HTTP Parameter Pollution

Chrysostomos Daniel

Introduction

Nowadays, many components from web applications are commonly run on the user's computer (such as Javascript), and not just on the application's provider server (such as Servlets). As time goes by, there is the need for web applications to provide a multitude of services to their users while at the same time being consistent with functionality, interactivity and ease of use. For this reason, even the simplest web application may possibly obtain and process a plethora of different HTTP parameters. This could result in the exposure of an extensive variety of input validation or injection vulnerabilities, such as Cross-site Scripting, SQL Injection and Command Injection, waiting to be manipulated and exploited.

These web vulnerabilities are now ordinary and there has been a lot of research around them which has helped the web application field to be more secure. Nevertheless, a vulnerability that has been around for a long time has only now begun to raise alertness in the web security world - HTTP Parameter Pollution (HPP). This vulnerability was first presented by Stefano di Paola and Luca Carettoni in 2009 at the OWASP Poland conference. The fact that this vulnerability has been around for a long time means that numerous vulnerabilities affecting or targeting real-world applications have been discovered.

HTTP Parameter Pollution (HPP) in detail

HTTP Parameter Pollution, as implied by the name, pollutes the HTTP parameters of a web application in order to perform or achieve a specific malicious task/attack different from the intended behavior of the web application.

This hacking technique is considered to be simple, but quite effective. Furthermore, the main reason this attack can be realized is because the input is not get sanitized properly. HPP injects encoded query string delimiters in existing or other HTTP parameters (i.e. GET/POST/Cookie), which make it feasible to supersede parameter values that already exist to inject a new parameter or exploit variables from direct access. This attack affects all web technologies, whether running client-side or server-side.

Generally, an attacker can use HPP vulnerabilities to:

- Supersede existing hardcoded HTTP parameters.
- Alter or modify the intended/normal application behavior.
- Access and potentially exploit variables that are not been controlled properly.
- Bypass WAFs rules or input validation mechanisms.

Thus, if a web application is vulnerable to HPP attacks, the security of the web application is compromised, giving an attacker an easy way to perform malicious or illegal activities.

Web Technologies

HTTP allows the submission of the same parameter more than once. The manipulation of the value of each parameter depends on how each web technology is parsing these parameters. So, what happens if the same parameter is provided more than one time?

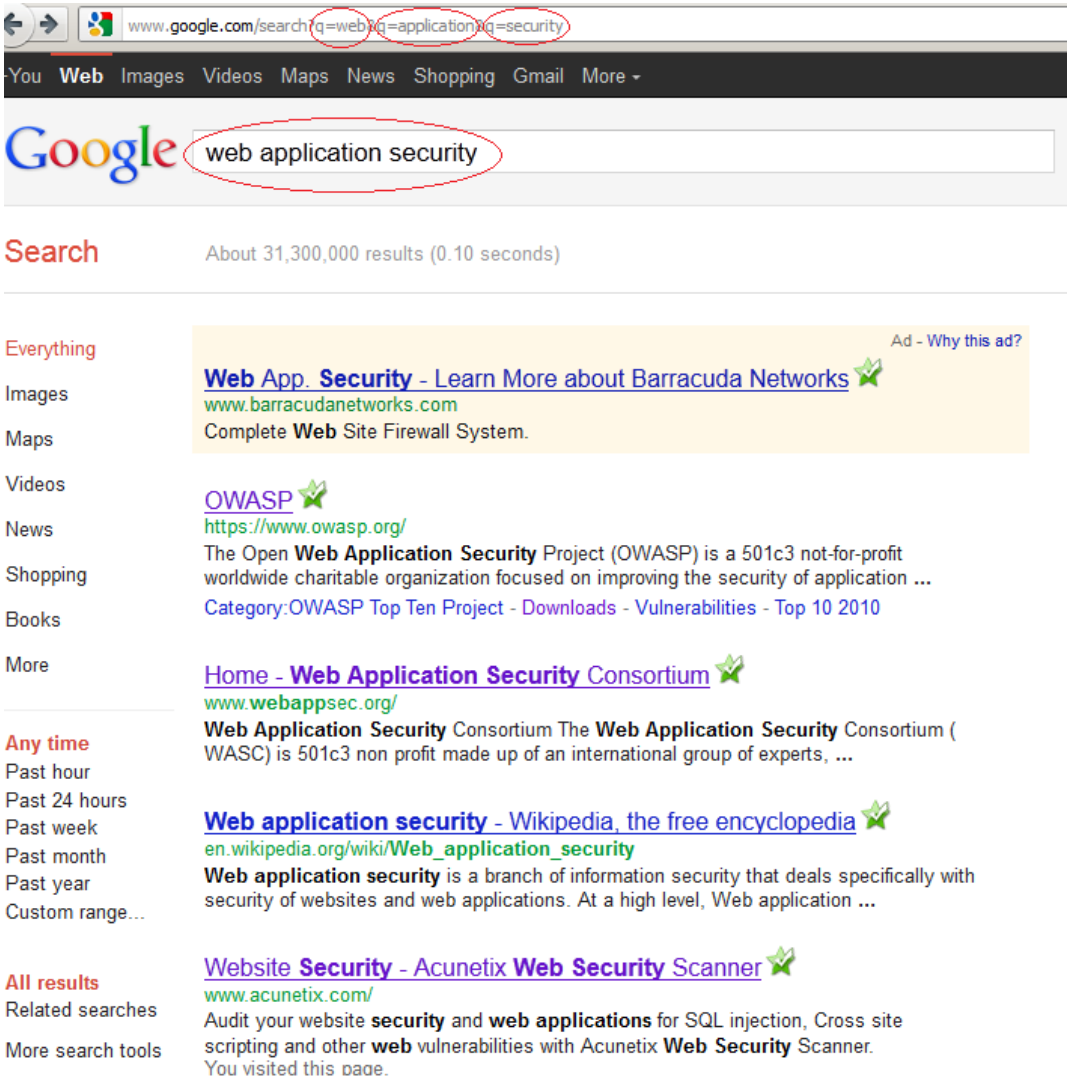
Some web technologies parse the first or the last occurrence of the parameter, some concatenate all the inputs and others will create an array of parameters. Below is a table showing how each web technology is parsing different values of the same parameters at the server-side.

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

[Reference: HTTP Parameter Pollution OWASP EU09 Poland presentation;
https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf]

The following examples show how the web technology of a web application is triggering or parsing same parameters in one query. The first example on how parameters are triggered can be shown below using Google search engine. In Google you can have the following query;

http://www.google.com/search?q=web&q=application&q=security

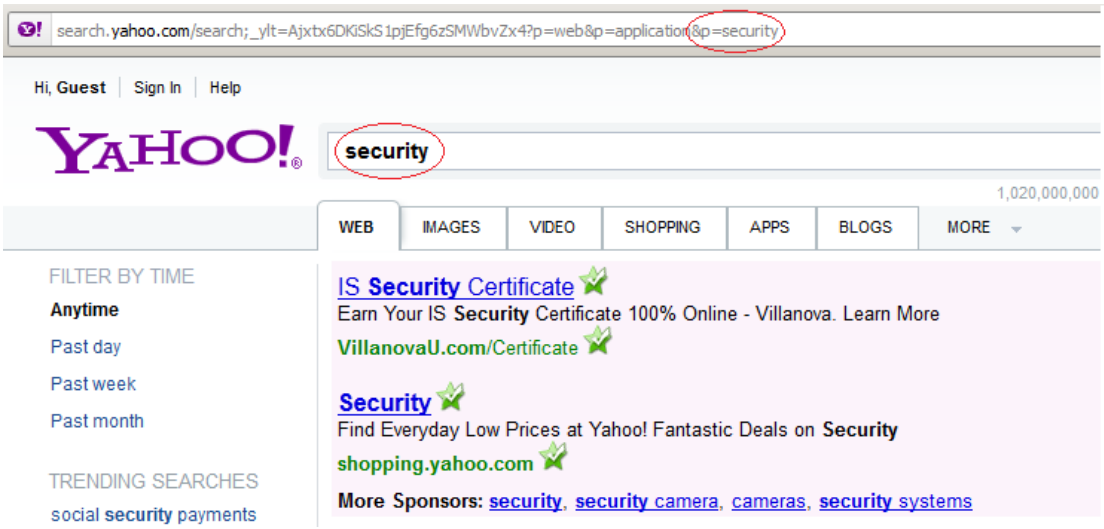


As shown in the above screenshot, the same parameter 'q' is being used three times. In this case, Google concatenates the three values with a space in-between, thus the end result will be 'web application security'.

A second example is with the search engine Yahoo!. The following query has been used:

http://search.yahoo.com/search;_ylt=Ajxtx6DKiSkS1pjEfg6zSMWbvZx4?p=web&p=application&p=security

Having the same three parameters as with the previous example, it is shown that Yahoo! is only parsing the last parameter, thus the end result will be 'security'.



This shows clearly how each technology is differently parsing the value parameters. The way each technology is parsing the parameters is not wrong, as long as the developer is aware of it. If the developer is not aware of this behavior or parameter triggering, then this can be dangerous for the web application. In addition, web technologies/languages have several secure functions that allow them to protect themselves by being able to control and manipulate these kinds of input parameters.

Client-side and Server-Side

HTTP Parameter Pollution can be classified in two categories - client-side or server-side. Each technology is parsing parameters differently, thus different attacks can be realized. This, depending on the way it is being triggered, enables client-side or server-side attacks. Moreover, in each case the parameters are manipulated accordingly to perform hacking activities at the front-end (client) or the back-end (server) of the web application.

Client-side HTTP Parameter Pollution vulnerability

The HTTP Parameter Pollution (HPP) Client-side attack has to do with the client or user environment, meaning that the user's actions (i.e. access a link in a browser) are affected and will trigger a malicious or unintended action without the user's knowledge. HPP Client-side attacks can be reflected HPP (such as an injection of additional parameters to URL links and/or other src attributes), stored HPP (which can be functional on all tags with data, src, and href attributes) and action forms with POST method. Another HPP client-side attack is the DOM-based attack which has to do mostly with parsing *unexpected* parameters and the realization of client-side HPP using JavaScript.

Obviously, the ability or capacity of the injection depends on the attributes of the link and its functionalities. Nevertheless, the main aim is to generate HPP attacks on the client side.

An example of a typical HPP client-side attack includes a website that is vulnerable to HPP and a group of victims that will interact with the vulnerable website. An attacker, after identifying a vulnerable website, will create a vulnerable link with its HTTP parameters polluted and will send this link or make it publicly available through emails or social networks for naive and unsuspecting victims to click on. After the victims have clicked on it, the intended malicious behavior will be performed, affecting the users and the web application (application providers).

The following scenario is a webmail service website from where a user can view and delete his/her emails. The URL of the webmail website is:

http://host/viewemail.jsp?client_id=79643215

The link to view an email is

* View *

The link to delete an email is:

* Delete *

When the user clicks on either of the above links, the appropriate *action* will be performed. The two links are built from the URL. The ID will be requested and will be embedded/added in the href link together with the according action. Thus:

ID = Request.getParameter("client_id")
href_link = "viewemail.jsp?client_id=" + ID + "&action=abc"

This web application, and more precisely the client_id, is vulnerable to HPP. As seen below, an attacker creates a URL and injects another parameter 'action' preceded by an encoded query string delimiter (e.g. %26) after the client_id parameter. This parameter holds the value 'delete':

http://host/viewemailn.jsp?client_id=79643215%26action%3Ddelete

After the creation of the malicious link, the page now contains two links which are injected with an extra action parameter. Thus:

 View
 Delete

As shown in the table above, JSP will parse the two same parameters (action) and will return the first value. The JSP query Request.getParameter("action") will return 'delete' in both cases. Thus, the user will click either of the two links, View or Delete, but the action Delete will always be performed.

This is a simple example how an attacker can exploit an HTTP Parameter Pollution vulnerable website and cause malicious code to run or be executed without being detected.

Server-side HTTP Parameter Pollution vulnerability

In the HPP Server-side the back-end environment of the web application will be affected. The attacker using HPP attacks will try to exploit the logic of the vulnerable web application by sending a triggered, or polluted URL, for example to access the database of a web application.

HPP Server-side can be also used to bypass several web application firewalls (WAFs) rules. Some WAFs only validate a single parameter occurrence, such as the first or the last one. In a case where the web technology concatenates the value of multiple parameters which are the same, such as ASP.NET/IIS, then an attacker can split the malicious code into those occurrences thus bypassing the security mechanism or rules of the web application firewall.

Moreover, URL rewriting can occur using HPP. For instance, an attacker can inject an encoded query string in order to cause the URL to be rewritten. An example can be seen below:

Encoded string:
http://host/xyz%26page%3dedit

Rewritten URL:
http://host/page.php?page=view&page=xyz&action=edit&id=0

As mentioned before, the capability of the injection depends on the attributes of the link and its exposed functionalities.

HPP Server-side attacks can also be used for cross-channel pollution and to bypass CSRF tokens.

In order to better understand the server-side HPP attack, the following example will try to explain how this attack can bypass web application firewall rules or signature-based filters using concatenation of parameters with the same values. The following URL/request is send to the server:

http://testaspnet.vulnweb.com/test/vuln.cgi?par1=val1&par2=val2

The web server will parse the above query and will split it into pairs (name/value) in order to be manipulated or used by the web application. Thus, the web application will take par1 and par2 with values val1 and val2 respectively. If the web application is vulnerable to HPP attacks, an attacker could exploit it and submit a malicious payload. Take the following case:

http://testaspnet.vulnweb.com/test/vuln.cgi?par1=val1&par1=val2

You can see that there are two par1 parameters, each holding two different values. In this case how is the application going to trigger this? It depends on the web technology, as seen in the Web Technologies section above. Because of the different handling methods of parameters, hackers can control them in order to avoid security mechanisms and attack the web application.

In another example, where the web technology is ASP.NET/IIS, a hacker can send the following request to the server:

http://testaspnet.vulnweb.com/test/vuln.cgi?par1=<script&par1=prompt."..."> …

Since ASP.NET/IIS concatenates the values of the same parameters, the end result will be <script prompt"…">. Consequently, an attacker can expand this into a complete cross-site scripting attack.

If there is an installed Web Application Firewall in front of this application then it will check each occurrence of the parameter separately against the rules for injection attacks. As a result, the web application firewall will check the first parameter par1=<script, which will not match any of the injection attack rules since this is not a malicious payload. Then it will make the same check for the second parameter which equals par2=src="…". Again, this is not considered as a dangerous payload and will not raise any alerts. Nevertheless, as mentioned before, ASP.NET/IIS will concatenate these values, based on how the technology parses these occurrences, resulting in executing an XSS attack (if it was expanded in a complete XSS payload).

This is an example how an attacker can bypass some web application firewalls rules using HPP, enabling further attacks.

Countermeasures / Prevention

In order to prevent these kinds of vulnerabilities, an extensive and proper input validation should be performed. There are safe methods to conform to with each web technology/language. Moreover, awareness about the fact that clients/users can provide more than one parameter should be raised.

Conclusion

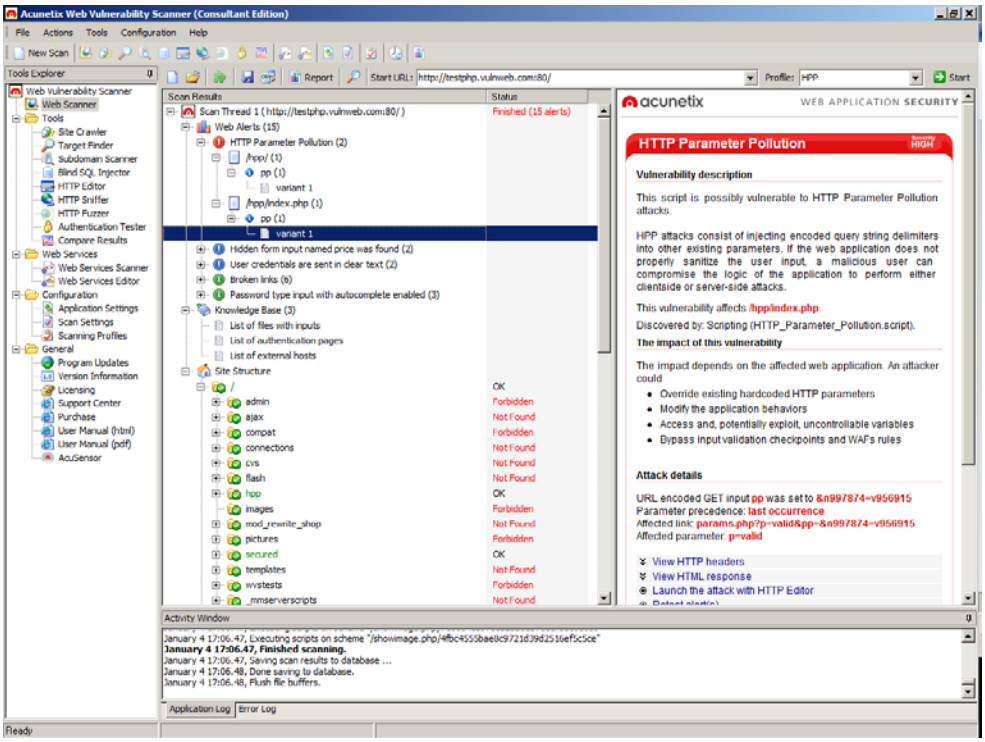
An injection attack that has been around for some years but never raised any alertness and didn't particularly intrigue the security world, has come to enlighten the web security industry to the fact that these kind of injection attacks should never be underestimated, and that the lack of standardization of the different parsing methods each web technology encompasses, together with complexity, might lead to vulnerabilities. HTTP Parameter Pollution takes advantage of the fact that HTTP allows more than one of the same parameters to be used, which causes some web applications, based on their web technology and how these trigger HTTP parameters, to be exposed and to be exploited by malicious users. HPP is a simple yet quite effective hacking technique which affects both client-side and server-side environments. When exploited, the impact of an HPP vulnerability depends on the functionality of the web application.

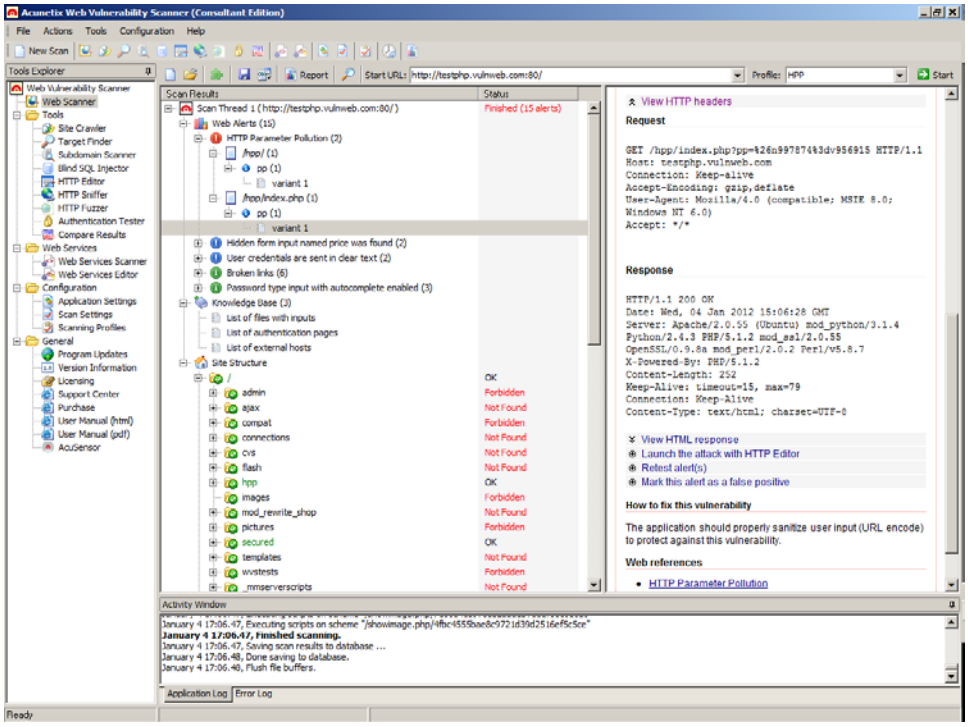
It has been made clear that this vulnerability, despite its simplicity, can be very dangerous and can compromise your website's security systems. Proper checks should be performed in order to determine if your site is vulnerable to HPP attacks in order to limit the possibilities and decrease the opportunities that hackers can exploit that could lead to a breach of the Confidentiality, Integrity and Availability of your site.

Scanning for HTTP Parameter Pollution with Acunetix Web Vulnerability Scanner!

Acunetix Web Vulnerability Scanner Version 8 scans any website or web application for HTTP Parameter Pollution vulnerabilities, reveals the relevant information for the user, such as the vulnerability location and suggests remediation techniques. Scanning for HPP is normally a quick process (depending on the size of the web-site).

The images below demonstrate how Acunetix Web Vulnerability Scanner crawls, scans and detects a site which is vulnerable to HTTP Parameter Pollution.





Acunetix Web Vulnerability Scanner Version 8 scans against HPP vulnerabilities, reporting the location of the vulnerability, the HTTP headers and HTML response with information regarding the HPP vulnerability and how this can be remediated.

About Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner ensures website security by automatically checking for SQL injection, Cross-Site Scripting and other vulnerabilities. The scanner checks password strength on authentication pages and automatically audits shopping carts, forms, dynamic content and other web applications. Detailed reports resulting from the scan identify where vulnerabilities exist. The Acunetix WVS Reporting Application allows security alerts to be presented in a document which abides by the PCI Compliance specification.

About Acunetix

Acunetix is a market leader in web application security technology, founded to combat the alarming rise in web attacks. Its flagship product, Acunetix Web Vulnerability Scanner, is the result of several years of work by a team of highly experienced security developers. Acunetix customers include the US Army, US Airforce, AT&T, KPMG, Telstra, Fujitsu, and Adidas. More information can be found here http://www.acunetix.com/.