Advances In Multidisciplinary & Scientific Research
AIMS Research Journal
A Multidisciplinary & Interdisciplinary Journal
Vol. 3 No. 4, December 2017

Implementation of Parameter Value Cache (PVC) for the protection of Web Applications against HTTP Parameter Pollution (HPP) Attacks

Adeyemo A. B. & Ayodele O. S.
Computer Science Department
University of Ibadan
Ibadan, Nigeria
sesanadeyemo@gmail.com

ABSTRACT

Manipulation of URL query string parameters is one of the numerous ways hackers exploit vulnerable web applications. HTTP Parameter Pollution (HPP) describes all forms of attack resulting from such activities. Application Request Cache (ARC) is an existing framework for protecting web applications from HPP by comparing URL Schema found in an HTTP request string with a set of benign schemas for the given web application. It rejects an HTTP request with a URL Schema not found in the set and record the event. Unfortunately ARC is not effective in cases with improper parameter sequences and polluted parameter values. Parameter Value Cache (PVC) is an extension of ARC that addresses its limitations. This study presents the implementation of a PVC security module in PHP.

Keywords: Parameter Value Cache, Application Request Cache, HTTP Parameter Pollution Attacks

---

Aims Research Journal Reference Format:
Adeyemo A. B. & Ayodele O. S. (2017): Implementation of Parameter Value Cache (PVC) for the protection of Web Applications against HTTP Parameter Pollution (HPP) Attacks. Advances in Multidisciplinary & Scientific Research Journal. Vol. 3. No.4, Pp1-12.

1. INTRODUCTION

HTTP Parameter Pollution (HPP) is a form of parameter tampering attack targeting the URL and has been around for many years which has also been under estimated (Luca and Stefano, 2009). Web application security has focused more on validating input values supplied to form fields and sanitizing output so that they're not misused when HTTP requests are processed (Scholte & Balzorottir, 2008). Making the URL tamper-proof will help to make web applications more secure since about 70 percent of security flaws listed out by Open Web Application Security Project (OWASP) can be exploited by polluting URL parameters with harmful values.

OWASP is a worldwide free and open community focused on improving the security of application software. The community is aimed at making application security visible, so that people and organizations can make informed decisions about application security risks (Meucci and Muller, 2013). OWASP keeps developers updated about new security risks and guidelines on how to mitigate them. The top ten most critical security risks according to OWASP are: Injection, Broken Authentic and Session Management, Cross-Site Scripting (XSS), Insecure Direct Object References, Security Misconfigurations, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery (CSRF), Using Components with Known Vulnerabilities and Unvalidated Redirects and Forwards (OWASP 2013). XSS and SQL injection are the most common web application vulnerabilities. These vulnerabilities can be mitigated through output sanitization and input validation (Scholte & Balzorottir, 2008).

Karim et al (2015) presents a classification of web attacks according to Web Application Security Consortium (WASC) and have grouped security tools against these attacks into two main categories; Web Application Firewalls (WAF) and Intrusion Detection Systems (IDS). WAF act as an agent (inverse proxy) to check exchanged requests between the client and web application while IDS detects abnormal or suspicious activity on a target data. The WASC classification of web attacks are: Insufficient Authentication, Insufficient Authorization, Client-Side Attacks, Command Execution, Information Leakage and Logical Attacks. The OWASP has not listed HPP as one of the top flaws that needs to be addressed. This is not because it is unimportant but because it covers a wide range of attacks. It is convenient to say that HPP is a foundation for propagating other forms of attack through the URL. Making URLs Tamper-proof does not guarantee that these flaws will cease to exist in a web application but it will not be possible to propagate them through the URL.

Various security tools exist for fighting security attacks listed by OWASP as well as those that fall under WASC classification. Many of these tools are open source (available in source code) while some are proprietary tools (Lemes, 2011). Examples of open source web application security tools include intrusion detection systems like "Snort" (http://www.snort.org), "Snare" (http://www.intersectalliance.com) and "Tripwire" (http://www.tripwire.com), vulnerability scanners "Nessus" (http://www.nessus.org) and "Saint" (http://www.saintcorporation.com) (Lawton, 2002).

1.1 HTTP Parameter Pollution (HPP)

A client sends input to the web server in the form of an HTTP request. GET and POST are the most common requests. The request encodes data created by the user in HTTP header fields including file names and parameters included in the requested URL (Nguyen-Tuong et al, 2005). HPP as the feasibility to override or add the HTTP GET/POST request by injecting query string delimiters (Carretoni and Stefano, 2009). Using HPP it may be possible to override existing hardcoded HTTP parameters, modify the application behaviors, access and, potentially exploit, uncontrollable variables, and bypass input validation checkpoints and WAF rules (Ronan, 2013).

For HTML forms that is submitted using the HTTP GET method, form parameters as well as their values appear as part of the URL that is accessed after the form is submitted. An attacker may directly edit the URL query string, embed malicious data in it, and then access this new URL to submit malicious data to the application (Livshits et al, 2006).

The first automated approach for the discovery of HTTP Parameter Pollution (HPP) vulnerabilities in web applications was presented by Balduzzi et al (2011) using a prototype implementation called PAPAS (PArameter Pollution Analysis System. The Application Request Cache (ARC) by Athamasopoulos et al (2013) is a framework for protecting web applications against HPP exploitation. ARC protects web applications against parameter manipulation attacks by comparing URL schemas (formats) in HTTP requests against a cache of benign schemas that compose the application's logic. The incoming HTTP request is rejected if the URL schema is not found and the event is logged (Athamasopoulos et al, 2013). Unfortunately this solution is only effective in cases where there is a duplication of parameters or parameters are combined in such a way that the web application does not handle them correctly.

ARC does not cover extreme cases where they are provided with malicious values such as JavaScript codes, SQL queries, large values that can lead to memory problems and so on. PVC is an extension of ARC that hosts constraints on parameter values for every URL schema stored by ARC. PVC extracts URL parameters and their values from each schema validated by ARC and checks if the value supplied to each parameter of the query string is beyond the limits specified by the constraints for that particular parameter. If the value provided does not meet the specified criteria in the cache, the HTTP request is rejected and the event logged. This study implements a PVC framework for making URLs handled by web applications more secure against HPP attacks by using a modified ARC formal threat model. PVC is implemented in PHP (the most widely used server-side programming language) but the system will work on any platform.

2. MATERIALS AND METHODS

Application Request Cache (ARC) is a framework by Athamasopoulos et al (2013) for protecting web applications against HPP exploitation. ARC hosts all benign URL schemas, which act as generators of the complete functional set of URLs that compose the application's logic. For each incoming request, ARC exports the URL, extracts the associated schema, and searches for it in the set of already known benign schemas. In case the schema is not found, the request is rejected, and the event is recorded.

2.1 ARC Formal Threat Model

A is a web application, and U_i is used for denoting any URL schema that has the following form: action?p_1=&p_2=&...&p_N=. The schema is composed of an action and a set of parameters that can take arbitrary values. URL schemas express families of HTTP requests that are served by the web application and act as descriptors of valid control flows. $U_a = \{u_1, u_2, ..., u_n\}$ is a set that contains all benign URL schemas that A can handle.

This means that for each incoming URL in U_a, a well-defined control flow f takes place, according to the application's logic. More formally:

$\forall u \in U_a \rightarrow f \in F_L$

$F_L = \{f_1, f_2, ..., f_N\}$ contains the control flows that can be handled safely by the web application. We denote as Fc the set of all possible control paths of A. Apparently, $F_L \subseteq F_c$ and $F_h = F_c - F_L$ is the set of all control flows that A can reach, but not initially programmed to execute.

Let define the set $U_{hpp} = \{v_1, v_2, ..., v_N\}$ that contains all URL schemas that can initiate a control flow $f \in F_h$. Ideally, A should reject all incoming v for which the following relationship holds:

$\forall_v \in U_{hpp} \rightarrow f \in F_h$.

It is noted that flows in F_h may have arbitrary consequences and force the web application to produce undesired results. The formal threat of ARC is the basis upon which that of PVC is derived and it formally illustrates how PVC intends to address the extreme cases that ARC cannot handle.

2.2 PVC Formal Threat Model

Parameter Value Cache (PVC) is a storage that holds a set of URL query string parameter value constraints once a web application is loaded into memory. The constraints determine benign values. These are all possible values that can be safely accepted by the parameters of a URL. To prevent HTTP Parameter Pollution (HPP) attacks in web applications, PVC extract parameters from benign URLs and compare their values against constraints in the cache. Benign URLs are determined by Application Request Cache (ARC). ARC is an existing tool that only reduce HPP vulnerabilities by comparing URLs against a cache of known schemas. If a URL conforms to the format, it is passed to PVC; else the HTTP request initiated is rejected. A formal threat model for PVC can be defined as follows:

Let A be a web application with a set of benign URL schemas $U_s = \{u_1, u_2, u_3, \ldots, u_n\}$.

This means that for each incoming URL in U_s, a well-defined control flow *f* takes place, according to the application logic. More formally:

$\forall u \in U_s \rightarrow f \in F_L$

$F_L = \{f_1, f_2, ..., f_N\}$ contains the control flows that can be handled safely by the web application.

If F_c is the set of all possible control paths of A, then $F_L \subseteq F_c$ and $F_h = F_c - F_L$ is the set of all control flows that A can reach, but not initially programmed to execute.

ARC was designed to reject such URL schemas that can initiate control flow $f \in F_h$. However, the values taken by the parameters of the so called benign schemas allowed by ARC can be manipulated in a way that can make the web application carry out undesired actions.

To address this:

If $P_s = \{p_1, p_2, p_3, \ldots, p_n\}$ is a set that contain parameters found in a valid URL schema *u* and

$C_v = \{c_1, c_2, c_3, \ldots, c_n\}$ a set that contains constraints for values each *p* may take.

For each parameter *p*, set $V_c = \{v_1, v_2, v_3, \ldots, v_n\}$ contains values that fall within the limits of C_v. Ideally, it is required that A accepts only incoming *u* for which the following relationship holds.

$v \in V_c$

The constraints are stored in PVC and are used as criteria for determining safe values. This is carried out after ARC must have checked that the URL schema is that for which a valid control flow takes place according to the application's logic.

An algorithm can be derived for PVC based on the threat model formalized above. The pseudo code for the algorithm is presented as:

```
Set A to Web Application
Set H to HTTP Request
For each H in A do
    Set U to URL initiating the request
    Extract U
    Set N to Number of parameters
    Set $P_{1}, P_{2}, P_{3,...,} P_{N}$ to query string parameters
    For counter =1 to N – 1 do
        counter++
        Extract $P_{N – counter}$
        For each $P_{N – counter}$ in U do
            Set V to parameter value
            Search Cache for Constraints
            Set $N_c$ to number of Constraints
            For counter = 1 to Nc do
                Compare V against constraint
                If V violate constraint then
                    Reject H
                    Request Geolocation
                    Log event
                Else
                    Process H
            end do
        end do
    end do
end do
```

The PVC algorithm is presented as:

```
PVC (A, H, U, V)
1  for each U in A do
2      N ← No of Parameters
3      for I ← 1 to N-1 do
4          I++
5          getParameter($P_{N-I}$)
6          V ← ParameterValue
7          for each $P_{N-I}$ in U do
8              loadCache
9              getConstraints
10             $N_c$ ← No of Constraints
11             for J ← 1 to $N_c$ - 1 do
12                 J++
13                 $C_j$ ← Constraint
14                 compare.data(V).constraint($C_j$)
15                 $V_c$ ← validValue
16                 if $V \in V_c$ then
17                     request.process(H)
18                 else
19                     !request.process(H)
20                     getLocation
21                     logEvent
22             end do
23         end do
24     end do
25 end do
```

2.3 The System Architecture
The system architecture for preventing HPP attacks is presented in figure 1.

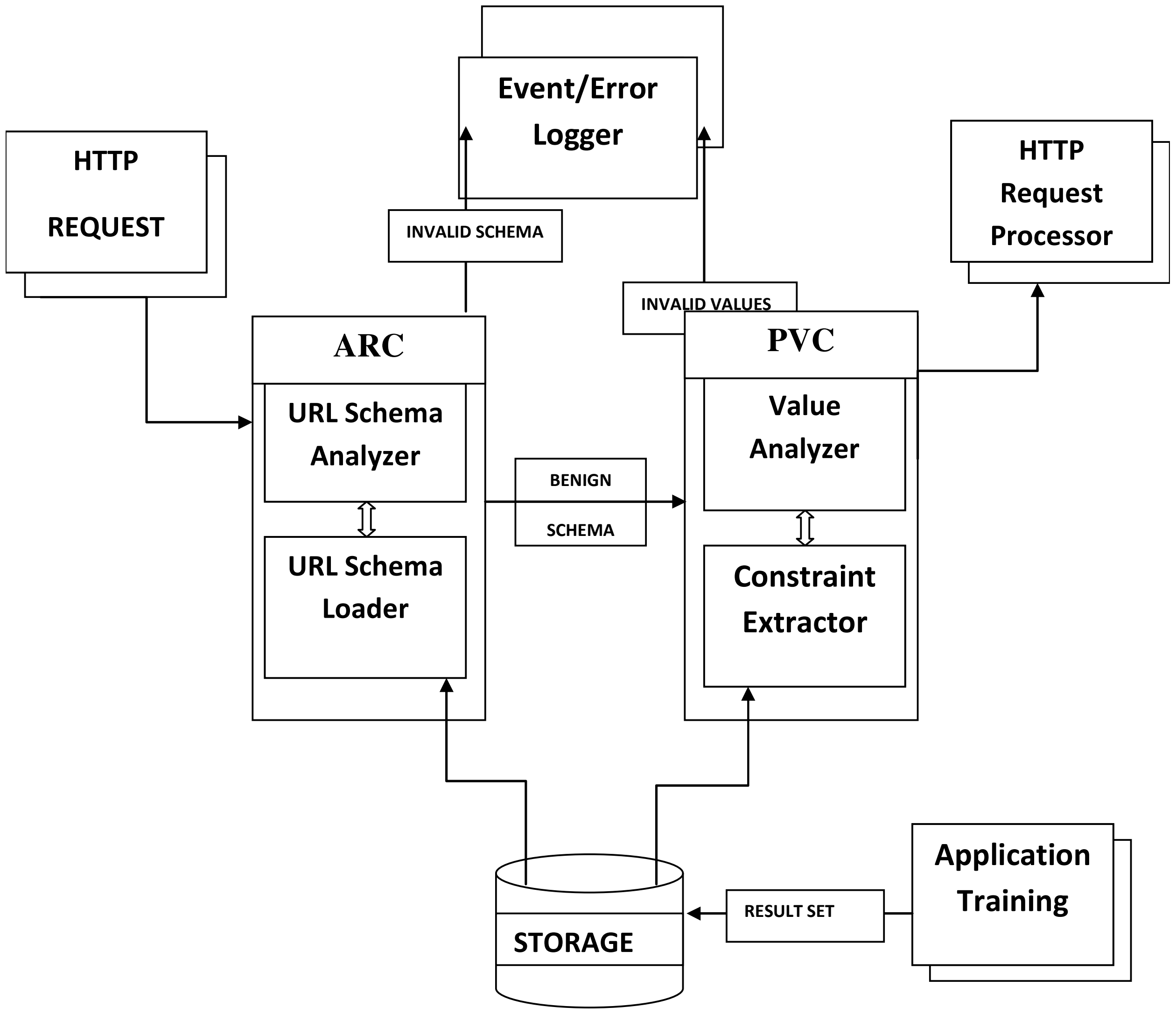


Figure 3.1: System Architecture for Preventing HPP

There are three basic phases in the system as follows:

1. The application in question is trained. Just like many anomaly detection systems, the essence of this is to extract data that enables the system, know in advance the collection of URL schemas valid for the application logic. Data (URLs) collected during training phase is stored in disk. The storage can be a file (text, JSON, XML) or database. The important thing is ability to determine which is faster for the implementation technique.
2. The second phase is handled by ARC. This phase features a URL Schema Analyzer and URL Schema Loader. The loader loads the URL schema list from storage, into the cache. The analyzer compares the list with incoming URL schemas in HTTP requests and allow only those that match any schema in the list. The rest are filtered out and dumped in the error log while rejecting the HTTP request they initiated. Benign schemas are passed to the PVC system in the nest phase.

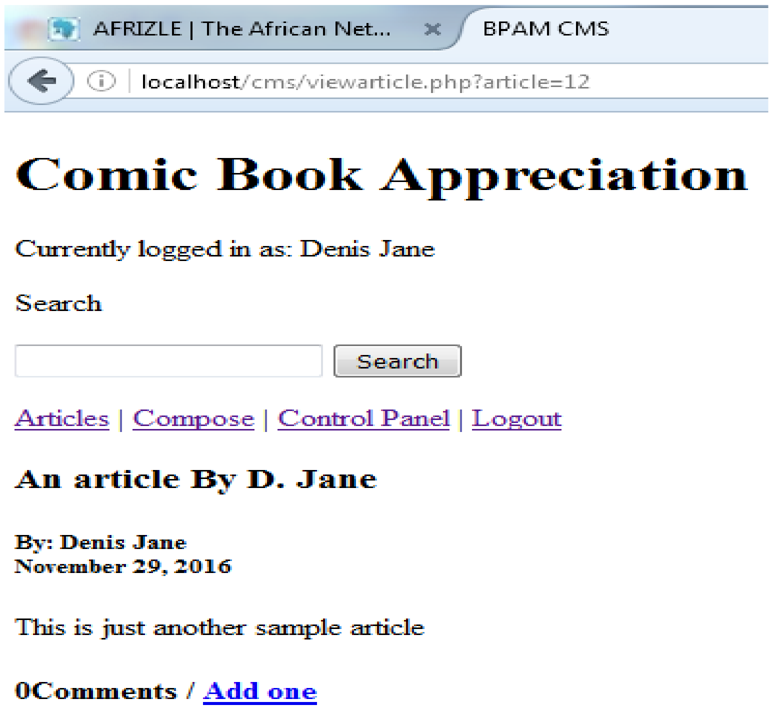
3. Finally, PVC accepts valid schemas from ARC for further checks. This phase features Parameter Value Analyzer and Constraint Extractor. The former analyze values held by parameters of the URLs and compare metadata (type, character length, etc) about them with constraints extracted from storage by the latter. If the values are within the limits set by parameter constraints, the HTTP request initiated by such URL is processed, else it is rejected and the event is logged.

2.4 Application Training and Data Structure
The PVC application has to be trained so that it can have a prior knowledge of constraints that will act as guide when determining whether a parameter value is safe or not. The training process involves running the web application in a controlled environment where only legitimate traffic is sent to the server. The data structure that is used by PVC is a hash table and a collection of linked lists.

3. RESULTS AND DISCUSSION

3.1 Software Implementation
The PVC software application was coded using Hypertext Preprocessor (PHP). The data structures (Hashes) were implemented using arrays. Data collected during application training were stored in a MySQL database. PVC was implemented as a module that resides on the web server and is imported as an “include” file into PHP scripts. The import statement is the first line of code for any script implementing it so that it can carry out its job of protecting the application against HPP before any other code is executed. Any form of HPP is rejected before execution of the major script begins. A database (stored in disk) is created before the training phase to store data collected while training the web application. The cache holds a memory table which is a temporary repository of data retrieved from the database and is maintained in memory until the application is closed. It also holds the set of URL parameters constraint needed at any point in time to check the URL parameters of any currently running scripts. Arrays (implemented as hashes in PHP) hold these parameters when the security check starts. Parameter value constraints are loaded as linked list using a class that implements linked lists in PHP.

Two types of MySQL storage engines were used. The first one is the InnoDB storage engine which was used to collect data during application training that will be stored in disk for later use. The InnoDB storage engine is designed to handle transactional applications that require crash recovery, automatically-enforced data integrity, high levels of user concurrency, and good response times. The second storage engine is the Memory storage engine and it will hold data needed while PVC is running in the RAM for fast access. The Memory storage engine is designed for non-transactional applications needing high-speed access to data that is not persistent. The fact that one application can use two storage engines at any given time makes it easy to divide data storage needs for this project into two sections. One for permanent storage of training data collected. The other, to hold parameters and their constraints (loaded from disk) only as long as the application runs. The test application used for the study is called “Comic Book Appreciation”. It is a simple application that stores articles in a database and allow visitors to read them online. Only registered users can create articles but articles can be read by anyone who uses the application. The application is vulnerable to HPP but is secured against it after integrating PVC protection. Figure 2 presents the application interface with the valid application address shown in the address bar.


AFRIZLE | The African Net...
BPAM CMS
localhost/cms/viewarticle.php?article=12

Comic Book Appreciation

Currently logged in as: Denis Jane

Search

Search

Articles | Compose | Control Panel | Logout

An article By D. Jane

By: Denis Jane
November 29, 2016

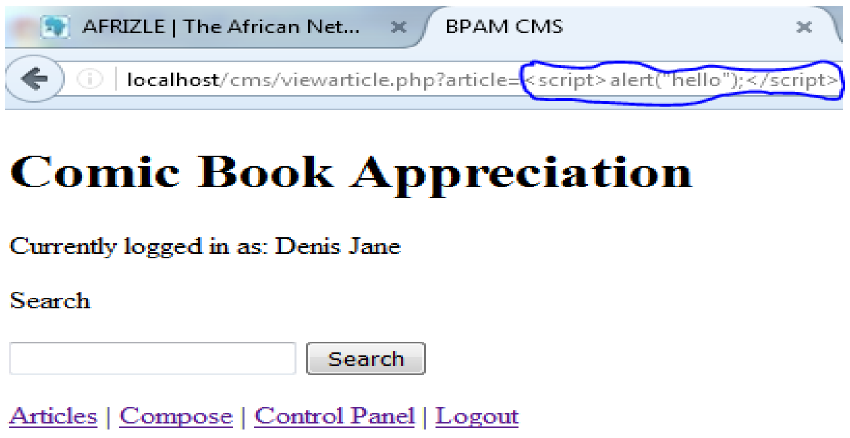
This is just another sample article

0Comments / Add one

Figure 2: Application home page listing articles.

3.2 Vulnerability Testing

Vulnerability tests were carried out on the test application site (Comic Book Appreciation) for different forms of HPP. The valid URL in the address bar shown in figure 2 was modified as an attacker would by inserting different values. The outcome of this varies from one programming platform to another. In other words, HTTP back-ends behave in different ways when there are multiple parameters with the same name. In PHP, the last parameter is used and may not interfere with the application's logic in small applications. However, some behaviors are quite bizarre. ASP and ASP.NET concatenate the values with a comma in between. Some other applications react in unexpected ways and may generate errors that can be used to initiate other attacks. In another vulnerability test that was carried out the integer 12 in the address bar was replaced with a simple JavaScript code (<script>alert("hello");</script>) as shown in figure 3.


AFRIZLE | The African Net...
BPAM CMS
localhost/cms/viewarticle.php?article=<script>alert("hello");</script>

Comic Book Appreciation

Currently logged in as: Denis Jane

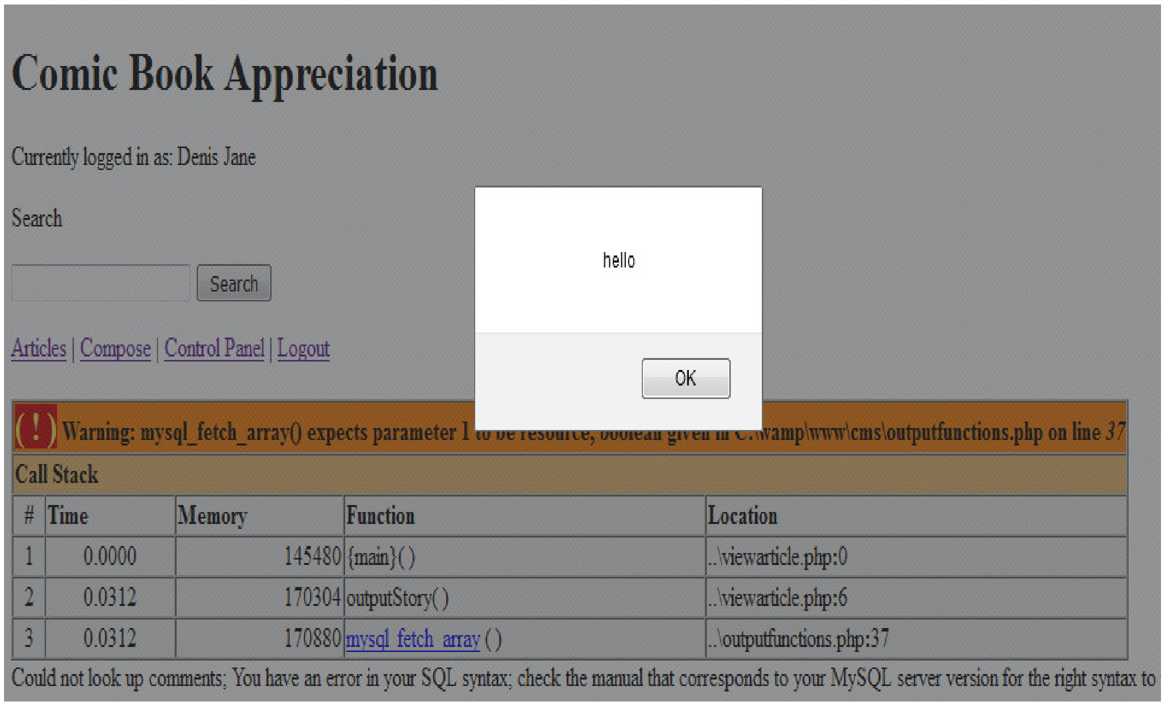
Search

Search

Articles | Compose | Control Panel | Logout

Figure 3: JavaScript code in address bar.

After pressing the “enter” button, the code executed and displayed a “hello” message and error messages appeared in the background as shown in figure 4. The code was not supposed to run if the system had been secured against HPP. No harm was done to the application because the script displayed a simple alert box and caused the application to display error messages that can reveal the application’s logic. The outcome of a more dangerous JavaScript or SQL injection code supplied to the parameter values of a sensitive application with the same level of insecurity can be disastrous. Databases can be deleted, account details can be revealed and the application compromised.


Comic Book Appreciation
Currently logged in as: Denis Jane
Search
Search
Articles | Compose | Control Panel | Logout
hello
OK

```
( ! ) Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in C:\wamp\www\cms\outputfunctions.php on line 37
```

Call Stack				
#	Time	Memory	Function	Location
1	0.0000	145480	{main}( )	..\viewarticle.php:0
2	0.0312	170304	outputStory( )	..\viewarticle.php:6
3	0.0312	170880	mysql_fetch_array ( )	..\outputfunctions.php:37

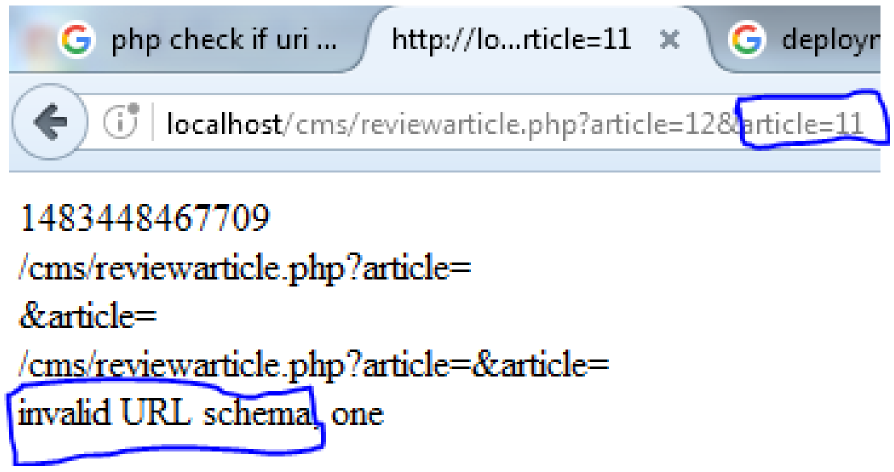
```
Could not look up comments; You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to
```

Figure 4: Effect of supplying parameters with erroneous value

3.3 Testing and Results

PVC was implemented as a security module to prevent HPP in the Comic Book Appreciation application. First a module called arc.php which collects data during training was included (imported) in the header of the application. After training the application, the module was removed and replaced with pvc.php script. The script protects the application from the HPP risks revealed during vulnerability testing of the application. The two modules were included at the top of the header.php file of the application so that they execute before the main program. Figure 5 presents a sample implementation of PVC.

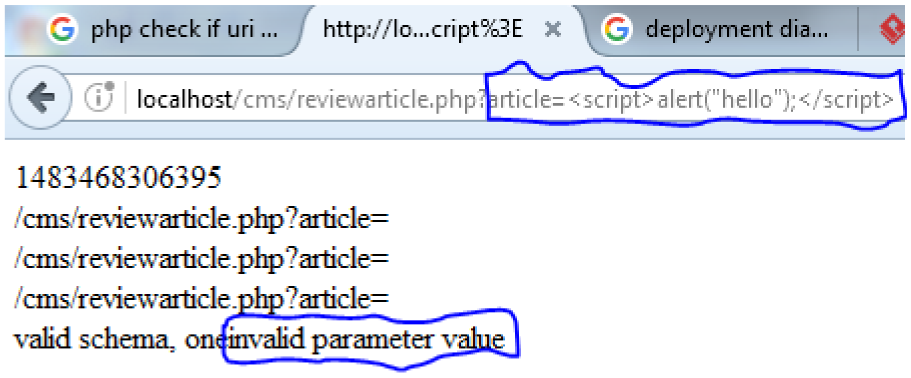
After including PVC in the application, the URL query string was tampered with. Instead of making the application behave abnormally, the HTTP request was stopped and a notice was displayed as shown in figure 5. URL parameters were also tested with erroneous values. The value of article parameter of the URI /reviewarticle.php?article=13 was changed from 13 to a JavaScript. Instead of executing the JavaScript code, it halts the script execution and did not complete the HTTP request. As shown in figure 6 PVC indicates that though the URL Schema is correct, a wrong value was supplied to a parameter.



```
localhost/cms/reviewarticle.php?article=12&article=11

1483448467709
/cms/reviewarticle.php?article=
&article=
/cms/reviewarticle.php?article=&article=
invalid URL schema, one
```

Figure 5: PVC indicating a URL Schema is Invalid.

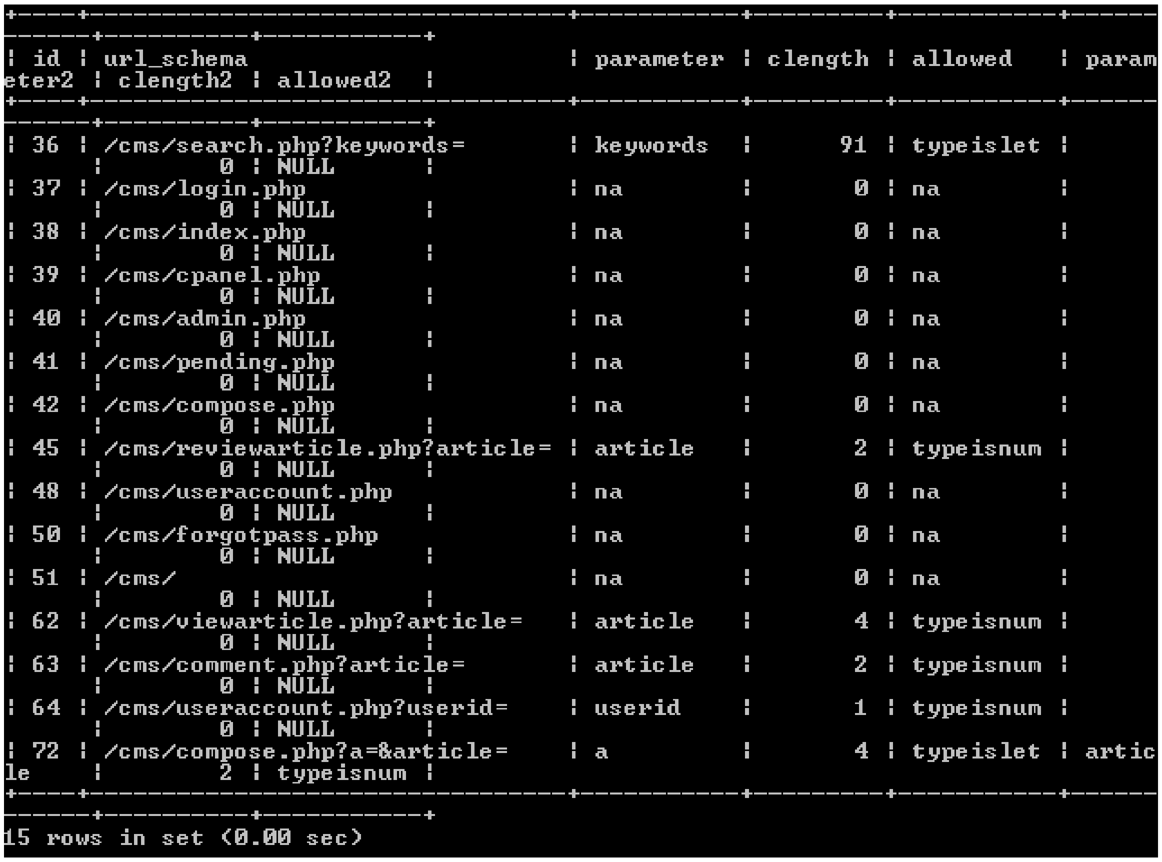


```
localhost/cms/reviewarticle.php?article=<script>alert("hello");</script>

1483468306395
/cms/reviewarticle.php?article=
/cms/reviewarticle.php?article=
/cms/reviewarticle.php?article=
valid schema, oneinvalid parameter value
```

Figure 6: PVC indicating erroneous values

The logic behind this is that while the arc.php module was collecting training data for the application, it constrained the parameter values to only integers. It also constrained the number of characters to 4 which is the number of input characters while training for that particular URL. PVC checked if the parameter value contained anything other than integers. Since the URL did not satisfy this condition, it halts the program. It would have gone further to check for character length if the URL had passed the test. Figure 7 shows the URL Schemas collected during training of the test application.

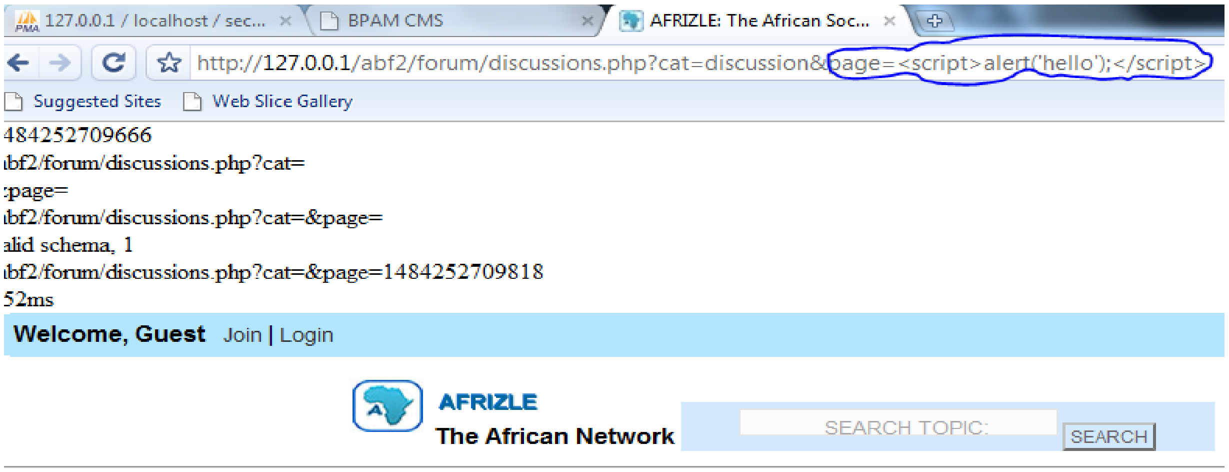


```
+----+-----------------------------------+------------+----------+------------+------
------+-----------+------------+
¦ id ¦ url_schema                        ¦ parameter  ¦ clength  ¦ allowed    ¦ param
eter2 ¦ clength2  ¦ allowed2   ¦
+----+-----------------------------------+------------+----------+------------+------
------+-----------+------------+
¦ 36 ¦ /cms/search.php?keywords=         ¦ keywords   ¦       91 ¦ typeislet  ¦
      ¦         0 ¦ NULL       ¦
¦ 37 ¦ /cms/login.php                    ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 38 ¦ /cms/index.php                    ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 39 ¦ /cms/cpanel.php                   ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 40 ¦ /cms/admin.php                    ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 41 ¦ /cms/pending.php                  ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 42 ¦ /cms/compose.php                  ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 45 ¦ /cms/reviewarticle.php?article=   ¦ article    ¦        2 ¦ typeisnum  ¦
      ¦         0 ¦ NULL       ¦
¦ 48 ¦ /cms/useraccount.php              ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 50 ¦ /cms/forgotpass.php               ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 51 ¦ /cms/                             ¦ na         ¦        0 ¦ na         ¦
      ¦         0 ¦ NULL       ¦
¦ 62 ¦ /cms/viewarticle.php?article=     ¦ article    ¦        4 ¦ typeisnum  ¦
      ¦         0 ¦ NULL       ¦
¦ 63 ¦ /cms/comment.php?article=         ¦ article    ¦        2 ¦ typeisnum  ¦
      ¦         0 ¦ NULL       ¦
¦ 64 ¦ /cms/useraccount.php?userid=      ¦ userid     ¦        1 ¦ typeisnum  ¦
      ¦         0 ¦ NULL       ¦
¦ 72 ¦ /cms/compose.php?a=&article=      ¦ a          ¦        4 ¦ typeislet  ¦ artic
le    ¦         2 ¦ typeisnum  ¦
+----+-----------------------------------+------------+----------+------------+------
------+-----------+------------+
15 rows in set (0.00 sec)
```

Figure 7: URL Schemas collected during training of test application.

PVC and ARC were also implemented on Afrizle.com (a recently launched social network site). The web application powering the social network was implemented on the same local server with the test application. ARC was able to guard the application against cases where query string parameters were duplicated but could not prevent attacks such as supplying harmful values to them. This problem was solved by implementing PVC.

ARC processed the URL even when a JavaScript code was supplied to the “page” parameter instead of a number. The JavaScript didn’t run because the application itself has some level of security against bad input but errors were still generated as shown in figure. The errors came up because SQL query meant to process the URL received unfamiliar data. It caused the application to malfunction and expose internal scripts. Developers try to prevent exposure of an application’s internal scripts and logic by configuring the application server to ignore errors. This however will not stop the application from behaving abnormally. Sometimes the effect of unexpected behavior is seen only by the attacker. At other times the database may be compromised. Figure 9 shows PVC rejecting the input to the “page” parameter while figure 10 shows how PVC halt requests due to bad parameter value on Afrizle.com. Figure 11 shows the database error generated due to the bad input to parameter value.


127.0.0.1 / localhost / sec... BPAM CMS AFRIZLE: The African Soc...

http://127.0.0.1/abf2/forum/discussions.php?cat=discussion&page=<script>alert('hello');</script>

Suggested Sites Web Slice Gallery

```
484252709666
ıbf2/forum/discussions.php?cat=
:page=
ıbf2/forum/discussions.php?cat=&page=
alid schema, 1
ıbf2/forum/discussions.php?cat=&page=1484252709818
52ms
```

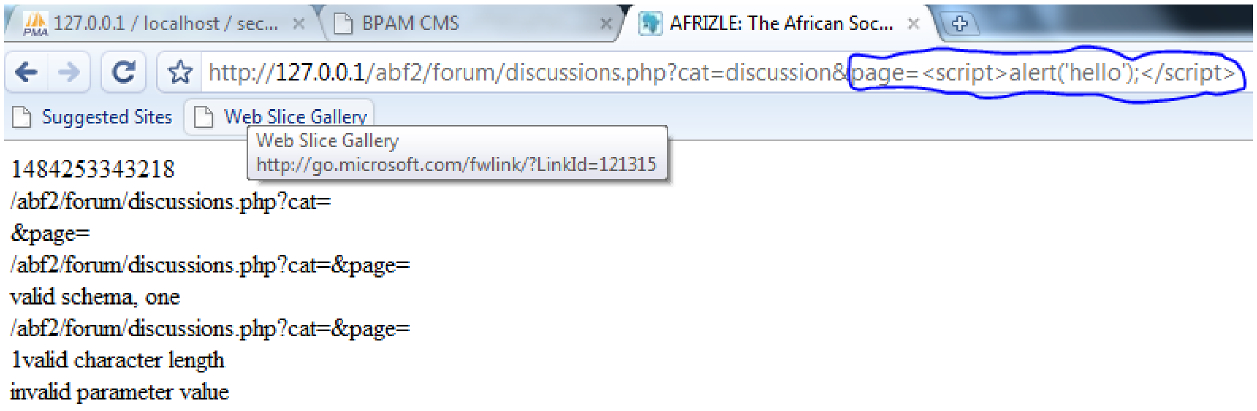
Welcome, Guest Join | Login

AFRIZLE
The African Network

SEARCH TOPIC: SEARCH

FORUM

Figure 9: ARC Unable to Prevent Bad Paramteter Value on Afrizle


127.0.0.1 / localhost / sec... BPAM CMS AFRIZLE: The African Soc...

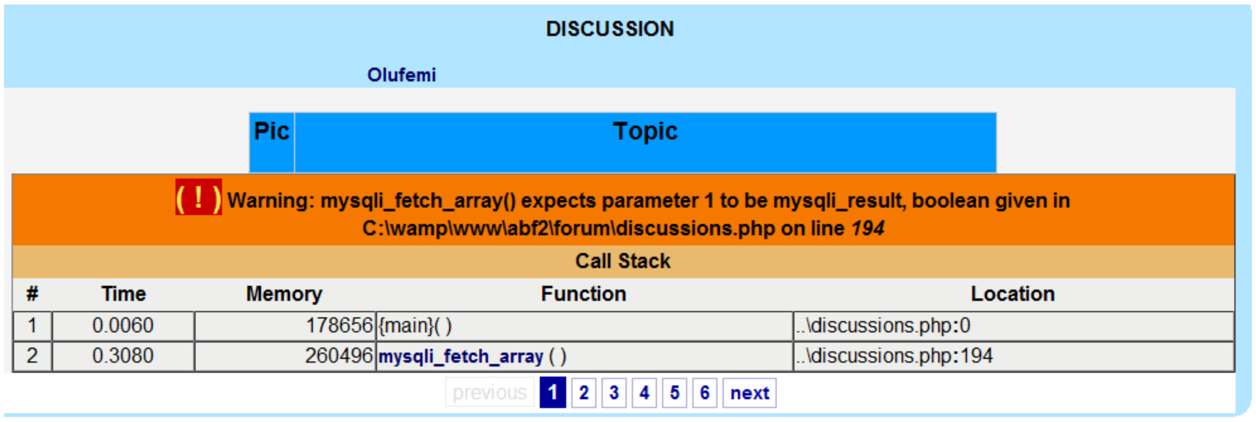
http://127.0.0.1/abf2/forum/discussions.php?cat=discussion&page=<script>alert('hello');</script>

Suggested Sites Web Slice Gallery

Web Slice Gallery
http://go.microsoft.com/fwlink/?LinkId=121315

```
1484253343218
/abf2/forum/discussions.php?cat=
&page=
/abf2/forum/discussions.php?cat=&page=
valid schema, one
/abf2/forum/discussions.php?cat=&page=
1valid character length
invalid parameter value
```

Figure 10: PVC Halt HTTP Request Due to Bad Parameter Value on Afrizle


DISCUSSION

Olufemi

Pic | Topic

(!) Warning: mysqli_fetch_array() expects parameter 1 to be mysqli_result, boolean given in C:\wamp\www\abf2\forum\discussions.php on line *194*

Call Stack

#	Time	Memory	Function	Location
1	0.0060	178656	{main}( )	..\discussions.php:0
2	0.3080	260496	mysqli_fetch_array ( )	..\discussions.php:194

previous 1 2 3 4 5 6 next

Figure 11: Database error due to bad input to parameter value

4. CONCLUSION

In this study an implementation of the Parameter Value Cache (PVC) security module for prevention of HPP attacks was carried out. The PVC prototype was implemented in PHP and evaluated on a dual core local server running at 1.6GHz with 2GB of RAM. The implementation was carefully analyzed and it was observed that PVC succeeds in addressing the limitations of ARC. It rejects all forms of HPP when tested on different URLs of the test web application. The time taken to process each URL was also noted and the average time to check each HTTP request was calculated (on the scale of milliseconds). The experiment revealed that PVC will incur additional overhead on web applications using live servers, however the overhead incurred is negligible and web application users are not likely to notice any delay.

ARC and PVC were evaluated and compared in terms of the time taken to process one URL in milliseconds and effectiveness in protecting web applications against HPP. The time taken for PVC and ARC to process URLs were compared. When an HTTP request was made, the time at which PVC and ARC started working were noted and the date-time stamp used for the data collection was converted to integer form (in milliseconds) to aid easier computation of the time. Also the same procedure was carried out after PVC and ARC had finished checking the URL. The time taken was the computed by subtracting start time from end time. For comparing throughput, URLs are crawled from the two different applications and analyzed. The first set of URLs is crawled from the test application. The second set was crawled while implementing a local version of a recently launched social network called Afrizle.com. An analysis of results obtained for both cases showed that there was an increase in computational overhead from PVC which is negligible and the user of a web application will not notice.

REFERENCES

1. Balduzzi, M., Gimenez, C., Balzarotti, D., Kirda, E. (2011). Automated discovery of parameter pollution vulnerabilities in web applications. In: Proceedings of the 18th Network and Distributed System Security Symposium.
2. Carettoni Luca and Stefano Di Paola: HTTP Parameter Pollution. OWASP EU09 Poland.
3. Athamasopoulos Elias, Kermels P. Vasileios, Polychronakis Michalis and Markalos P. Evangelos, (2013). ARC: Protecting against HTTP Parameter Polution Attacks Using Application Request Caches. Colombia University, New York, NY, USA.
4. Karim Abouelmehdi, Ahmed Bentajer, Loubna Dali, NacerSefiani (2015). A New Approach of Web Attacks Classification for Testing Security Tools at the Application Level. Journal of Theoretical and Applied Information Technology, Vol. 72, No 3.
5. Lawton George (2002). Open Source Security: Opportunity or Oxymoron? Computer vol.35, 3: 18-21.
6. Lemes Samir (2011). Information Security Management of Web Portals Based on Joomla CMS. University of Zenica, September 2011.
7. Livshits Benjamin, Martin Michael, Lam S. Monica (2006). Runtime Protection and Recovery from Web Application Vulnerabilities. Stanford University.
8. Meucci Matteo and Muller Andrew (2013). OWASP TESTING GUIDE 4.0.
9. Nguyen-Tuong Anh, Salvatore Guarnian, Dong Greane, Jeff Shirley, David Evans (2004). Automatically Hardening Web Applications Using Precise Tainting. University of Virgina, USA, June 2005.
10. OWASP TOP 10, (2013). The Ten Most Critical Application Security Risks.
11. Ronan Dunne (2013). HTTP Paramter Pollution. https://dunnesec.com/2013/05/26/http- parameter-pollution/
12. Scholte Theodore and Balzorottir Davide, Institute Eurecom, Sophia Antipolis, France; William Robertson & EnginKirda (2008). Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis. Northeastern University, Boston, USA.