Security Assessment

Prepared For

AppCheck Labs

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

Assessment Schedule	
Assessment Performed on	Thursday 13 June 2019
Report Prepared on	Thursday 13 June 2019

1.1. CONTENTS

2. PROJECT OVERVIEW

2.1. SCANNING METHOLOGY OVERVIEW

AppCheck includes two distinct scanning engines designed to test Web Applications and computer systems for vulnerabilities.

Application Scanning

For each URL configured with the scan, AppCheck performs online reconnaissance to gather information pertaining to the site that is publicly available in search engines and other online indexing services.

Next, AppCheck will map out the application using a sophisticated crawling engine. The crawler combines traditional web scraping with a browser-based crawler which implements artificial intelligence to mimic typical application user behaviour.

The "Mapped Attack Surface" enumerated during the initial phases of the scan, is then subject to methodical security testing. Typically, the assessment process works by taking each user supplied data component, such as a form field of query string parameter, then modifies it to include a specific test case before submitting it to the server. Based on the applications response, further test cases are then submitted through the same method to confirm the vulnerability.

Common vulnerabilities detected during the web application scan include; Injection flaws such as *SQL*, *NoSQL*, XML, *Code*, and *Command* injection, Cross-Site Scripting and hundreds of other vulnerability classes arising from insecure code.

Infrastructure & Platform Scanning

In this context, *Infrastructure* includes all components that are not covered within the application scanning phase. The infrastructure scan beings by port scanning each host to identified accessible services. Each service is then probed for vulnerabilities such as missing security patches, configuration weaknesses and information disclosure vulnerabilities.

Common vulnerabilities detected during the infrastructure scanning phase include; missing operating system patches, weak administrative passwords and access control vulnerabilities.

If the target system is hosted within Amazon Web Services, Google Cloud or Azure, specific configuration assessment modules are launched to identify common configuration weaknesses.

2.2. SCAN SCOPE

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

Application URLs / IP Addresses
http://shop.appcheck.com/
http://catalog.appcheck.com/

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.

Vulnerability Summary

Impact	Probability High	Probability Medium	Probability Low
High	5	5	
Medium	1	1	1
Low			1
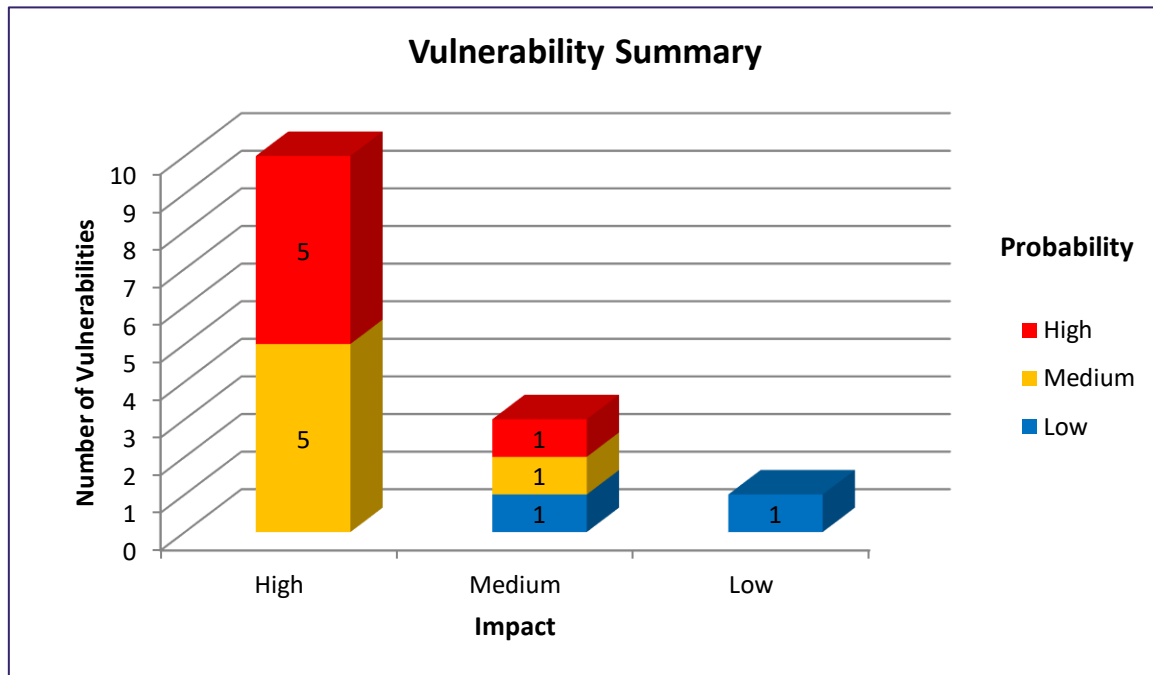
3.2. HIGH IMPACT VULNERABILITIES

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: 9.4 **Impact/Prob:** **High**/**High**	**SQL Injection** SQL Injection Vulnerabilities were discovered within the listed resources and applications. SQL injection vulnerabilities occur when client supplied data is included within an SQL Query without sanitisation. This vulnerability could be exploited by a malicious attacker to read, modify, delete or create SQL table data. In many cases it also possible to exploit features of SQL server to execute system commands and/or access the local file system. The following SQL Injection variants were detected: *Content Variance SQL Injection: Microsoft SQL Server* *Error Based SQL Injection*	http://catalog.appcheck.com
CVSS: 7.6 **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://catalog.appcheck.com
CVSS: 6.2 **Impact/Prob:** **High**/**High**	**User passwords stored in clear-text** User registered account credentials appear to be stored in clear-text or reversible encryption. AppCheck attempts to automatically register an account on the system using an email address that is later received by the Sentinel out-of-band monitoring system. The same email address is used within password recovery forms to determine if the password submitted during registration is sent back to the user. If Sentinel receives the registered password via email, then the application must either store the password in clear-text or using reversible encryption.	http://shop.appcheck.com

Impact / Ref	Description	Affected Hosts
CVSS: 9.4 **Impact/Prob:** **High**/Medium	**AWS: S3 Bucket write permission granted to All AWS users** An Amazon Web Servers (AWS) Simple Storage Service (S3) Bucket was found to be world writable. It is possible for an attacker to upload files to this Bucket that can then be later downloaded by other users. This flaw could be exploited to host malicious content or exploit other systems that rely on files stored within the bucket.	https://s3.amazonaws.com
CVSS: 8.8 **Impact/Prob:** **High**/Medium	**Possible External Service Takeover: Amazon S3 via Cloudfront** A resource imported by or linked to the target application resolves to a CNAME record for a cloud-based service that has been incorrectly registered or has been deleted. It's possible for an attacker to register this service and host malicious content which would then be automatically linked to or imported by your application.	http://shop.appcheck.com
CVSS: 6.4 **Impact/Prob:** **High**/**High**	**Sentinel: Password Reset Buffer Truncation Vulnerability** A potential buffer truncation vulnerability was discovered within a password reset field. It may be possible to reset another user's password by exploiting this flaw.	http://shop.appcheck.com
CVSS: 5.0 **Impact/Prob:** **High**/Medium	**Source Code Disclosure via Path Manipulation** Path Traversal Vulnerabilities occur when server-side scripts process file paths supplied by client without correctly validating the file path. This check determines if restricted files can be accessed by manipulating client supplied input to include path traversal sequences such as ../ and ..\	http://catalog.appcheck.com
CVSS: 4.3 **Impact/Prob:** **High**/Medium	**Reflected Cross-site Scripting (Exploited)** Cross Site scripting vulnerabilities were discovered. A malicious attacker could exploit these flaws to perform a social engineering attack against users of the affected application.	http://shop.appcheck.com

Impact / Ref	Description	Affected Hosts
CVSS: 4.3 **Impact/Prob:** **High**/**Medium**	**Cross-Site Scripting via postMessage (process_message_event)** A postMessage handler function attached to the affected page implements a code pattern that is vulnerable to Cross-Site Scripting (XSS). AppCheck detected this potential flaw by tracing Message Events through JavaScript handler functions within the page then providing the flaw via client-side exploitation.	http://shop.appcheck.com
CVSS: 9.4 **Impact/Prob:** **High**/**High**	**Remote Code Execution via Insecure File Upload** AppCheck identified a form that permits server scripts to be uploaded and executed on the server. A malicious attacker could exploit this flaw to execute system commands on the server.	http://catalog.appcheck.com

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/Low	**Vulnerable JavaScript Libraries Detected** One or more outdated JavaScript libraries were detected. The presence of a vulnerable library does not necessarily mean that the application is vulnerable, however the library could allow a vulnerability to be introduced even when secure code practices are followed.	http://shop.appcheck.com
CVSS: 5.0 **Impact/Prob:** Medium/High	**Padding Oracle** A web application hosted on the remote server is potentially prone to a padding oracle attack.	http://shop.appcheck.com
CVSS: 4.3 **Impact/Prob:** Medium/Medium	**HTTP Header Injection** The listed web application is vulnerable to one or more header injection vulnerabilities. HTTP header injection vulnerabilities occur when user-supplied data is copied into a response header in an unsafe way. If an attacker can inject newline characters into the header, then they can insert new malicious headers into the servers response. A number of attacks are possible by exploiting this flaw such as HTTP Response Splitting, Session Fixation, URL Redirection and Cross Site Scripting.	http://catalog.appcheck.com

4.   ASSESSMENT RESULTS

4.1.   CONTENT VARIANCE SQL INJECTION: MICROSOFT SQL SERVER

CVSS Score: 9.4   **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:C/A:N   **Impact/Probability: High/High**

Affected: http://catalog.appcheck.com

SQL injection vulnerabilities occur when client supplied data is included within an SQL Query without sanitisation. This vulnerability could be exploited by a malicious attacker to read, modify, delete or create SQL table data. In many cases it also possible to exploit features of SQL server to execute system commands and/or access the local file system.

4.1.1.   REMEDIATION

The most effective way to prevent SQL injection attacks is to use prepared statements, also known as paramaterized or binded queries. This method separates out the structure of the query from the data therefore preventing the query from being manipulated in an unsafe way.

You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries.

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 methods of preventing the problem see the following resources:

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

4.1.2.   TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://catalog.appcheck.com	/product_detail.aspx	payload.product_id

4.1.3.   TECHNICAL ANALYSIS

Affected Component: http://catalog.appcheck.com/product_detail.aspx [param: payload.product_id]

The SQL Injection flaw was identified by injecting multiple SQL statements designed to return different application responses. This test attempts to identify the backend platform by injecting SQL statements that will only successfully execute on a specific database or platform, in this case; "Microsoft SQL Server".

Detection Payloads

The following payload evaluates to *True* when executed by the database server:

```
1 AND 292-(SElEcT @@PACK_RECEIVED-@@PACK_RECEIVED)-249=292-(sElEcT @@PACK_RECEIVED-@@PACK_RECEIVED)-249 OR 1234=4321
```

Conversely, the following payload evaluates to *False*:

```
1 AND 160-(SElEcT @@PACK_RECEIVED-@@PACK_RECEIVED)-109=986-(sElEcT @@PACK_RECEIVED-@@PACK_RECEIVED)-934 OR 1234=4321
```

By sampling the applications response under True/False injections, AppCheck identified a string that is returned when the statement is *True* but is never returned when the statement is *False*.

String Matching:

The following string is returned when a SQL statement is injected that evaluates to True:

```
<td height="20" colspan="5" align="right" bgcolor="#f7f7f7" class="basketmiddle">Surplus delivery charges for Europe or Rest of the World will be calculated on the Checkout page</td>
```

4.2.   ERROR BASED SQL INJECTION

CVSS Score: 9.4   **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:C/A:N   **Impact/Probability: High/High**

Affected: http://catalog.appcheck.com

SQL injection vulnerabilities occur when client supplied data is included within an SQL Query without sanitisation. This vulnerability could be exploited by a malicious attacker to read, modify, delete or create SQL table data. In many cases it also possible to exploit features of SQL server to execute system commands and/or access the local file system.

4.2.1.   REMEDIATION

The most effective way to prevent SQL injection attacks is to use prepared statements, also known as parameterized or binded queries. This method separates out the structure of the query from the data therefore preventing the query from being manipulated in an unsafe way.

You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries.

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 methods of preventing the problem see the following resources:

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

4.2.2.   TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://catalog.appcheck.com	/product_list.aspx?cat1=1	cat1

4.2.3.   TECHNICAL ANALYSIS

http://catalog.appcheck.com/product_list.aspx?cat1=1 [param: cat1]

Technical Details

A SQL Injection vulnerability was detected by injecting a payload designed to return a specific token within application error messages. Each injected payload is designed to only return the token when executed by a specific SQL server, in this case "Microsoft SQL Server".

Injected Payload:

1 AND 1234=convert(int,char(0x33)+char(0x5f)+char(0x31)+char(0x5f)+char(0x33)+char(0x5f)+char(0x33)+char(0x5f)+char(0x37)) OR 1234=4321

Token returned by the server:

3_1_3_3_7

4.2.4.   PROOF OF CONCEPT

To provide proof of concept, the SQL Injection flaw was exploited to extract a list of databases and table names from the database server.

List of databases:	List of tables in current database:
- master - master1 - MF20 - model - msdb - securesec - tempdb - users	- sqlmapoutput - tblAdmin - tblBrands - tblCats - tblEnquiries - tblMailing - tblMembers - tblMembersAddress - tblProductsOptions - tblProductsReviews - tblTestimonials - tblVouchers

4.3. HIGHLY PRIVILEGED SQL USER ACCOUNT DETECTED

CVSS Score: 7.6   **CVSS Vector:** AV:N/AC:H/Au:N/C:C/I:C/A:C   **Impact/Probability: High/High**

Affected: http://catalog.appcheck.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.

4.3.1. REMEDIATION

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

4.3.2. TECHNICAL ANALYSIS

Example: http://catalog.appcheck.com

Technical Summary

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

4.4. USER PASSWORDS STORED IN CLEAR-TEXT

CVSS Score: 6.2   **CVSS Vector:** AV:L/AC:L/Au:S/C:C/I:C/A:N   **Impact/Probability: High/High**

Affected: http://shop.appcheck.com

AppCheck attempts to automatically register an account on the system using an email address that is later received by the Sentinel out-of-band monitoring system. The same email address is used within password recovery forms to determine if the password submitted during registration is sent back to the user.

If Sentinel receives the registered password via email, then the application must either store the password in clear-text or using reversible encryption. This presents significant vulnerability since any attacker who is able to gain access to the user database and/or application is likely to also have access to user passwords.

4.4.1. REMEDIATION

User passwords should never be stored in clear-text, instead, a hash of the users password should be used.

- A strong password hashing algorithm should be used such as SHA-512 to help defend against offline attacks.
- Each password should be combined with a unique Salt value before hashing takes place to defend against

cryptanalysis attacks

See the following resource for further recommendations:

https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

4.4.2. TECHNICAL ANALYSIS

Example: http://shop.appcheck.com/self_signup/reset_password

Technical Details

A user account was registered via **http://shop.appcheck.com/self_signup/register** with the email address **owxgh0ag49@sentinel.pentesting.us** and a password of **AppCh3ck!9fOCt**.

The password reset feature was invoked via **http://shop.appcheck.com/self_signup/reset_password** returning the following email which appears to contain the clear text password:

```
To complete registration please follow the URL: http://shop.appcheck.com/self_signup/activate?code=zCUarZfJmR

Your account password is AppCh3ck!9fOCt
```

4.5. AWS: S3 BUCKET WRITE PERMISSION GRANTED TO ALL AWS USERS

CVSS Score: 9.4   **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:C/A:N   **Impact/Probability: High/Medium**

Affected: https://s3.amazonaws.com

An Amazon Web Servers (AWS) Simple Storage Service (S3) Bucket was found to be world writable. It is possible for an attacker to upload files to this Bucket that can then be later downloaded by other users. This flaw could be exploited to host malicious content or exploit other systems that rely on files stored within the bucket.

4.5.1. REMEDIATION

Review permissions on the affected S3 Bucket

4.5.2. TECHNICAL ANALYSIS

It is possible to write to the Amazon Web Services S3 bucket: https://s3.amazonaws.com/appcheck-news/ as any AWS user. The S3 bucket was referenced via the following pages on the scanned application:

http://shop.appcheck.com/product_news/

4.5.3. PROOF OF CONCEPT

This flaw was detected by uploading the file: https://s3.amazonaws.com/appcheck-news/bqrfkqfssu.txt using the Amazon S3 API configured with AppCheck credentials (which should have not explicit permissions set).

This file can then be downloaded by other users using the API or using a standard web browser. This flaw is typically introduced when the 'AuthenticatedUsers' group is permitted write access to the bucket

4.6. POSSIBLE EXTERNAL SERVICE TAKEOVER: AMAZON S3 VIA CLOUDFRONT

CVSS Score: 8.8   **CVSS Vector:** AV:N/AC:M/Au:N/C:C/I:N/A:C   **Impact/Probability: High/Medium**

Affected: http://shop.appcheck.com

A resource imported by or linked to the target application resolves to a CNAME record for a cloud-based service that has been incorrectly registered or has been deleted. It's possible for an attacker to register this service and host malicious content which would then be automatically linked to or imported by your application.

Subdomain Takeover Background

A 'Subdomain Takeover' vulnerability typically arises when a resource such as a JavaScript is loaded from an external domain which becomes publicly available (often unexpectedly) and can be hijacked by another party. There are two different classes of Subdomain Takeover, each with a different underlying cause. The following scenarios describe the most common cases:

CNAME / DNS Takeover

A Web site with a domain name of appcheck.com uses a CNAME record to another domain, for example:

        static.appcheck.com   CNAME   cndserver.otherdomain.com

At some point in time, otherdomain.com expires and is available for registration by anyone. Since the CNAME record is not removed from then appcheck.com DNS zone, anyone who registers otherdomain.com has full control over shop.appcheck.com until the DNS record is updated.

External Service / CDN Takeover

Modern web applications often make use of Content Delivery Networks (CDN) and third-party service provides in order to deploy static content, integrate services, improve bandwidth efficiency and provide protection against DDoS attacks. Since the CDN or service resides on a different domain, most providers offer the ability to delegate via CNAME. For example, consider a website with the address www.appcheck.com has an integrated shop provided by Shopify at https://appcheckshop.myshopify.com. To allow users to deploy services with a friendlier recognisable name, Shopify (and many others) allow their services to be accessed by creating a CNAME record that points to the hosted service. For example:

        shop.appcheck.com   CNAME   appcheckshop.myshopify.com

Now the user is able to access the resource at https://shop.appcheck.com which Shopify is configured to recognise as the shop configured at appcheckshop.myshopify.com.

A vulnerability can arise is the hosted service expires, is deleted or was not correctly configured correctly allowing a malicious attacker to register the service and hijack the domain.

4.6.1.   REMEDIATION

Review the affected component to determine if the linked resource has expired, had its configuration changed or being deleted. Typically, this flaw is resolved by addressing the issue with the third party, registering the affected domain or removing the link to it.

4.6.2.   TECHNICAL ANALYSIS

A URL embedded within the page http://shop.appcheck.com/shoes/ was found to resolve to a CNAME for a third-party provider **Amazon S3 via Cloudfront**. A request to this resource matches a signature that suggests it is not currently registered and could therefore be hijacked by a malicious attacker.

Parent Page: http://shop.appcheck.com/shoes/

External URL: http://deletedbucket.pentesting.us/file3.js

CNAME:	`deletedbucket.pentesting.us resolves to: d233da7fg936sj.cloudfront.net`

URL Matches

The output below shows references to the affected hostname **deletedbucket.pentesting.us** within the page:

```
...m a thrid party provider (AWS via cloudfront)

    <script src='http://deletedbucket.pentesting.us/file3.js'></script>
    <a href="https://www.appcheck.com">appcheck</a...
```

External provider signatures

The output below shows references to signature "NoSuchBucket" within the services providers response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Error><Code>NoSuchBucket</Code><Message>The specified bucket does not exist</Message><BucketName>de...
```

4.7.   SENTINEL: PASSWORD RESET BUFFER TRUNCATION VULNERABILITY

CVSS Score: 6.4   **CVSS Vector:** AV:N/AC:L/Au:N/C:P/I:P/A:N   **Impact/Probability: High/High**

Affected: http://shop.appcheck.com

A buffer truncation vulnerability was discovered within a password reset field. It may be possible to reset another users password by exploiting this flaw.

AppCheck identified this flaw by targeting a password reset form within the application. An email address discovered during the crawl phase was submitted along with a long string of whitespace characters and a second email address. Upon receipt of an email to the AppCheck analyses the email to determine if contains password reset information.

The flaw occurs when the password reset system truncates the supplied email address value before checking for it in the database. If the original buffer is then used when the password reset email is sent, it is possible to supply additional recipients.

4.7.1.   REMEDIATION

Validate the supplied value to ensure only trusted values are accepted. Alternatively review backend application processing to ensure the email address for the supplied account is retrieved from the database and used as the target for the email.

4.7.2.   TECHNICAL ANALYSIS

http://shop.appcheck.com/sentinel/sentinel_password_reset

Technical Details

AppCheck Submitted an email address found during the crawl phase; **testuser1@appcheck.com** followed by **128** white space characters then a second email address in the format; **;yqdd5PszpP@sentinel.pentesting.us** An email was then received by the AppCheck Sentinel Monitor indicating that the targeted form is vulnerable.

Recieved Email

```
Received: from [172.18.0.11] by 46.17.59.217 with ESMTP ;
 Thu, 13 Jun 2019 14:20:12 +0100

To reset your password please access https://shop.appcheck.com/pwreset.aspx?token=182fa27a52ba1fba1
```

4.8.   SOURCE CODE DISCLOSURE VIA PATH MANIPULATION

CVSS Score: 5.0   CVSS Vector: AV:N/AC:L/Au:N/C:P/I:N/A:N   **Impact/Probability: High**/**Medium**

Affected: http://catalog.appcheck.com

Path Traversal Vulnerabilities occur when a server-side component process file paths supplied by client without correctly validating the file path. This check determines if restricted files can be accessed by manipulating client supplied input to include path traversal sequences such as ../ and ..\ (and encoded variants)

Path traversal vulnerabilities could be exploited to access sensitive configuration information such as user databases and application source code.

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'.
- 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://catalog.appcheck.com	/download.aspx?pdf=Brochure.pdf	pdf

4.8.3.   TECHNICAL ANALYSIS

Source Code Match

```
<%@ LANGUAGE="VBSCRIPT" %>
<% 'If InStr(request.querystring("file"),"../")=0 and InStr(request.querystring("file"),"%2E%2E%2F")=0 then
' ...
```

By submitting the payload **../download.aspx** within the **query.pdf** parameter it was possible to disclose the source code of the **download.aspx** server-side script. This typically occurs when the supplied parameter value is passed to a function designed to read local files and return the contents to the user. In this case the target script is submitted in an attempt to disclose its source code. This vulnerability could be exploited to disclose application source code and the contents of configuration files and scripts.

4.9.   REFLECTED CROSS-SITE SCRIPTING (EXPLOITED)

CVSS Score: 4.3   **CVSS Vector:** AV:N/AC:M/Au:N/C:N/I:P/A:N   **Impact/Probability: High/Medium**

Affected: http://shop.appcheck.com

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 could by exploited to:

- Read user session cookies and submit them to the attacker. The attacker can then hijack the users session with the application.
- Access sensitive information stored within the body of the page such as HTML forms (or the entire page). The attacker could exploit this to read data protected by the Same Origin policy.
- Perform "Onsite Request forgery". Since JavaScript executes within the context of the victim user it is possible to perform any action the user can perform. The attacker could exploit XSS flaws to invoke dangerous functions such as "transfer funds".
- Deploy exploit frameworks (e.g. BeEF, XSSShell, XSS Harvest) to conduct maintain control of the users session even if the user browses away from the affected page.
- Redirect the user to a malicious website.
- Deface the application.

4.9.1.   REMEDIATION

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

- Validate data against a white list of known good data
- Perform white-list based data sanitisation of output
- Apply validation in a context specific way, where possible avoiding inherently dangerous DOM areas such as JavaScript event handlers, existing scripts and attribute names.

To Prevent XSS attacks a multi-layered approach is recommended. One of the key challenges when defending against XSS vulnerabilities is ensuring the defence matches the context of the data being rendered within the page (DOM). For example, HTML Entity encoding tainted data may succeed when the data rendered directly within the page body but would fail when the data is embedded within a JavaScript event handler.

As a first line of defence, it is recommended that all input originating from an untrusted source should be validated against "known good". This not only helps prevent XSS but also other web application vulnerabilities.

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 be as strict as possible. Ensure that data is canonicalised / decoded before being compared to the filter (in line with the way data is used by the application when it is rendered).

Sanitize data based on a whitelist before embedding in the DOM (page)

Before rendering tainted data, it should be sanitized to ensure it cannot inject script code or otherwise manipulate the page in a dangerous way. Due to the different ways browsers will interpret different parts of the DOM, the sanitisation should be specific to the context where it is used.

A common approach to data sanitisation involves defining a list of safe characters such as *0-9, A-Z,a-z* then HTML Entity encoding or replacing any character that does not fall within the allowed range. HTML Entity encoding is a method of representing any character in a way that will be displayed when rendered but not interpreted as markup. Both named entities and decimal/hexadecimal values can be used.

For example, HTML entity encoding and opening tag character < could by represented as < (less than) or < to use its hexadecimal notation (0x3c translates to <). The key benefit to this approach is that it allows the user of the application to input a range of characters and have them rendered as intended. However the attacker would be prevented from injecting HTML tags as part of a Cross-Site Scripting attack.

It is however important to understand the context in which the sanitised data will be used or processed to ensure further decoding will not allow attacks against the application. For example, if user input is embedded within an existing JavaScript event handler (e.g. onclick="userinput"), the browser will decode HTML entities before use and therefore sanitisation in this context (using this method) is not effective.

The following describes some common areas of the DOM in which data originating from the user may be rendered along with some guidance on the encoding strategy to use:

Page body: Data rendered within the page body, e.g. within the payload of a <div> or anywhere within the page outside of an attribute string or script should be white list sanitised to HTML entity encode characters that are not within the range Aa-Zz, 0-9. Entity encoding coverts characters that are used to build DOM elements to an alternative that is displayed but not interpreted as markup. E.g < is converted to < and > is converted to >

Attribute String: Data embedded within an attribute value should he sanitised so that it is not possible to break out of the attribute value into another context. For quote delimited strings, quote characters could be escaped to read " for double quotes and ' for single quotes. This will prevent the attacker from breaking out of the attribute value to embed a JavaScript event handler or form a new tag. URL/SRC attributes: Attributes that are used to store URLs, such as "href", "src" and "data" may support the JavaScript protocol hander (javascript:code_to_execute). For these cases it is also important to strictly validate input against known good values to ensure that the attacker cannot embed a JavaScript URL. Note that HTML entity encoding URLS is not effective and javascript:alert(1) would be decoded and executed by the browser.

Script Blocks / Event-Handler attributes: Embedding tainted data within the script context is inherently dangerous and should be avoided if possible. For JavaScript data values, e.g JSON elements, it is essential to ensure data is properly escaped to ensure the attacker cannot terminate the script using </script> (effective even if quotes are escaped). Some JavaScript functions are almost never safe to use with untrusted data such as Eval() and window.setInterval().

4.9.2.   TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://shop.appcheck.com	/get_shipping_info?data=id	id

4.9.3.   TECHNICAL ANALYSIS

http://shop.appcheck.com/get_shipping_info?data=id [param: id]

Accuracy

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

```
&#39;&#45;alert&#40;1&#41;&#45;&#39;
```

Encoding

Standard URL Encoding (IE Compatible)

Technical Summary

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 **'lvsFCqw** added to the protocol handler that would be executed when the hanlder is invoked by the browser:

- <iframe src='javascript:var x=1234lvsFCqw;alert(x);' >

Example Exploit URL

The following URL should trigger a JavaScript `alert(1)` when using the Firefox browser: http://shop.appcheck.com/get_shipping_info?data=id%26%2339%3B%26%2345%3Balert%26%2340%3B1%26%2341%3B%26%2345%3B%26%2339%3B
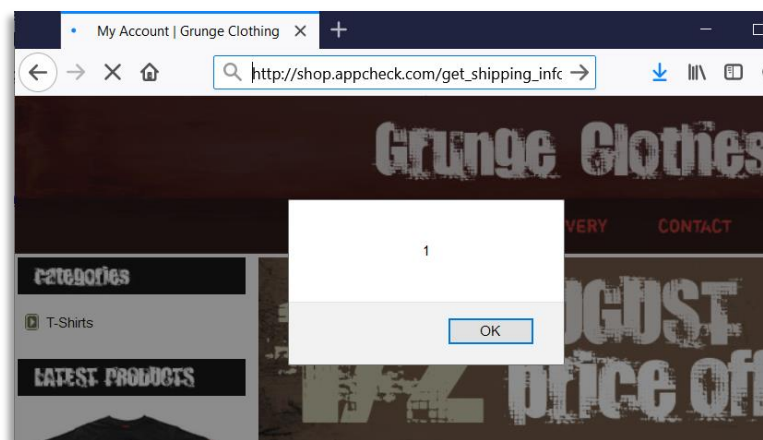
4.10. CROSS-SITE SCRIPTING VIA POSTMESSAGE (PROCESS_MESSAGE_EVENT)

CVSS Score: 4.3   **CVSS Vector:** AV:N/AC:M/Au:N/C:N/I:P/A:N   **Impact/Probability: Medium/Medium**

Affected: http://shop.appcheck.com/facebook

A postMessage handler function attached to the affected page implements a code pattern that is vulnerable to Cross-Site Scripting (XSS). AppCheck detected this potential flaw by tracing Message Events through JavaScript handler functions within the page then providing the flaw via client side exploitation.

Cross-Site Scripting

Cross Site Scripting (XSS) vulnerabilities occur when data submitted to the application is not properly handled before being embedded within the application's Page/DOM.

XSS vulnerabilities could by exploited to:

- Read user session cookies and submit them to the attacker. The attacker can then hijack the users session with the application.
- Access sensitive information stored within the body of the page such as HTML forms (or the entire page). The attacker could exploit this to read data protected by the Same Origin policy.
- Perform "Onsite Request forgery". Since JavaScript executes within the context of the victim user it is possible to perform any action the user can perform. The attacker could exploit XSS flaws to invoke dangerous functions such as "transfer funds".
- Deface the application.

4.10.1.   REMEDIATION

Where possible the origin of postMessage events should be strictly validated to ensure the message originated from a trusted source. When regular expressions are used, care should be taken to ensure wildcard expressions such as "." do not compromise the security of the validation.

For example, a regular expression defined as ^https*://www.appcheck.com$/ would also allow domains such as http://wwwXappcheck.com.

If processing code from any origin is expected behaviour, data should be safely encoded before being written to the DOM in a way that could allow JavaScript to be embedded.

For example:

- Changing the page location based on message payload data.
- If the attacker is able to navigate the page to a *Javascript:* URI it is possible to execute malicious code in the context of the affected domain.
- Updating innerHTML / outerHTML attributes of an element.

4.10.2.   TECHNICAL ANALYSIS

http://shop.appcheck.com/facebook

Technical Details

A postMessage Event Handler named **process_message_event** processed a message event insecurely.

Handler Code

```
function process_message_event(event){
        if(event.origin.indexOf(window.location.hostname) == -1){
            return
        }

        if(event.data.hasOwnProperty("url")){
            redirectTo(event.data.url);
        }

    }
```

Dangerous function triggered at:

```
function redirectTo(url){
        window.open(url);
    }
```

Caller Code:

```
function redirectTo(url){
        window.open(url);
    }.. truncated.. see above for full code.
```

Description: If the attacker can control page navigation though window.open or by setting location.href. It is possible to perform a Cross-Site Scripting attack by supplying a 'JavaScript:' URI

4.10.3.   PROOF OF CONCEPT

The following postMessage payloads triggered JavaScript execution within the affected page:

```
{"url": "javascript:(opener||top).alert(551820052008)//<svg onload=top.alert(551820052008)>https://appcheck-ng.com"}
```

4.11. REMOTE CODE EXECUTION VIA INSECURE FILE UPLOAD

CVSS Score: 9.4   **CVSS Vector:** AV:N/AC:L/Au:N/C:C/I:C/A:N   **Impact/Probability: High/High**

Affected: http://catalog.appcheck.com

The affected application form does not correctly validate the uploaded file type and permits server-side scripts to be uploaded and executed on the server. This flaw could be exploited by the attacker to upload a server-side script designed to allow remote command execution and other dangerous activity (commonly referred to as a "webshell")

4.11.1. REMEDIATION

Strictly validate the uploaded file type to ensure only known good/benign files are permitted. File names should be fully decoded and canonicalized before being validated ensuring path traversal structures such as ../ are not permitted. If possible, files should be stored outside of the web server root or subdirectory to prevent direct access to the file. Directories used to store files should have script execute permissions revoked.

4.11.2. TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://catalog.appcheck.com	/upload_forms/upload_with_csrf	payload.data.uploaded_file

4.11.3. TECHNICAL ANALYSIS

The HTML form found at http://catalog.appcheck.com/upload_forms/upload_with_csrf allows dangerous script files to be uploaded and executed on the server. The vulnerability was identified by uploading a PHP script file designed to echo a dynamically generated string upon successful execution.

Script file name: Appcheck_CPpXu.php

Uploaded File Description: A PHP script designed to return a dynamically generated string (HnYmkOHQVKmKEMcdUtpj) when executed

Script File Contents:

```
<?php  echo chr(72).chr(110).chr(89).chr(109).chr(107).chr(79).chr(72).chr(81).chr(86).chr(75).chr(109).chr(75).chr(69).chr(77).chr(99).chr(100).chr(85).chr(116).chr(112).chr(106); ?>
```

String produced upon successful execution: *HnYmkOHQVKmKEMcdUtpj*

The uploaded script was executed by accessing the following URL: http://catalog.appcheck.com/upload_forms/render_php_file

4.12. VULNERABLE JAVASCRIPT LIBRARIES DETECTED

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

Affected: http://shop.appcheck.com

One or more outdated JavaScript libraries were detected. The presence of a vulnerable library does not necessarily mean that the application is vulnerable, however the library could allow a vulnerability to be introduced even when secure code practices are followed.

In some cases, the library may expose vulnerable components (including unused features) that are readily exploitable. AppCheck has encountered several examples of this in the wild, some of which are analysed within our Hunting PostMessage vulnerabilities whitepaper: https://appcheck-ng.com/hunting-html-5-postmessage-vulnerabilities/

This finding is reported to highlight components that should be updated to prevent vulnerabilities being introduced into otherwise secure code.

4.12.1. REMEDIATION

Upgrade the affected library to the latest release

4.12.2. AFFECTED LIBRARIES

Library	Vulnerability	Detail	Affected Pages
jQuery	jQuery before 3.4.0 Object.prototype pollution (CVE-2019-11358)	AppCheck analysed the jQuery library version by executing it within a local browser engine. The detected version number 3.3.1 appears to be vulnerable to one or more known security flaws. **Library Url:** https://code.jquery.com/jquery-3.3.1.js **Vulnerable below version:** 3.4.0 **References** https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358	http://shop.appcheck.com/

Library	Vulnerability	Detail	Affected Pages
jQuery	XSS with location.hash (CVE-2011-4969)	AppCheck analysed the jQuery library version by executing it within a local browser engine. The detected version number 1.6.1 appears to be vulnerable to one or more known security flaws. **Library Url:** http://webappsim:8686/jquery/jquery.js **Vulnerable below version:** 1.6.3 **References** https://nvd.nist.gov/vuln/detail/CVE-2011-4969 http://research.insecurelabs.org/jquery/test/ https://bugs.jquery.com/ticket/9521	http://shop.appcheck.com/en/
jQuery	Selector interpreted as HTML (CVE-2012-6708)	AppCheck analysed the jQuery library version by executing it within a local browser engine. The detected version number 1.6.1 appears to be vulnerable to one or more known security flaws. **Library Url:** http://webappsim:8686/jQuery/jQuery.js **Vulnerable below version:** 1.9.0b1 **References** http://bugs.jquery.com/ticket/11290 https://nvd.nist.gov/vuln/detail/CVE-2012-6708 http://research.insecurelabs.org/jquery/test/	http://shop.appcheck.com/en/
AngularJS	Outdated angularjs library detected: The attribute usemap can be exploited	AppCheck analysed the angularjs library version by executing it within a local browser engine. The detected version number **1.1.5** appears to be vulnerable to one or more known security flaws. **Library Url:** http://shop.appcheck.com/ria_1/application.js **Below Version:** 1.2.30 **At or above:** 1.0.0 **References** https://github.com/angular/angular.js/blob/master/CHANGELOG.md#1230-patronal-resurrection-2016-07-21	http://shop.appcheck.com/chat

4.13. PADDING ORACLE

CVSS Score: 5.0   **CVSS Vector:** AV:N/AC:L/Au:N/C:P/I:N/A:N   **Impact/Probability: Medium/High**

Affected: http://shop.appcheck.com

A web application hosted on the remote server is potentially prone to a padding oracle attack.

By manipulating the padding on an encrypted string, it was possible to generate an error message that indicates a likely 'padding oracle' vulnerability. Such a vulnerability can affect any application or framework that uses encryption improperly, such as some versions of ASP.net, Java Server Faces, and Mono. An attacker may exploit this issue to decrypt data and recover encryption keys, potentially viewing and modifying confidential data.

4.13.1. REMEDIATION

Update the affected server software, or modify the affected scripts so that they properly validate encrypted data before attempting decryption.

4.13.2. TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://shop.appcheck.com	/userinfo/base64?v1=j2yaGy6J1K063T55dnGNbbQyUExnUlWFzJmxW%2bcApB/35y4zHVVEUz5NE06LfGXg	v1

4.13.3. TECHNICAL ANALYSIS

Technical Summary

The Padding Oracle vulnerability was identified by observing the application's responses to a series of values that had a single bit changed before being returned to the application in the 'query.v1' parameter.

The original base64 encoded value: 'j2yaGy6J1K063T55dnGNbbQyUExnUlWFzJmxW%2bcApB/35y4zHVVEUz5NE06LfGXg'.

The first test flipped the last bit resulting in: 'j2yaGy6J1K063T55dnGNbbQyUExnUlWFzJmxW+cApB/35y4zHVVEUz5NE06LfGXh'.

The second test flipped the first bit resulting in: 'D2yaGy6J1K063T55dnGNbbQyUExnUlWFzJmxW+cApB/35y4zHVVEUz5NE06LfGXg'.

The application's response to the first test had a status code begining with 5 and contained the string 'server error' which was not present in the other responses, this indicates that the Padding Oracle vulnerability is present.

4.14.   HTTP HEADER INJECTION

CVSS Score: 4.3   CVSS Vector: AV:N/AC:M/Au:N/C:N/I:P/A:N   **Impact/Probability: Medium/Medium**

Affected: http://catalog.appcheck.com

The listed web application is vulnerable to a header injection vulnerability. HTTP header injection vulnerabilities occur when user-supplied data is copied into a response header in an unsafe way. If an attacker can inject newline characters into the header, then they can insert new malicious headers into the servers response. A number of attacks are possible by exploiting this flaw such as HTTP Response Splitting, Session Fixation, URL Redirection and Cross Site Scripting.

To gain a better understanding of these issues see the following resources:

- http://www.owasp.org/index.php/HTTP_Response_Splitting
- http://www.owasp.org/index.php/XSS
- http://www.owasp.org/index.php/Session_Fixation
- http://en.wikipedia.org/wiki/HTTP_header_injection

4.14.1.   REMEDIATION

If possible, applications should avoid copying user-controllable data into HTTP response headers. If this is unavoidable, then the data should be strictly validated to prevent header injection attacks. In most situations, it will be appropriate to allow only short alphanumeric strings to be copied into headers, and any other input should be rejected. At a minimum, input containing any characters with ASCII codes less than 0x20 should be rejected.

4.14.2.   TECHNICAL OVERVIEW

Vulnerable parameters:

App	URI	Parameter
http://catalog.appcheck.com	/account_login.aspx?origin=checkout_shipping.aspx	origin

4.14.3. TECHNICAL ANALYSIS

http://catalog.appcheck.com/account_login.aspx?origin=checkout_shipping.aspx [param: origin]

Technical Summary

By inserting the new line characters **%0D%0A** into the parameter value it was possible to inject a new HTTP header named **testheaderxxx** into the servers response

Manipulated Headers

```
Server: Microsoft-IIS/5.0
Date: Mon, 07 Jan 2019 15:31:22 GMT
X-Powered-By: ASP.NET
Connection: close
Location: account.aspx?origin=checkout_shipping.aspx
testheaderxxx: testval1&login=user
Content-Length: 121
Content-Type: text/html
Cache-control: private
```

4.15.   SECURITY HEADERS REPORT

CVSS Score: 0   **CVSS Vector: Informational**   **Impact/Probability: Informational**

Affected: http://catalog.appcheck.com, http://shop.appcheck.com

Several security headers have been introduced in recent years to help mitigate certain attacks against web applications. This finding reports the result of an audit against the most common headers.

See the Technical Details section for a description of audited header.

4.15.1.   TECHNICAL ANALYSIS: HTTP://SHOP.APPCHECK.COM

Header	Description
X-Frame-Options	The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame>, <iframe> or <object>. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites. Available options include; *DENY* preventing all framing, *SAMEORIGIN* to allow framing by sites with the same origin. Some browsers permit *ALLOW-FROM http://www.appcheck.com* to permit framing by a specific URL. However, this is not longer supported by Chrome. In most cases 'X-Frame-Options: SAMEORIGIN' is the recommended option. Alternatively, *Content-Security-Policy* is the preferred option when framing by some sites should be permitted. **Finding:** Partial. The security header was found for some pages but not others. For example, the following pages did not return the security header: • http://shop.appcheck.com/landing/signout.html • http://shop.appcheck.com/accounts/o8/id • http://shop.appcheck.com/sitemap/sitemap_testfile.txt • http://shop.appcheck.com/test.exe • http://shop.appcheck.com/met_virus_sig.exe

Header	Description
X-XSS-Protection	X-XSS-Protection sets the configuration for the cross-site scripting filter built into most browsers. Recommended value 'X-XSS-Protection: 1; mode=block'. **Finding:** Partial. The security header was found for some pages but not others. For example, the following pages did not return the security header: • http://shop.appcheck.com/landing/signout.html • http://shop.appcheck.com/accounts/o8/id • http://shop.appcheck.com/sitemap/sitemap_testfile.txt
X-Content-Type-Options	X-Content-Type-Options stops a browser from trying to MIME-sniff the content type and forces it to stick with the declared content-type. The only valid value for this header is *X-Content-Type-Options: nosniff*. **Finding:** Partial. The security header was found for some pages but not others. For example, the following pages did not return the security header: • http://shop.appcheck.com/landing/signout.html • http://shop.appcheck.com/accounts/o8/id • http://shop.appcheck.com/sitemap/sitemap_testfile.txt
Content-Security-Policy	Content Security Policy is an effective measure to protect your site from XSS attacks. By whitelisting sources of approved content, you can prevent the browser from loading malicious assets. **Finding:** Partial. The security header was found for some pages but not others. For example, the following pages did not return the security header: • http://shop.appcheck.com/landing/signout.html • http://shop.appcheck.com/accounts/o8/id • http://shop.appcheck.com/sitemap/sitemap_testfile.txt • http://shop.appcheck.com/test.exe • http://shop.appcheck.com/met_virus_sig.exe
Referrer-Policy	Referrer Policy is a new header that allows a site to control how much information the browser includes with navigations away from a document and should be set by all sites. **Finding:** Missing. The security header was not found for any pages

4.16.   SUBDOMAIN TAKEOVER AUDIT

CVSS Score: N/A **CVSS Vector: N/A** **Impact/Probability: N/A**

Affected: See table

External resources were audited for Subdomain and Service Take over vulnerabilities. This finding lists the audited domains that were assessed during this scan.

Subdomain Takeover Background

A 'Subdomain Takeover' vulnerability typically arises when a resource such as a JavaScript is loaded from an external domain which becomes publicly available (often unexpectedly) and can be hijacked by another party. There are two different classes of Subdomain Takeover, each with a different underlying cause. The following scenarios describe the most common cases:

CNAME / DNS Takeover

A Web site with a domain name of appcheck.com uses a CNAME record to another domain, for example:

static.appcheck.com   CNAME   cndserver.otherdomain.com

At some point in time, otherdomain.com expires and is available for registration by anyone. Since the CNAME record is not removed from then appcheck.com DNS zone, anyone who registers otherdomain.com has full control over shop.appcheck.com until the DNS record is updated.

External Service / CDN Takeover

Modern web applications often make use of Content Delivery Networks (CDN) and thrid-party service provides in order to deploy static content, integrate services, improve bandwidth efficiency and provide protection against DDoS attacks. Since the CDN or service resides on a different domain, most providers offer the ability to delegate via CNAME. For example, consider a website with the address www.appcheck.com has an integrated shop provided by Shopify at https://appcheckshop.myshopify.com. To allow users to deploy services with a more friendly recognisable name, Shopify (and many others) allow their services to be accessed by creating a CNAME record that points to the hosted service. For example:

shop.appcheck.com   CNAME   appcheckshop.myshopify.com

Now the user is able to access the resource at https://shop.appcheck.com which Shopify is configured to recognise as the shop configured at appcheckshop.myshopify.com.

A vulnerability can arise is the hosted service expires, is deleted or was not correctly configured correctly allowing a malicious attacker to register the service and hijack the domain.

4.16.1.   REMEDIATION

Review the audit output. Vulnerabilities are also reported individually with more verbose technical detail.

4.16.2.　　TECHNICAL ANALYSIS

4.16.3.　　CNAME CHECKS

The table below lists links to external resources that have a CNAME record pointing to a host in another domain. For each resource, the CNAME record is checked to determine if the parent domain is currently registered.

Discovery URL	External Resource	CNAMEs	Vulnerable
http://shop.appcheck.com/shoes/	http://deletedbucket.pentesting.us/file3.js	deletedbucket.pentesting.us -> d233da7fg936sj.cloudfront.net	False
http://shop.appcheck.com/shoes/	https://www.appcheck.com	www.appcheck.com -> appcheck.com	False
http://shop.appcheck.com/shoes/	https://code.jQuery.com/jQuery-3.3.1.js	code.jQuery.com -> cds.s5x3j6q5.hwcdn.net	False

4.16.4.　　EXTERNAL SERVICE CHECKS

Discovery URL	External Resource	Service	Vulnerable
http://shop.appcheck.com/shoes/	http://deletedbucket.pentesting.us/file3.js	Amazon S3 via Cloudfront deletedbucket.pentesting.us -> d233da7fg936sj.cloudfront.net	**True**

4.16.5.　　CHECKED EXTERNAL HOSTNAME DOMAINS (A RECORD & WHOIS)

Discovery URL	External Resource	Parent Domain	Vulnerable
http://shop.appcheck.com/shoes/	http://www.w3.org/1999/xhtml	w3.org	False
http://shop.appcheck.com/shoes/	https://code.jQuery.com/jQuery-3.3.1.js	jQuery.com	False
http://shop.appcheck.com/shoes/	https://www.appcheck.com	appcheck.com	False
http://shop.appcheck.com/shoes/	http://deletedbucket.pentesting.us/file3.js	pentesting.us	False

5. EXTENDED INFORMATION

5.1. PORT SCAN

Please review the following open ports. You should ensure there are no unnecessary ports or services open.

Host	Port	Service	Version
catalog.appcheck.com	80	http	Microsoft IIS/8.0
shop.appcheck.com	80	http	Microsoft IIS/8.0