
# Web Application Scan

| Document Revision | | |
|---|---|---|
| Initial Report Prepared By: | AppCheck-NG | Version: 1.0 |

| Assessment Schedule | |
|---|---|
| Assessment Performed on | Tuesday 10 June 2014 |
| Report Prepared on | Tuesday 10 June 2014 |

# 1. CONTENTS

# 2. PROJECT OVERVIEW

## 2.1. SCOPE OF ENGAGEMENT

The following IP address ranges were defined within the scope for this assessment:

| Infrastructure IP Address Ranges |
| --- |
| ecommerce.appcheck-ng.com |
| target.appcheck-ng.com |

The following web applications were defined within the scope for this assessment:

| Application URLs / IP Addresses |
| --- |
| http://ecommerce.appcheck-ng.com/ |
| http://target.appcheck-ng.com:8686/ |
| http://blog.acmeplc.local/wordpress\|/ |

## 3. SUMMARY OF VULNERABILITIES: HIGH / MEDIUM

### Vulnerability Impact Ratings

**HIGH**

**HIGH:** Successful exploitation could lead to highly privileged access to the target host or cause a denial of service condition.

Vulnerabilities are labelled "HIGH" severity if they have a CVSS base score of 7.0 -10.0.

**Medium**

**Medium**: Exploitation of the vulnerability will not directly lead to privileged access to the host, service or data. However, vulnerabilities with a Medium impact can often be combined with other flaws to elevate their impact.

Vulnerabilities will be labelled "Medium" severity if they have a base CVSS score of 4.0-6.9

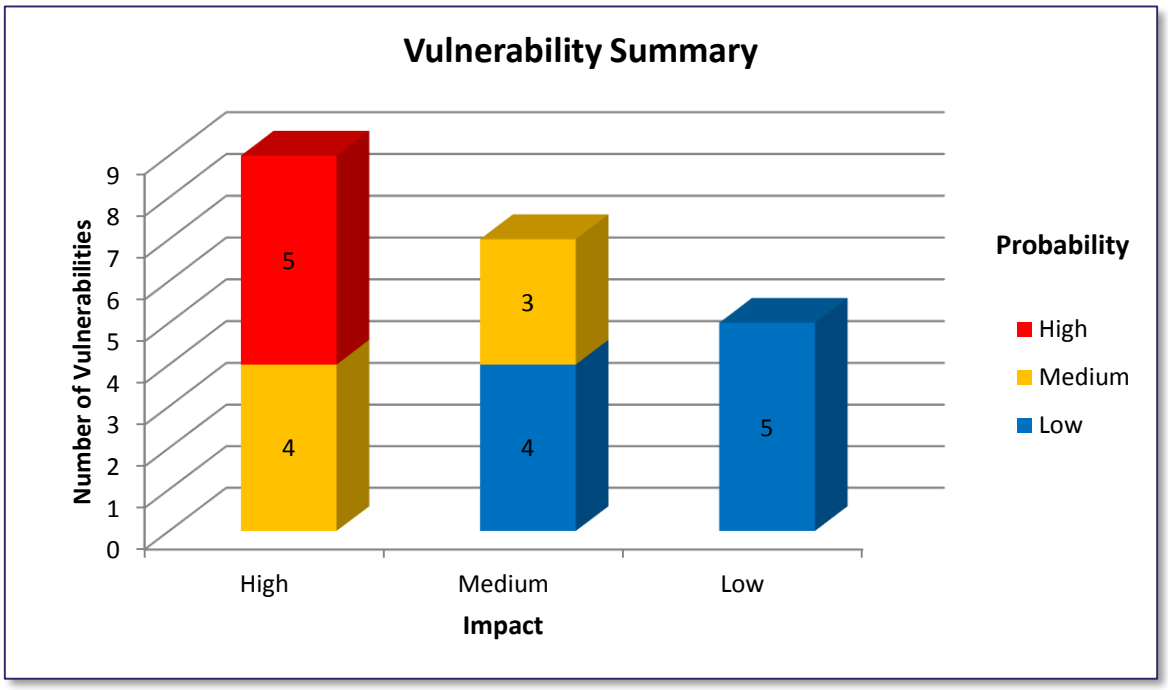
**Low**

**Low:** This impact rating is assigned to vulnerabilities that, when exploited in isolation, have a negligible impact on security. Typically vulnerabilities that disclose information that may be useful to the attacker are considered to have a low impact.

Vulnerabilities are labelled "Low" severity if they have a CVSS base score of 0.0-3.9.

## 3.1. GRAPHICAL SUMMARY

Key findings have been ranked and positioned in the following table according to the relative risk or probability of exploit. Vulnerabilities are split into 3 impact categories: High, Medium and Low. Risk is calculated by comparing the impact vs. the probability of exploit which is represented using colour coding.



## 3.2. HIGH IMPACT VULNERABILITIES

The following vulnerabilities have been assigned a **HIGH** impact rating. Successful exploitation of these vulnerabilities could lead to highly privileged access to the affected host or data or a denial of service condition.

| Impact / Ref | Description | Affected Hosts |
|---|---|---|
|  **CVSS: 9.0** **Impact/Prob: High/High** | **SQL Injection** SQL Injection Vulnerabilities were discovered within the listed resources and applications. | http://ecommerce.appcheck-ng.com |
|  **CVSS: 9.0** **Impact/Prob: High/High** | **Highly privileged SQL user account detected** The affected application appears to be using a default administrative account to connect to the backend database. The use of a highly privileged account increases the impact of a SQL Injection vulnerability and could allow the attacker to take complete control of the database host. | http://ecommerce.appcheck-ng.com |
|  **CVSS: 9.0** **Impact/Prob: High/High** | **HTML5 cross-origin resource sharing (Wildcard * and sensitive data match)** The application implements an HTML5 cross-origin resource sharing (CORS) policy which allows access from any domain. Permitting access from all could present a security risk unless the affected application consists of only unprotected public content. | http://target.appcheck-ng.com:8686 |
|  **CVSS: 9.0** **Impact/Prob: High/High** | **Command Injection (blind)** A command injection vulnerability was found on one or more web applications, which allows arbitrary system commands to be executed by an attacker. This is likely to lead directly to full system compromise. | http://target.appcheck-ng.com:8686 |

| Impact / Ref | Description | Affected Hosts |
|---|---|---|
|  **CVSS: 9.0** **Impact/Prob: High/High** | **Flash: Tainted data passed to a dangerous function: sendToJs**<br>Adobe Flash content is commonly invoked with a number of configuration parameters known as *FlashVars*. Although Flashvars are typically supplied within the body of the HTML document, it is also possible to supply them directly via the query string (e.g moive.swf?flashvar1=value&flashvar2=value2). If a Flashvar value is passed to a function that performs navigation or JavaScript execution, it may be possible to perform a Cross Site Scripting attack (XSS) (Read more on XSS). | http://target.appcheck-ng.com:8686 |
|  **CVSS: 9.0** **Impact/Prob: High/High** | **Flash Cross Site Scripting via ExternalInterface.call**<br>Adobe Flash content is commonly invoked with a number of configuration parameters known as *FlashVars*. Although Flashvars are typically supplied within the body of the HTML document, it is also possible to supply them directly via the query string (e.g moive.swf?flashvar1=value&flashvar2=value2). If a Flashvar value is passed to a function that performs navigation or JavaScript execution, it may be possible to perform a Cross Site Scripting attack (XSS) (Read more on XSS). | http://target.appcheck-ng.com:8686 |
|  **CVSS: 9.0** **Impact/Prob: High/High** | **WordPress user account configured with a weak password**<br>It was possible to guess the account password for one of more enumerated WordPress Account. | http://blog.acmeplc.local |
|  **CVSS: 8.3** **Impact/Prob: High/Medium** | **Path Traversal Vulnerability**<br>Path Traversal Vulnerabilities occur when server side scripts process file paths supplied<br>by client without correctly validating the file path. This check determines if operating system files can be accessed by manipulating script parameters to include the parent path specifier ../ and .. Path traversal vulnerabilities could be exploited in an attempt to access sensitive configuration information such as user databases and application source code. Path traversal vulnerabilities are often exploited to gain information for use in other attacks against the system such as brute force authentication attacks. | http://ecommerce.appcheck-ng.com |

| Impact / Ref | Description | Affected Hosts |
|---|---|---|
|  <br>**CVSS: 8.3**<br>**Impact/Prob:**<br>**High/Medium** | **DOM-based Cross-Site Scripting Vulnerability**<br>The listed web applications are vulnerable to one or more DOM-based Cross Site Scripting (DOM-XSS) vulnerabilities. | http://target.appcheck-ng.com:8686 |
|  <br>**CVSS: 8.3**<br>**Impact/Prob:**<br>**High/Medium** | **Reflected Cross-site Scripting**<br>Cross Site scripting vulnerabilities were discovered across several of the assessed web applications. A malicious attacker could exploit these flaws to perform a social engineering attack against users of the affected application. | http://ecommerce.appcheck-ng.com,<br>http://target.appcheck-ng.com:8686 |
|  <br>**CVSS: 8.3**<br>**Impact/Prob:**<br>**High/Medium** | **Stored Cross-site Scripting**<br>Cross Site Scripting (XSS) vulnerabilities were discovered across several of the assessed web applications. A malicious attacker could exploit these flaws to perform a social engineering attack against users of the affected application. | http://target.appcheck-ng.com:8686 |

## 3.3. MEDIUM IMPACT VULNERABILITIES

The following vulnerabilities have been assigned a **Medium** impact rating.

| Impact / Ref | Description | Affected Hosts |
|---|---|---|
|  **CVSS:** 5.1 **Impact/Prob:** Medium/Medium | **HTML5 cross-origin resource sharing (Wildcard *)** The application implements an HTML5 cross-origin resource sharing (CORS) policy which allows access from any domain. Permitting access from any origin could present a security risk unless the affected application hosts only unprotected public content. | http://target.appcheck-ng.com:8686 |
|  **CVSS:** 5.1 **Impact/Prob:** Medium/Medium | **HTML5 cross-origin resource sharing (Domain Wildcard and sensitive data match)** The application implements an HTML5 cross-origin resource sharing (CORS) policy which allows access from a wild card domain. Permitting access from a range or hosts could present a security risk unless the affected application consists of only unprotected public content. | http://target.appcheck-ng.com:8686 |
|  **CVSS:** 5.1 **Impact/Prob:** Medium/Low | **Cookie Scoped to Parent Domain** A cookie's domain attribute determines which domains can access the cookie. Browsers will automatically submit the cookie in requests to in-scope domains, and those domains will also be able to access the cookie via JavaScript. If a cookie is scoped to a parent domain, then that cookie will be accessible by the parent domain and also by any other subdomains of the parent domain. If the cookie contains sensitive data (such as a session token) then this data may be accessible by less trusted or less secure applications residing at those domains, leading to a security compromise. | http://target.appcheck-ng.com:8686 |
|  **CVSS:** 5.1 **Impact/Prob:** Medium/Low | **Password Reset CSRF Vulnerability** A potential password reset form was detected that does not validate the users old password or implement CSRF protection | http://target.appcheck-ng.com:8686 |

| Impact / Ref | Description | Affected Hosts |
|---|---|---|
|  <br>**CVSS:** 5.1<br>**Impact/Prob:**<br>Medium/Low | **Password submitted via the GET method**<br>User credentials transmitted via the GET method. | http://target.appcheck-ng.com:8686 |
|  <br>**CVSS:** 5.1<br>**Impact/Prob:**<br>Medium/Low | **Cleartext Submission of password**<br>User authentication credentials are transmitted in clear text | http://blog.acmeplc.local |
|  <br>**CVSS:** 5.1<br>**Impact/Prob:**<br>Medium/Medium | **Open Redirect Vulnerability**<br>A URL Redirect vulnerability was discovered. It is possible to craft a link to the affected component to redirect the user to a malicious website. An attacker could exploit this vulnerability to trick users into accessing malicious content on the basis that they trust the affected application | http://target.appcheck-ng.com:8686 |

# 4. ASSESSMENT RESULTS

## 4.1. SQL INJECTION

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://ecommerce.appcheck-ng.com

SQL Injection Vulnerabilities were discovered within the listed resources and applications.

SQL injection vulnerabilities occur when user supplied data is inserted into a dynamic SQL Query without proper handling or input validation. This vulnerability could be exploited by a malicious attacker to read, modify, delete or create SQL table data. In many cases it is possible to attack the SQL server directly via this vulnerability and potentially and gain administrative control of the database Server.

### 4.1.1. REMEDIATION

Data processed from an external source such as user input should be subject to an input validation filter. The most secure approach is to white list known good characters such as those within the Aa-Zz range and deny all others.

To gain a better understanding of SQL Injection issues and to learn different ways to prevent the problem see the following resources:

### 4.1.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://ecommerce.appcheck-ng.com | /product_detail.asp?prod=15 | payload.product_id |
| http://ecommerce.appcheck-ng.com | /admin/cms_auth.asp | payload.username |
| http://ecommerce.appcheck-ng.com | /product_list.asp?cat1=1 | cat1 |
| http://ecommerce.appcheck-ng.com | /product_list.asp?cat1=1&records=2 | cat1 |
| http://ecommerce.appcheck-ng.com | /product_detail.asp?prod=15 | prod |
| http://127.0.0.1 | /mysqlapp/newsblind.php?id=1 | id |

### 4.1.3. TECHNICAL ANALYSIS

#### Example: http://ecommerce.appcheck-ng.com/product_detail.asp?prod=15 [param: payload.product_id]

**Technical Details**

SQL Injection was detected via the following methods:

**Detection Method 1:** Injected Time Delay Inference:

The vulnerability was detected by measuring a controlled response delay over several high-low injected delay cycles:

```
Delay cycle 0: low=1.17, high=10.20, delta=9.03 (injected=10)
Delay cycle 1: low=1.17, high=10.22, delta=9.05 (injected=10)
Delay cycle 2: low=1.17, high=10.18, delta=9.01 (injected=10)
Delay cycle 3: low=1.17, high=10.21, delta=9.04 (injected=10)
Delay cycle 4: low=1.19, high=10.18, delta=8.99 (injected=10)
Random delay cycle 0: low delay=1.19
Random delay cycle 1: high delay=10.19 (injected=10)
Random delay cycle 2: high delay=10.19 (injected=10)
Random delay cycle 3: high delay=10.15 (injected=10)
Random delay cycle 4: high delay=10.19 (injected=10)
Random delay cycle 5: high delay=10.17 (injected=10)
Random delay cycle 6: high delay=10.17 (injected=10)
Random delay cycle 7: high delay=10.17 (injected=10)
Random delay cycle 8: low delay=1.15
Random delay cycle 9: low delay=1.22The following request can be used to recreate the issue:
POST /product_detail.asp?prod=15 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: ASPSESSIONIDQSTQTSRR=GCBMKDLALJBLCHMNBPKHKDPP
Host: ecommerce.appcheck-ng.com
Content-Length: 163

product_quantity=1&buynow.x=0&buynow.y=0&product_code=4TR43G&product_name=Shoes&product_image=tshirt_t.jpg&Charon_Cart=1&product_id=15+waitfor+delay+'0%3A0%3A10'--
```

**Exploit**

Successfully exploited Blind MSSQL Injection.

**SQL Server username:** sa

#### Example: http://ecommerce.appcheck-ng.com/admin/cms_auth.asp [param: payload.username]

**Technical Details**

SQL Injection was detected via the following methods:

**Detection Method 1:** Database Error Messages

A SQL statement sumbmitted via the affected parameter was successfully executed by the SQL server.

*Detection Payload*

The following payload is designed to return the string chFibSahxo when executed by a the database server:

```
or 1 in (char(0x63)+char(0x68)+char(0x46)+char(0x69)+char(0x62)+char(0x53)+char(0x61)+char(0x68)+char(0x78)+char(0x6f))
```

The following request can be used to recreate the issue:

```
POST /admin/cms_auth.asp HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: ASPSESSIONIDQSTQTSRR=JOAMKDLALGEJPNDJIJFELHPK
Host: ecommerce.appcheck-ng.com
Content-Length: 223

password=Pa%24%24w0rd123&username=Test'or+1+in+%28char%280x63%29%2Bchar%280x68%29%2Bchar%280x46%29%2Bchar%280x69%29%2Bchar%280x62%29%2Bchar%280x53%29%2Bchar%280x61%
29%2Bchar%280x68%29%2Bchar%280x78%29%2Bchar%280x6f%29%29--+
```

**Exploit**

Successfully exploited SQL Injection. The SQL Server Username, a list of databases and, a list of tables from the current database was extracted.

**SQL Server username:** sa

**List of databases:**

- master
- master1
- MF20
- model
- msdb
- securesec
- tempdb
- users

**List of tables in current database:**

- cmd
- news
- tblUsers
- users123

*Additional examples were omitted for brevity*

## 4.2. HIGHLY PRIVILEGED SQL USER ACCOUNT DETECTED

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://ecommerce.appcheck-ng.com

The affected application appears to be using a default administrative account to connect to the backend database. The use of a highly privileged account increases the impact of a SQL Injection vulnerability and could allow the attacker to take complete control of the database host.

### 4.2.1. REMEDIATION

Create and deploy a new user account with the least amount of privilege necessary for the application to operate.

### 4.2.2. TECHNICAL ANALYSIS

#### Example: http://ecommerce.appcheck-ng.com

The following database server administative user account was identified: **sa**

## 4.3. HTML5 CROSS-ORIGIN RESOURCE SHARING (WILDCARD * AND SENSITIVE DATA MATCH)

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://target.appcheck-ng.com:8686

The application implements an HTML5 cross-origin resource sharing (CORS) policy which allows access from any domain. Permitting access from all could present a security risk unless the affected application consists of only unprotected public content.

NOTE: The impact of this flaw has been raised from MEDIUM to HIGH since it appears that sensitive information such as session tokens may be exposed within the page body,

### 4.3.1. REMEDIATION

Review the domains which are allowed by the CORS policy in relation to any sensitive content within the application.

### 4.3.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686

**Vulnerable URL Examples**

http://target.appcheck-ng.com:8686/resources/Members_Sensitive (Sensitive Data Match:sessionid)

## 4.4. COMMAND INJECTION (BLIND)

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://target.appcheck-ng.com:8686

A command injection vulnerability was found on one or more web applications, which allows arbitrary system commands to be executed by an attacker. This is likely to lead directly to full system compromise.

### 4.4.1. REMEDIATION

Ensure all input is sanitised before being passed to the process creation API, ideally against a whitelist of values.

See the following link for more information: Command Injection

### 4.4.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /command/linux?v1=hi | v1 |

### 4.4.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/command/linux?v1=hi [param: v1]

The vulnerability was detected by measuring a controlled response delay over several high-low injected delay cycles:

```
Delay cycle 0: low=1.08, high=10.08, delta=8.99 (injected=10)
Delay cycle 1: low=1.12, high=10.13, delta=9.02 (injected=10)
Delay cycle 2: low=1.13, high=10.14, delta=9.00 (injected=10)
Random delay cycle 0: low delay=1.01
Random delay cycle 1: high delay=10.02 (injected=10)
Random delay cycle 4: high delay=10.03 (injected=10)
```

**Exploit:** The Unix ID command was successfully executed:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

## 4.5. FLASH: TAINTED DATA PASSED TO A DANGEROUS FUNCTION: SENDTOJS

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://target.appcheck-ng.com:8686

Adobe Flash content is commonly invoked with a number of configuration parameters known as *FlashVars*. Although Flashvars are typically supplied within the body of the HTML document, it is also possible to supply them directly via the query string (e.g moive.swf?flashvar1=value&flashvar2=value2). If a Flashvar value is passed to a function that performs navigation or JavaScript execution, it may be possible to perform a Cross Site Scripting attack (XSS) (Read more on XSS).

The locally defined function sendToJs passes one or more of its supplied arguments to a built in function that is considered unsafe (ExternalInterface.call).

**Function Code**:

```
function sendToJs(param1:String)
      void {
      var _loc2_:String = null; if(ExternalInterface.available) {
            _loc2_ = getSharedObjectValue(); ExternalInterface.call(param1,_loc2_);
      } else {
      debug("ExternalInterface not available");
      }
}
```

A FlashVar variable named **onResponse** was assigned to the local variable which was in turn passed into the dangerous function **sendToJs**

### 4.5.1. REMEDIATION

Strictly Filter User Input

### 4.5.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /flash/tracker2js.swf | onResponse |

### 4.5.3. TECHNICAL ANALYSIS

Example: http://target.appcheck-ng.com:8686/flash/tracker2js.swf [param: onResponse]

**Function Name**

sendToJs

**Vulnerable Flash Variable**

onResponse

**Flash Variables**

&onResponse=

**Code Snippet**

The following ActionScript code is affected:

```
function Tracker2js() {
         super();
         Security.exactSettings = false;
         SharedObject.defaultObjectEncoding = ObjectEncoding.AMF0;
         var _loc1_:Object = stage.loaderInfo.parameters;
         var _loc2_:String = _loc1_.onResponse;
         if(_loc2_)
         {
            sendToJs(_loc2_);
         }
         else
         {
            debug("No javascript function to call - flashvars onResponse=functname");
         }
      }
```

## 4.6. FLASH CROSS SITE SCRIPTING VIA EXTERNALINTERFACE.CALL

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://target.appcheck-ng.com:8686

Adobe Flash content is commonly invoked with a number of configuration parameters known as *FlashVars*. Although Flashvars are typically supplied within the body of the HTML document, it is also possible to supply them directly via the query string (e.g moive.swf?flashvar1=value&flashvar2=value2). If a Flashvar value is passed to a function that performs navigation or JavaScript execution, it may be possible to perform a Cross Site Scripting attack (XSS) (Read more on XSS).

**The following XSS vulnerabilities were idetified**

The ActionScript Function **ExternalInterface.call** is used to execute JavaScript within the web browser. If unfiltered user controllable input is passed to this function, it may be possible to perform a Cross Site Scripting attack.

A tainted variable named *_highlighterId* loaded via the **highlighterId** flashVar is passed to the function **ExternalInterface.call**

### 4.6.1. REMEDIATION

Strictly Filter User Input

### 4.6.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /flash/clipboard.swf | highlighterId |

### 4.6.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/flash/clipboard.swf [param: highlighterId]

**Function Name**: ExternalInterface.call

**Vulnerable Flash Variable:** highlighterId

**Flash Variables**: &highlighterId=

**Code Snippet**

The following ActionScript code is affected:

```
function executeCommand(param1:String, param2:String=null) : Object {
         if(ExternalInterface.available)
         {
            return ExternalInterface.call("SyntaxHighlighter.toolbar.executeCommand",null,null,_highlighterId,"copyToClipboard",
               {
                  "command":param1,
                  "message":param2
               });
         }
         return null;
      }
```

## 4.7. WORDPRESS USER ACCOUNT CONFIGURED WITH A WEAK PASSWORD

**CVSS Score:** 9.0 **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:P/A:P **Impact/Probability:** High/High

**Affected:** http://blog.acmeplc.local

It was possible to guess the account password for one of more enumerated WordPress Accounts.

### 4.7.1. REMEDIATION

Set a strong password and enable account lockouts

### 4.7.2. TECHNICAL ANALYSIS

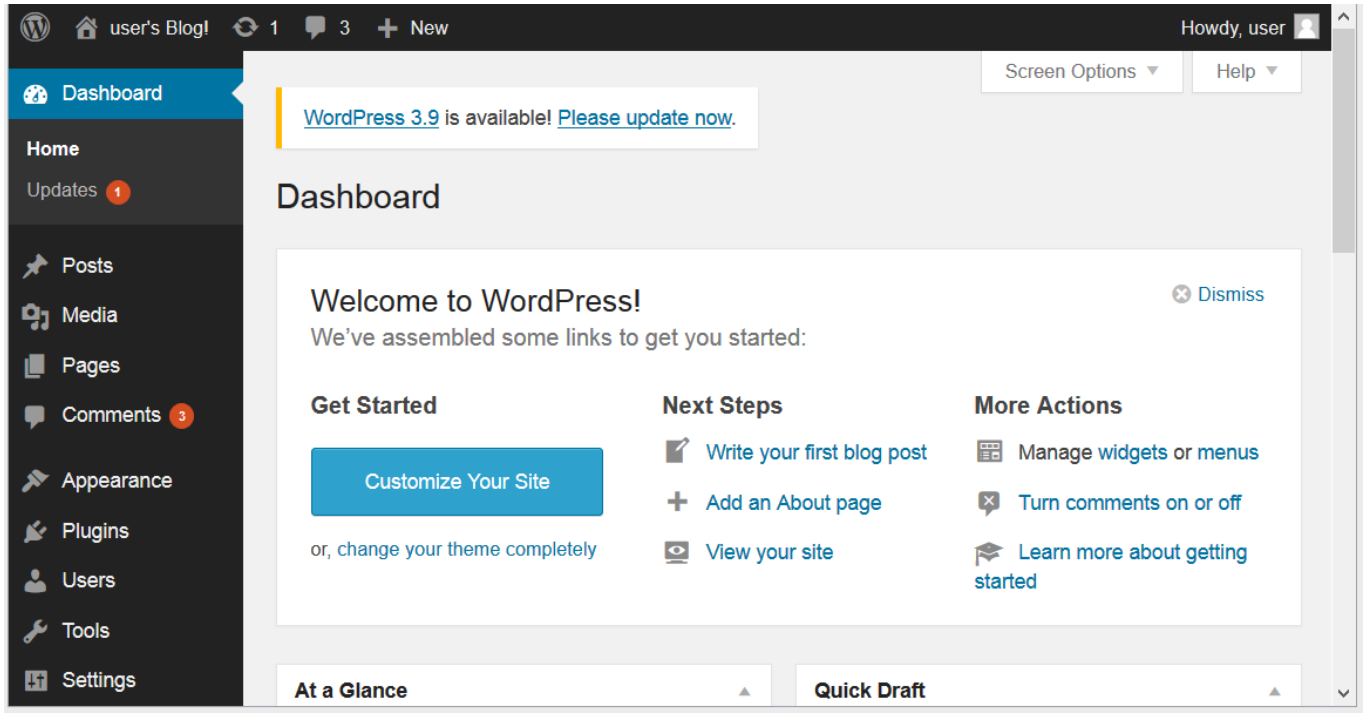
The following user account passwords were guessed via a simple dictionary attack.

#### Example: http://blog.acmeplc.local

**Cracked Accounts**

- **Username:** test user **Password:** password
- **Username:** test.user2 **Password:** password1
- **Username:** user **Password:** bitnami

**Exploit:** The following screenshot was captured by Appcheck-Ng following successful authentication:



## 4.8. PATH TRAVERSAL VULNERABILITY

**CVSS Score:** 8.3 **CVSS Vector:** AV:N/AC:M/Au:N/C:C/I:P/A:P **Impact/Probability:** High/Medium

**Affected:** http://ecommerce.appcheck-ng.com

Path Traversal Vulnerabilities occur when server side scripts process file paths supplied by client without correctly validating the file path. This check determines if operating system files can be accessed by manipulating script parameters to include the parent path specifier ../ and .. Path traversal vulnerabilities could be exploited in an attempt to access sensitive configuration information such as user databases and application source code. Path traversal vulnerabilities are often exploited to gain information for use in other attacks against the system such as brute force authentication attacks.

### 4.8.1. REMEDIATION

Strictly validate user supplied input using a white list filter to ensure that complete file paths or directory traversal structures are not permitted. The following additional recommendations were taken from; https://www.owasp.org/index.php/File_System#Path_traversal

- When a URI request for a file/directory is to be made, build a full path to the file/directory if it exists, and normalize all characters (e.g., %20 converted to spaces).
- It is assumed that a 'Document Root' fully qualified, normalized, path is known, and this string has a length *N*. Assume that no files outside this directory can be served. Ensure that the first *N* characters of the fully qualified path to the requested file is exactly the same as the 'Document Root'.
- Use indexes rather than actual portions of file names when templating or using language files (ie value 5 from the user submission = Czechoslovakian, rather than expecting the user to return “Czechoslovakian”).
- Ensure the user cannot supply all parts of the path – surround the file request with your own path code.
- If forced to use user input for file operations, normalize the input before using in a file I/O API.

### 4.8.2. TECHNICAL OVERVIEW

**Vulnerable parameters:**

| App | URI | Parameter |
|---|---|---|
| http://ecommerce.appcheck-ng.com | /download.aspx?pdf=Brochure.pdf | pdf |

It was possible to retrieve the c:\windows\win.ini file by submitting the following URL

http://ecommerce.appcheck-ng.com/download.asp?pdf=%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5C..%5Cwindows%5Cwin.ini

## 4.9. DOM-BASED CROSS-SITE SCRIPTING VULNERABILITY

**CVSS Score:** 8.3 **CVSS Vector:** AV:N/AC:M/Au:N/C:C/I:P/A:P **Impact/Probability:** High/Medium

**Affected:** http://target.appcheck-ng.com:8686

The listed web applications are vulnerable to one or more DOM-based Cross Site Scripting (DOM-XSS) vulnerabilities.

DOM-XSS is a subtle variant of XSS where the vulnerability resides within the validation of *browser-side* rather than server-side code. Often, a piece of JavaScript code parses a portion of the current URL and this gets written back into the current page's DOM without sanitisation, for example:

```
<script>
  document.write("Current URL: " + document.location.href + ".");
</script>
```

### 4.9.1. REMEDIATION

Sanitise all user controlled inputs within JavaScript code, using white-listing or regular expressions. Note, server-side input sanitisation cannot prevent this attack.

References:

- DOM-Based XSS

### 4.9.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /users/news?url=http://www.Appcheck-NG.com | URL |
| http://target.appcheck-ng.com:8686 | /users/new_topic?id=x#vuln | URL |
| http://target.appcheck-ng.com:8686 | /users/redirect?url=/xss | URL |

### 4.9.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/users/news?url=http://www.Appcheck-NG.com [param: URL]

**Accuracy**

This vulnerability was detected through actual injected JavaScript execution via the following URL :

http://target.appcheck-ng.com:8686/users/news?url=http://www.Appcheck-NG.com”-alert(1)//

**Triggering**

The vulnerability is triggered by a click event

http://target.appcheck-ng.com:8686/users/news?url=’-alert(1)//

### Example: http://target.appcheck-ng.com:8686/users/new_topic?id=x#vuln [param: id]

**Accuracy**

This vulnerability was detected through actual injected JavaScript execution.

http://target.appcheck-ng.com:8686/users/new_topic?id=x#vuln”><ScRiPt>alert(1)</script>

**Triggering**

The vulnerability is triggered directly upon loading the vulnerable page

### Example: http://target.appcheck-ng.com:8686/users/redirect?url=/xss [param: URL]

**Accuracy**

This vulnerability was detected through actual injected JavaScript execution.

http://target.appcheck-ng.com:8686/users/redirect?url=javascript:alert(1)

**Triggering**

The vulnerability is triggered directly upon loading the vulnerable page

s

## 4.10. REFLECTED CROSS-SITE SCRIPTING

**CVSS Score:** 8.3 **CVSS Vector:** AV:N/AC:M/Au:N/C:C/I:P/A:P **Impact/Probability:** High/Medium

**Affected:** http://ecommerce.appcheck-ng.com, http://target.appcheck-ng.com:8686

Cross Site Scripting (XSS) vulnerabilities occur when data submitted to the application is not properly handled before being embedded within the application's response or stored for later retrieval.

Reflected XSS vulnerabilities are typically exploited by embedding malicious script code within links to the application. The attacker would then attempt to coerce the user into following the maliciously crafted link via a social engineering attack such as a Phishing email.

Upon clicking the malicious link the embedded script code is inserted into the server's response and executed within user's web browser.

XSS vulnerabilities can by exploited to hijack authenticated user sessions, perform a virtual defacement and deploy Trojan functionality.

### 4.10.1. REMEDIATION

To Prevent XSS attacks a multi-layered approach is recommended;

Validate Input

Input received from the client should be strictly validated on the server side before any further processing takes place.

The filter should use a "White List" approach by only accepting "Known Good" characters. Validation should be performed on a per-field basis and should endeavour to be as strict as possible.

Ensure that data is fully canonicalised and decoded before being compared to the filter.

Encode Output

All client supplied data should be HMTL encoded at the point where it is displayed to the user. This includes request data such as query string parameters and data retrieved from storage.It is recommended that all alphanumeric characters be HTML encoded to avoid XSS. However the following characters must be encoded:" ' &<>

### 4.10.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://ecommerce.appcheck-ng.com | /account_script.aspx?origin= | payload.address1 |

| App | URI | Parameter |
|---|---|---|
| http://ecommerce.appcheck-ng.com | /account.aspx?origin=checkout_shipping.aspx | origin |
| http://target.appcheck-ng.com:8686 | /users/login?data= | data |

### 4.10.3. TECHNICAL ANALYSIS

#### Example: http://ecommerce.appcheck-ng.com/account_script.aspx?origin= [param: payload.address1]

Tainted data submitted to the application via the **address1 (POST)** parameter is insecurely returned within the applications response. By including the **"** delimiter within the submitted payload it was possible to break out the string data and add an arbitrary attribute (named *Husfk* in this example).

The attacker may exploit this to add a JavaScript event handler designed to execute JavaScript. It may also be possible to form a new tag by closing the affected tag before appending a new one, e.g. `"><img src=a onerror=alert(1)>`

To detect this flaw the character string *Husfk* was submitted to the application and was insecurely embedded within the following HTML tags:

- Attribute named *Husfk* added to `<input name="address1" type="text" class="input" id="address1" value="1"Husfk="" size="35" style="width:200px;" />`

**Accuracy**

This vulnerability was confirmed through actual JavaScript execution. The following payload will execute the JavaScript code *alert(1)*:

```
"autofocus onfocus="alert(1)
```

#### Example: http://ecommerce.appcheck-ng.com/account.aspx?origin=checkout_shipping.aspx [param: origin]

Tainted data submitted to the application via the **origin (Query String)** parameter is insecurely embedded within the page body. To exploit this flaw the attacker could submit HTML markup designed to execute JavaScript.

It was possible to create a new tag within the page named UIAaUjR by submitting the payload `'"><UIAaUjR></UIAaUjR>`

**Accuracy**

This vulnerability was confirmed through actual JavaScript execution. The following payload will execute the JavaScript code *alert(1)*:

```
'"></ScRipT><script>alert(1)</sCriPt>
```

**Example Exploit URL**

The following URL should trigger a JavaScript `alert(1)` when using the Firefox browser: http://ecommerce.appcheck-ng.com/account.aspx?origin=checkout_shipping.aspx'"></ScRipT><script>alert%281%29</sCriPt>

### Example: http://target.appcheck-ng.com:8686/users/login?data=hello [param: data]

Tainted data submitted to the application within the **data (Query String)** is insecurely embedded within an existing JavaScript Event Handler. It may be possible to add inject JavaScript code via the affected parameter that will be executed when the handler is invoked.

Payloads such as `"-alert(1)-"` or `'-alert(1)-'` will often demonstrate this flaw by displaying an alert box.

**Note:** HTML Entity encoded data within an event handler is automatically decoded before being interpreted by the browser. Therefore, even when metacharacters such as ' and " are HTML encoded (e.g. as ' and "), it is still possible to break out of quoted strings and inject JavaScript.

By injecting a **'** delimiter it is possible to break out of the string context and append JavaScript The Example below shows the tainted value **'izOhvAa** added to the event handler script as code rather than data:

- <a onclick="x('http://127.0.0.1:8686/users/gosXSSEvent?data=hello'**izOhvAa**');" href=''>

**Accuracy**

This vulnerability was detected through signature based techniques.

## 4.11. STORED CROSS-SITE SCRIPTING

**CVSS Score:** 8.3 **CVSS Vector:** AV:N/AC:M/Au:N/C:C/I:P/A:P **Impact/Probability:** High/Medium

**Affected:** http://target.appcheck-ng.com:8686

Cross Site Scripting (XSS) vulnerabilities occur when data submitted to the application is not properly handled before being embedded within the application's response or stored for later retrieval.

Reflected XSS vulnerabilities are typically exploited by embedding malicious script code within links to the application. The attacker would then attempt to coerce the user into following the maliciously crafted link via a social engineering attack such as a Phishing email.

Upon clicking the malicious link the embedded script code is inserted into the server's response and executed within user's web browser.

XSS vulnerabilities can by exploited to hijack authenticated user sessions, perform a virtual defacement and deploy Trojan functionality.

### 4.11.1. REMEDIATION

To Prevent XSS attacks a multi-layered approach is recommended:

**Validate Input**

Input received from the client should be strictly validated on the server side before any further processing takes place.

The filter should use a "White List" approach by only accepting "Known Good" characters. Validation should be performed on a per-field basis and should endeavour to be as strict as possible.

Ensure that data is fully canonicalised and decoded before being compared to the filter.

**Encode Output**

All client supplied data should be HMTL encoded at the point where it is displayed to the user. This includes request data such as query string parameters and data retrieved from storage.It is recommended that all alphanumeric characters be HTML encoded to avoid XSS. However the following characters must be encoded:" ' &<>

### 4.11.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /users/stored | payload.name |
| http://target.appcheck-ng.com:8686 | /blog/postblog | synth |

### 4.11.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/users/stored [param: payload.name]

The injected script is executed at the following url:

http://target.appcheck-ng.com:8686/users/render_stored

**Accuracy**

Note, this vulnerability is not yet accurately confirmed.

#### Example: http://target.appcheck-ng.com:8686/blog/postblog [param: synth]

The injected script is executed at the following url:

http://target.appcheck-ng.com:8686/blog/home

**Accuracy**

Note, this vulnerability is not yet accurately confirmed.

## 4.12. COOKIE SCOPED TO PARENT DOMAIN

**CVSS Score:** 5.1 **CVSS Vector:** AV:N/AC:H/Au:N/C:P/I:P/A:P **Impact/Probability:** Medium/Low

**Affected:** http://target.appcheck-ng.com:8686

A cookie's domain attribute determines which domains can access the cookie. Browsers will automatically submit the cookie in requests to in-scope domains, and those domains will also be able to access the cookie via JavaScript. If a cookie is scoped to a parent domain, then that cookie will be accessible by the parent domain and also by any other subdomains of the parent domain. If the cookie contains sensitive data (such as a session token) then this data may be accessible by less trusted or less secure applications residing at those domains, leading to a security compromise.

### 4.12.1. REMEDIATION

By default, cookies are scoped to the issuing domain and all subdomains. If you remove the explicit domain attribute from your Set-cookie directive, then the cookie will have this default scope, which is safe and appropriate in most situations. If you particularly need a cookie to be accessible by a parent domain, then you should thoroughly review the security of the applications residing on that domain and its subdomains, and confirm that you are willing to trust the people and systems which support those applications.

### 4.12.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /session_setup/cookies | cookie.some_session1 |

### 4.12.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/session_setup/cookies [param: cookie.some_session1]

**Affected Cookie**

some_session1=276a9fae774c40a09dededcae40ef7b4; Path=/; domain=.appecheck-ng.co.uk

## 4.13. PASSWORD RESET CSRF VULNERABILITY

**CVSS Score:** 5.1 **CVSS Vector:** AV:N/AC:H/Au:N/C:P/I:P/A:P **Impact/Probability:** Medium/Low

**Affected:** http://target.appcheck-ng.com:8686

A password reset page was detected that does not appear to validate the users old password or implement other forms for Cross-Site-Request Forgery (CSRF) protection. It maybe possible to reset the users password by invoking link or form post from a malicous website

### 4.13.1. REMEDIATION

Ensure password reset forms validate the users old password before changing the users password

### 4.13.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/resources/password_reset

The password reset form found at the following URL is vulnerable to Cross Site Request Forgery (CSRF):

http://target.appcheck-ng.com:8686/resources/password_reset

## 4.14. PASSWORD SUBMITTED VIA THE GET METHOD

**CVSS Score:** 5.1 **CVSS Vector:** AV:N/AC:H/Au:N/C:P/I:P/A:P **Impact/Probability:** Medium/Low

**Affected:** http://target.appcheck-ng.com:8686

A form within the affected page appears to transmit user passwords within the query string (using the GET method). Sensitive data within URLs may be logged at a number locations, including the web server, user's browser and any proxy servers between the client and the server.

Passwords included within the URL could also be disclosed via the HTTP Referer header when any off-site link is followed.

### 4.14.1. REMEDIATION

All user credentials including passwords and session tokens should be transmitted using the POST method.

### 4.14.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/resources/user_creds_via_get

The authentication form at found at the following URL transmits user credentials within the URL:

http://target.appcheck-ng.com:8686/resources/user_creds_via_get

## 4.15. CLEARTEXT SUBMISSION OF PASSWORD

**CVSS Score:** 5.1 **CVSS Vector:** AV:N/AC:H/Au:N/C:P/I:P/A:P **Impact/Probability:** Medium/Low

**Affected:** http://blog.acmeplc.local

Passwords submitted over an unencrypted connection are vulnerable to interception by an attacker who is suitably positioned on the network. This includes the user's own network, within their ISP and within the application's hosting infrastructure.

### 4.15.1. REMEDIATION

The application should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed.

### 4.15.2. TECHNICAL ANALYSIS

#### Example: http://blog.acmeplc.local

The following forms submit authentication credentials in clear text:

- http://blog.acmeplc.local/wordpress/wp-login.php?ModPagespeed=noscript

## 4.16. OPEN REDIRECT VULNERABILITY

**CVSS Score:** 5.1 **CVSS Vector:** AV:N/AC:H/Au:N/C:P/I:P/A:P **Impact/Probability:** Medium/Medium

**Affected:** http://target.appcheck-ng.com:8686

A URL Redirect vulnerability was discovered. It is possible to craft a link to the affected component to redirect the user to a malicious website. An attacker could exploit this vulnerability to trick users into accessing malicious content on the basis that they trust the affected application

### 4.16.1. REMEDIATION

Amend application code to ensure the affected component only permits redirecting to trusted sites To gain a better understanding of Open Redirect issues and to learn different ways to prevent the problem see the following resources:

http://www.owasp.org/index.php/Open_redirect

### 4.16.2. TECHNICAL OVERVIEW

Vulnerable parameters:

| App | URI | Parameter |
|---|---|---|
| http://target.appcheck-ng.com:8686 | /resources/redirect_and_validate?url=http://www.google.com | vuln |

### 4.16.3. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/resources/redirect_and_validate?url=http://www.google.com [param: url]

The www.Appcheck-NG.com domain was supplied along with a legitimate domain (www.google.com) seperated by an @ symbol.

The payload: `http://www.Appcheck-NG.com@www.google.com` triggered a redirect to: http://www.Appcheck-NG.com/www.google.com

## 4.17. JSONP RESOURCE DETECTED

**CVSS Score:** 2.6 **CVSS Vector:** AV:N/AC:H/Au:N/C:N/I:N/A:P **Impact/Probability:** Low/Low

**Affected:** http://target.appcheck-ng.com:8686

JSONp is a method of returning data cross domain. If sensitive information is returned via JSONp, it may be possible to read this data via a CSRF attack.

### 4.17.1. REMEDIATION

If user specific data is returned by any of the affected URLs, ensure that adequate Cross Site Request forgery protection is used.

### 4.17.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686

**Vulnerable URL Examples**

http://target.appcheck-ng.com:8686/resources/gosJSONp?callback=foo&data=goo

## 4.18. HTML5 POSTMESSAGE OBSERVED

**CVSS Score:** 2.6 **CVSS Vector:** AV:N/AC:H/Au:N/C:N/I:N/A:P **Impact/Probability:** Low/Low

**Affected:** http://target.appcheck-ng.com:8686

Cross-origin messaging was observed, which should be checked manually for vulnerabilities such as cross-site scripting.

### 4.18.1. REMEDIATION

Ensure input from incoming client-side messages are sanitised in the message handler before use. For example, the message origin should be verified.

### 4.18.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686

The following information gives details on observed PostMessage messages for each distinct handler detected in the web application.

**Handler 1**

**Message handler code**:

```
function receiveMessage(event) {
   log("Message Origin:" + event.origin);
   log(event.data);
   window.location = event.data.vuln;
}
```

**Observed messages**:

http://target.appcheck-ng.com:8686 -> http://target.appcheck-ng.com:8686/html5/post_message_1_child:

```
{"greeting":"hello","vuln":"/admin","something":{"blah":[1,2,3]}}
```

## 4.19. DETECTED USER AGENT ADAPTATION

**CVSS Score:** 2.6 **CVSS Vector:** AV:N/AC:H/Au:N/C:N/I:N/A:P **Impact/Probability:** Low/Low

**Affected:** http://blog.acmeplc.local, http://target.appcheck-ng.com:8686

Areas within the scanned application(s) were found to adapt content for client devices based on User Agent strings.

This behaviour is interesting to the scanner since web applications may present content to particular devices using a slightly or entirely different code base, therefore extending the attack surface.

### 4.19.1. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/

Content adaptations were discovered by probing the web application under emulation of a range of devices.

**User Agents**

The following UserAgent strings were found to cause content adaptation:

**Safari on iPhone**

Mozilla/5.0 (iPhone; CPU iPhone OS 5_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3

**Content Adaptation**

The following urls were found to adapt to UserAgent strings:

- http://target.appcheck-ng.com:8686/device_adaption/user_agent_1

#### Example: http://blog.acmeplc.local/wordpress//

Content adaptations were discovered by probing the web application under emulation of a range of devices.

**User Agents**

The following UserAgent strings were found to cause content adaptation:

**Bingbot**

Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)

**Content Adaptation**

The following urls were found to adapt to UserAgent strings:

- http://blog.acmeplc.local/wordpress
- http://blog.acmeplc.local/wordpress/
- http://blog.acmeplc.local/wordpress/2013/
- http://blog.acmeplc.local/wordpress/2013/12/
- http://blog.acmeplc.local/wordpress/2013/12/14/
- http://blog.acmeplc.local/wordpress/2013/12/14/hello-world/
- http://blog.acmeplc.local/wordpress/category/uncategorized/
- http://blog.acmeplc.local/wordpress/sample-page/

## 4.20. SESSION TOKEN OBSERVED IN URL

**CVSS Score:** 2.6 **CVSS Vector:** AV:N/AC:H/Au:N/C:N/I:N/A:P **Impact/Probability:** Low/Low

**Affected:** http://target.appcheck-ng.com:8686

Session tokens appear to passed within URLs and may be stored in various locations, including the user's web browser, the server and in any forward or reverse proxies. If an attacker can recover the token they may be able to use the token to interact with the application as that user.

### 4.20.1. REMEDIATION

The application should use an alternative mechanism for transmitting session tokens, such as HTTP cookies or hidden fields in forms that are submitted using the POST method.

### 4.20.2. TECHNICAL ANALYSIS

#### Example: http://target.appcheck-ng.com:8686/

**Observed URLs**

http://target.appcheck-ng.com:8686/session_setup/token?SESSION=F27ED2A6AAE4C6DA409A3044E79B8B48

## 4.21. A WORDPRESS INSTALLATION WAS DETECTED

**CVSS Score:** 2.6 **CVSS Vector:** AV:N/AC:H/Au:N/C:N/I:N/A:P **Impact/Probability:** Low/Low

**Affected:** http://blog.acmeplc.local

A Wordpress installation was detected.

### 4.21.1. TECHNICAL ANALYSIS

#### Example: http://blog.acmeplc.local/wordpress/

A Wordpress installation was detected at the following URI:

http://blog.acmeplc.local/wordpress/

**WordPress Version:** 3.8.3

**Login Page:** http://blog.acmeplc.local/wordpress/wp-login.php

**Enumerated Usernames**

The following potential Wordpress usernames were enumerated:

- test-user2 / test.user2 / test user2 (test-user2)
- test-user / test.user / test user (test-user)
- user (user)

**Pentest Note:** Dash characters *(-)* within enumerated usernames may actually be dots or spaces within the real username.for example, 'john-smith' may actually be 'john.smith' or 'john smith', this is due to the way usernames are converted to be URL friendly values by wordpress. All varients should be attempted when performing dictionary attacks against the application

# 5. APPENDIX A: WEB APPLICATION DEFENSIVE STRATEGIES

## 5.1. VALIDATING INPUT

Input validation is an important defence strategy in preventing SQL injection, code injection, cross site scripting and a host of other vulnerabilities. The aim of input validation is to ensure that data processed by the application contains only known good characters and that all others are discarded.

There are two high approaches that can be adopted to achieve this aim: white listing and black listing. It is widely accepted that the white list approach is the most secure option. However, where possible a two tiered approach to input validation is beneficial.

**Black Listing: Reject Known Bad**

**Note:** Black listing should not be used in isolation but as a complementary measure to white listing.Data that successfully passes the black list filter should then be validated using the white list approach.

Black listing is the process of rejecting known bad input. This technique receives a lot of criticism from security purists because it is impossible to know all dangerous character sequences ahead of time. Whilst this is true, a black list approach can be a good first line of defence against attempts to discover and exploit vulnerabilities within the application. Any user violates the black list filter should have any active sessions invalidated (logged out) and the event should be logged. In some circumstances an alerting process may be appropriate, depending on the sensitivity of the chosen black list and the expected volume of false positives.

- Create a black list containing characters and character sequences which are associated with the attack or attacks.
- Decode all input until no further decoding is possible before comparing against the black list.
- Users who violate the filter should have their session invalidated and a logging and/or alerting process should be invoked.
- Black listing should only be used as a complementary measure to white listing. Data that successfully passes the black list filter should then be passed to the white list process.

**White Listing: Allow Known Good**

White listing is the process of validating input to ensure it contains only known good (safe) characters and is constructed in an expected manner. This approach is preferred over black listing since it does not require that the developer know all potentially dangerous characters ahead of time. A white list filter should be context dependant and be as restrictive as possible. Potential features to validate include:

- Data contains only permitted, safe characters.
- The data is of a certain length (e.g. falls between a minimum and maximum length).
- The data matches a defined regular expression (e.g. to validate the structure of an email address).

Care should be taken to ensure the white list is sufficiently restrictive to be secure whilst allowing the user to include all of the characters he/she needs to interact with the application.

**Sanitising (Converting) Output**

Whenever data that originated from a user or an out of bound channel is copied into a servers' response it should be HTML encoded to sanitise potentially dangerous characters. HTML encoding is the process of converting characters to their HTML equivalents. The converted character appears the same as the original character when viewed in a web browser but does not affect the structure of the HTML document.

The recommended approach is to HTML encode all non-alphanumeric characters to ensure that all special characters that could be useful in constructing a malicious script are converted.

The following characters should always be converted to their inert HTML equivalents before being included within the page:

| Character | HTML Equivalent |
|---|---|
| “ | &quote; |
| ‘ | &apos; |
| & | &amp; |
| < | &lt; |
| > | &gt; |

Additional characters can be converted to HTML equivalents using their character code in decimal prefixed with&# and terminated with a semicolonas follows:

| Character | HTML Equivalent |
|---|---|
| ; | &#59; |
| + | &#43; |

**XSS: Remove Unnecessary Exploit Vectors.**

Performing input validation and output sanitisation will form a robust defence against reflected and stored cross site scripting vulnerabilities. However there are a number of circumstances where cross site scripting vulnerabilities may still be possible. There are a number of locations where it is inherently dangerous to insert user supplied data.

Where possible user supplied data should not be included within existing scripts. Doing so increases the chances that an input validation filter can be bypassed since the attacker does not need as many special characters such as< and>to construct a malicious script.