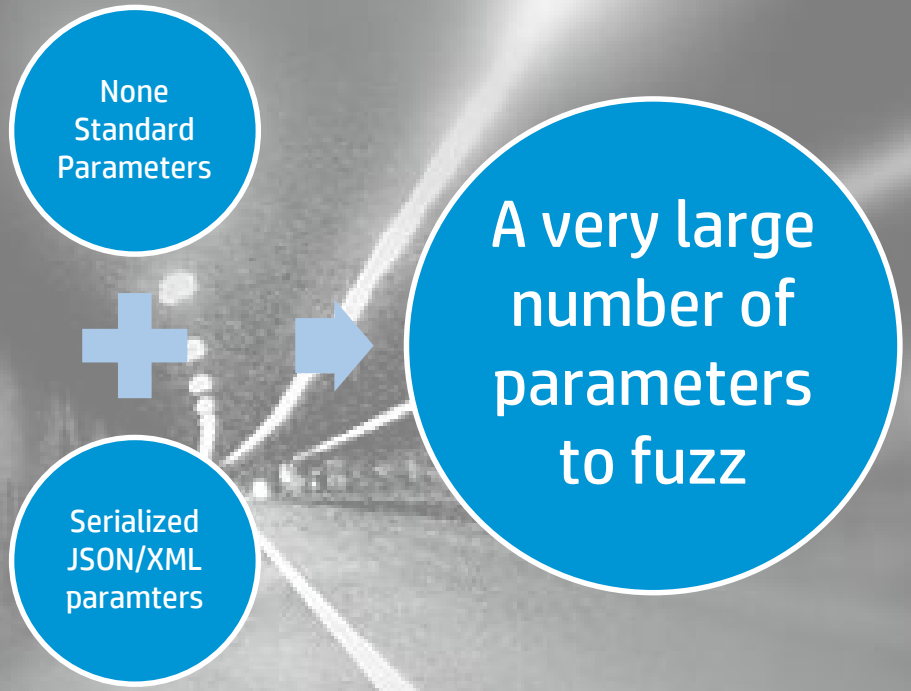
Detecting Attack Surface

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.

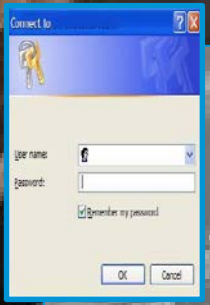
Challenges

Mega fuzzing

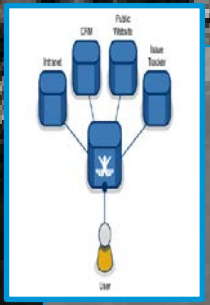


Challenges

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
- Programing guides
- Configuration information
- Application source

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.

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

Solutions

Use Documentation

Determine:

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

Potential Sources:

- WADL
- Programing guides
- Configuration information
- Application source

```
<resources base="http://api.search.yahoo.com/News
    <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"
                    <option value="all"/>
                    <option value="any"/>
                    <option value="phrase"/>
```

```
[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);
```

Solutions

Use a proxy

Determining attack surface when no documentation exists

Extremely helpful for:

- None web applications
- Dynamically generated requests

Useful also when documentation exists to determine initial fuzzing values

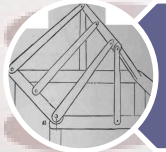
Solutions

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?

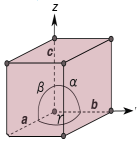
Crawling

- 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

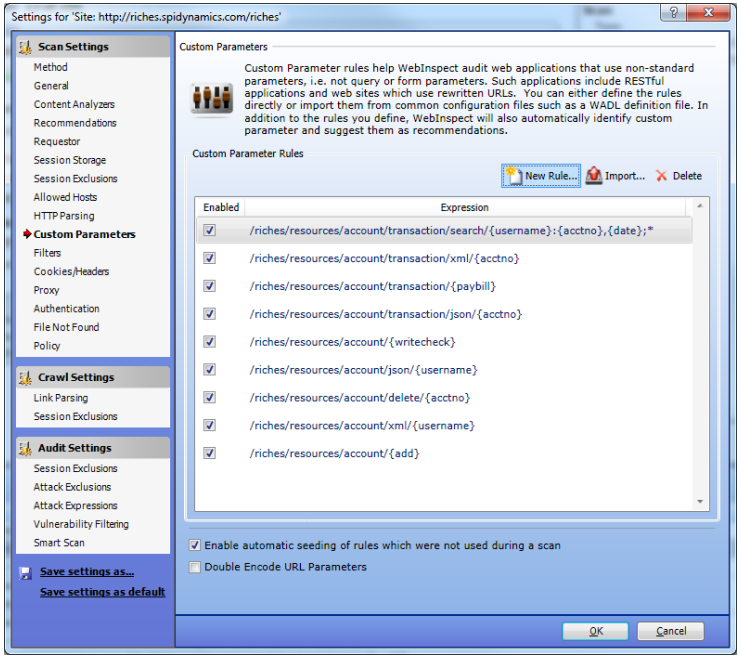
Use rules when crawling and attacking

Rule can be:

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

Or…

Get smart!

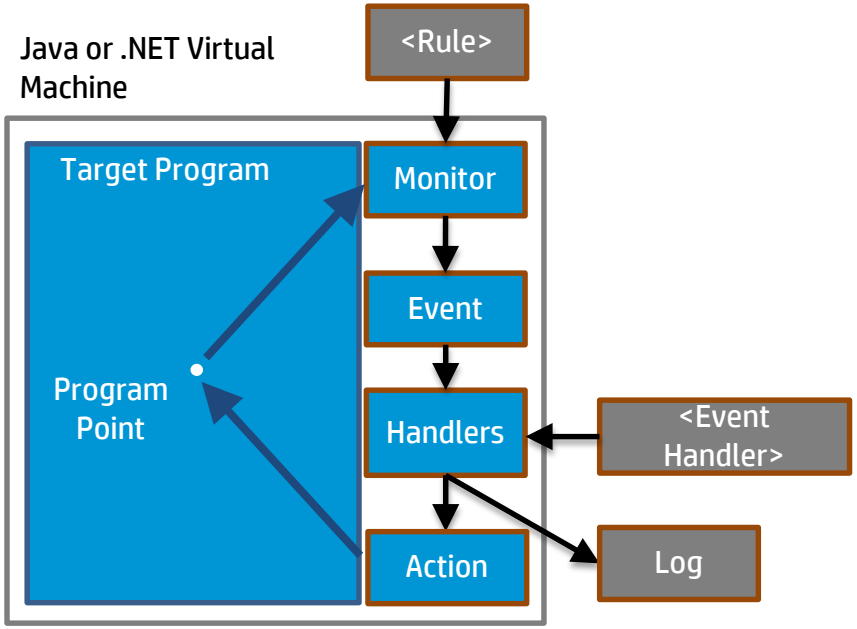


WebInspect 9.2 REST rule editor

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



Fortify Run Time Architecture

Three: Look for highly varying URL segments

```
http://www.44tips.com:80/svc/Grid.asmx/GetContentItems
http://www.44tips.com:80/js/templates/new/controlPanelSearch.htm?v=2
http://www.44tips.com:80/svc/Grid.asmx/GetRelatedListItems
http://www.44tips.com:80/svc/Grid.asmx/GetContentItems
http://www.44tips.com:80/js/templates/publishSetDialog.htm
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/svc/Grid.asmx/GetContentItems
http://www.44tips.com:80/svc/grid.asmx/GetUserSetThumbUrls
http://www.44tips.com:80/svc/Grid.asmx/GetRelatedListItems
http://www.44tips.com:80/svc/Grid.asmx/GetRelatedListItems
http://www.44tips.com:80/svc/grid.asmx/GetUserCollectionInfo
http://www.44tips.com:80/c/k1collection/Sem_Schilt/i72665/Sem_Schilt
http://www.44tips.com:80/c/k1collection/Sem_Schi  i72662/Mirko_Cro_Cop
_vs__Semmy_Schilt__Video_Game_
http://www.44tips.com:80/c/k1collection/Sem_Scn  1/Josh_Barnett_
vs__Semmy_Schilt_II_Part_1
http://www.44tips.com:80/c/k1collection/Sem_Sch.  2660/_Part_2__Fedo
r_vs_Semmy_Schilt__PRIDE_21____23_06_2002_
http://www.44tips.com:80/c/k1collection/Sem_Schi  659/Fedor_Emelian
enko_vs_Semmy_Schilt__Part_4_4_
…
```

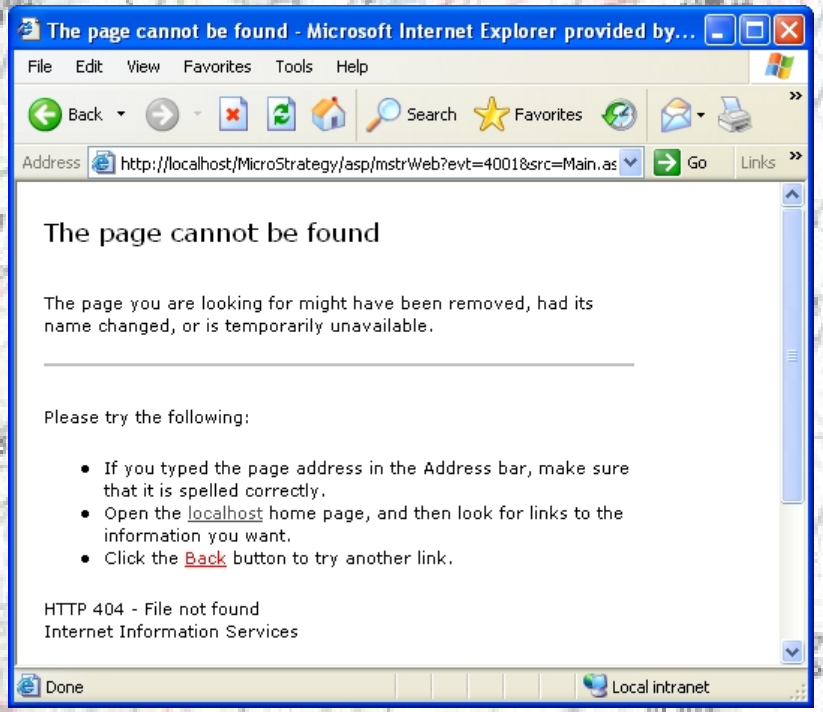
svc/Grid.asmx/{param}
c/k1collection/Sem_Schilt/{param1}/{param2}
~~{param1}/{param2}/{param3}/{param4}/{param5}~~

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