ONSEC

ExampleCompany Inc.

Information Security Audit Report

February 30, 2024

1 General information

1.1 Project description

This report is presented by "Onsec" (hereinafter - the Contractor) for ExampleCompany Inc. (hereinafter - the Customer) and contains the results of the audit of the platform and related components (hereinafter — the System). The works were carried out in the period from August 17, 2022, to August 26, 2022.

This report contains an expert evaluation of the current level of security of the Customer's web applications, a description of the testing progress with information on all identified vulnerabilities, as well as confirmation of their presence and the result of the exploitation of these vulnerabilities by the Contractor's staff.

1.2 Objectives of the security audit

The objectives of the security audit were to identify the weaknesses of the System’s protection and exploitability of vulnerabilities that can lead to unauthorized access to closed System components.

The main purpose of the attacker, we assume, is unauthorized access to the closed components of the System and critical information stored, processed, and transmitted in the Customer's web application.

The contractor aimed to complete the following tasks while performing the System's security level audit:

- ○ identification of weaknesses in the implementation of applications that can be used to carry out malicious actions against the Customer or Systems's end users;
- ○ analyze the threat level of identified vulnerabilities, search and describe the attack scenarios implementing threats of different levels against the System and/or its end users;
- ○ development of practical recommendations to eliminate the vulnerabilities found
- ○ report on the found vulnerabilities, methods of their exploitation
- ○ development of recommendations for the elimination of vulnerabilities found.

1.3 Model of the Attacker

The Contractor imitated the process of analyzing a web application by an attacker aiming to detect and exploit the System's vulnerabilities. The attacker in this model is supposed to be a highly qualified specialist with skills comparable to those of the Contractor.

The attacker in the used model had the highest level of system privileges:

- access to System source code;
- access to the system with user privileges;
- access to the system with administrator privileges.

1.4 Security Audit Methodology

The methodology is a sequence of actions carried out by the Contractor to perform an analysis of the security level of information resources of the Customer (The System).

It also implies the use of other Contractor's methodologies to analyze the security level of information systems and the application of the following international standards and practices:

- Penetration Testing Execution Standard (PTES);
- Special Publications NIST 800-115 Technical Guide to Information Security Testing and Assessment;
- Open Source Security Testing Methodology Manual (OSSTMM);
- Information Systems Security Assessment Framework (ISSAF);
- Web Application Security Consortium (WASC) Threat Classification;
- Open Web Application Security Project (OWASP) Testing Guide;
- Open Web Application Security Project (OWASP) Code Review Guide;
- Payment Card Industry Data Security Standard (PCI DSS);
- Center for Internet Security (CIS) standards);

While the audit is conducted, both manual checks of possible vulnerabilities and automated checks with special tools are used.

Security analysis is aimed at identifying weaknesses and exploitable vulnerabilities that lead to unauthorized access to private components of applications or increasing privileges in Customer applications.

1.5 The scope of the work and description of the test objects

Before testing, the Customer provided the source data as presented in Table 3.

Table 3 — Source data

Parameter	Value
Public web-interface	examplecomp.com
API	api.examplecomp.com

1.6 Audit conditions

The analysis of the web application security was carried out in accordance with the Contractor’s methodology by simulating the actions of a potential attacker and analyzing the consequences of exploiting the detected vulnerabilities. The detailed security analysis procedure is available on request.

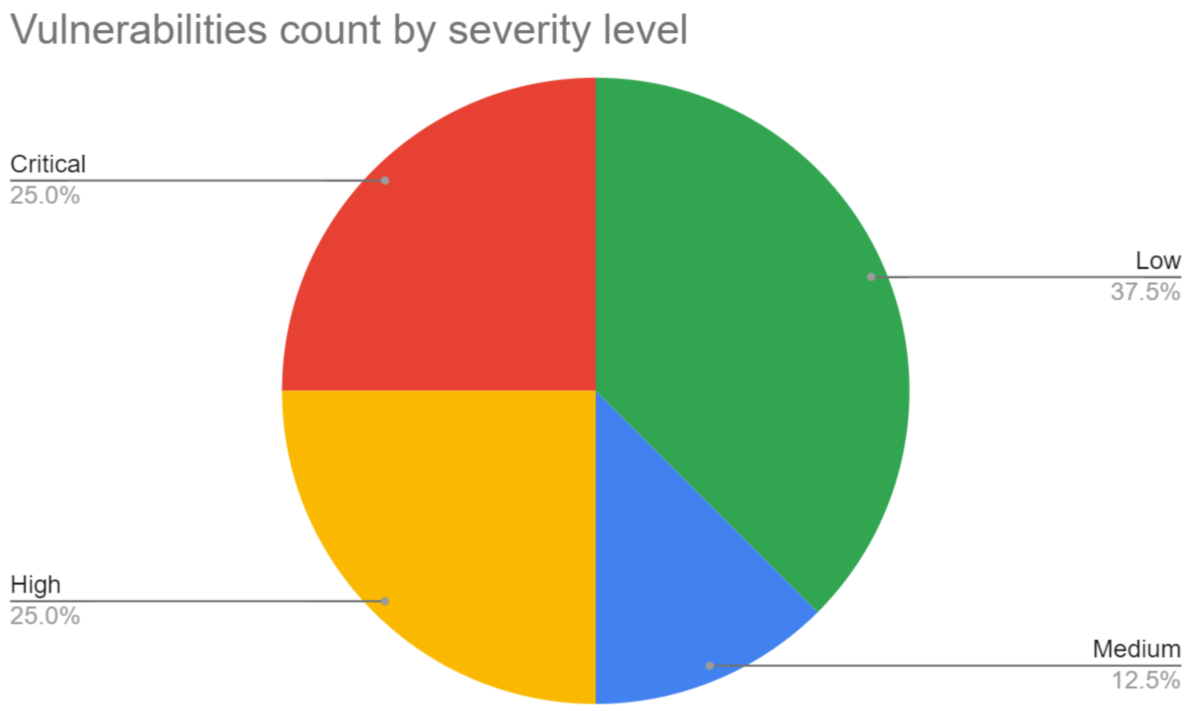
In accordance with the initial information provided, the security analysis was carried out by the “white box " method.

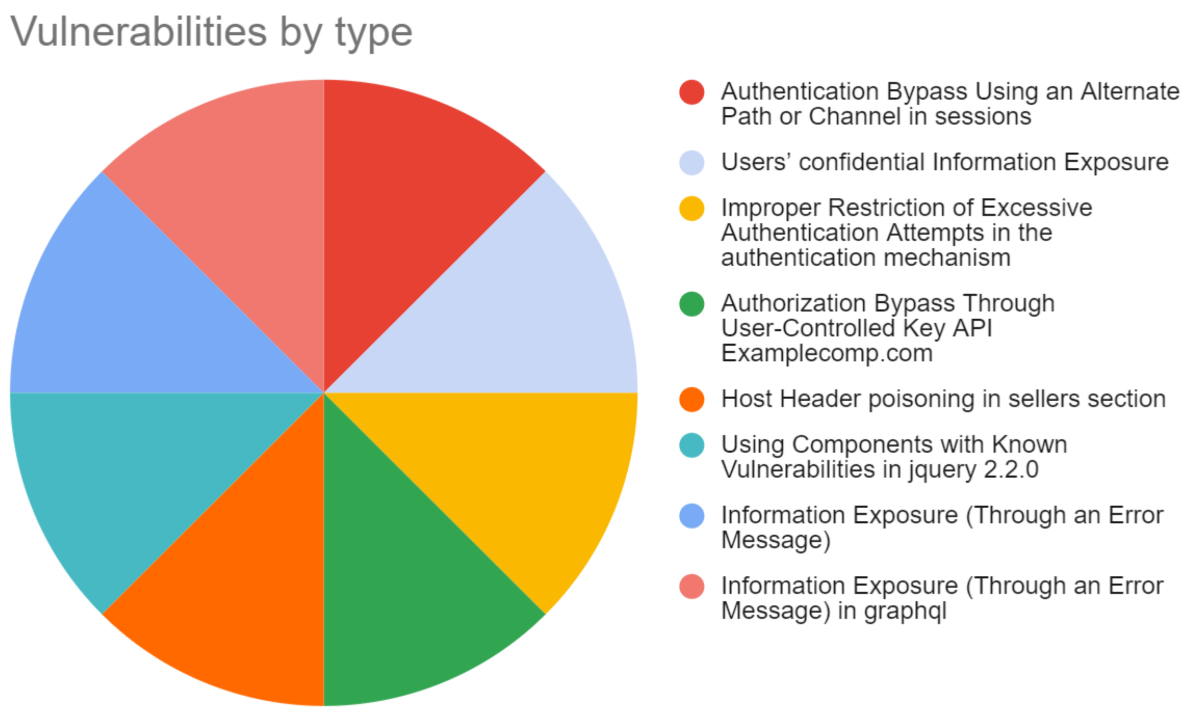
2 Executive summary

2.1. Risk analysis

The overall security level of the audited web services could be evaluated as **Low**. This evaluation is based on the **highest risk of all the vulnerabilities discovered**. Since at least one critical exploitable security issue was detected, which was classified as high risk for business, the overall security level of the audited project should not be evaluated any higher than **Low**.

The Contractor's specialists have discovered **8** vulnerabilities: 2 of critical, 2 of high, 1 of medium, and 3 of low severity level.

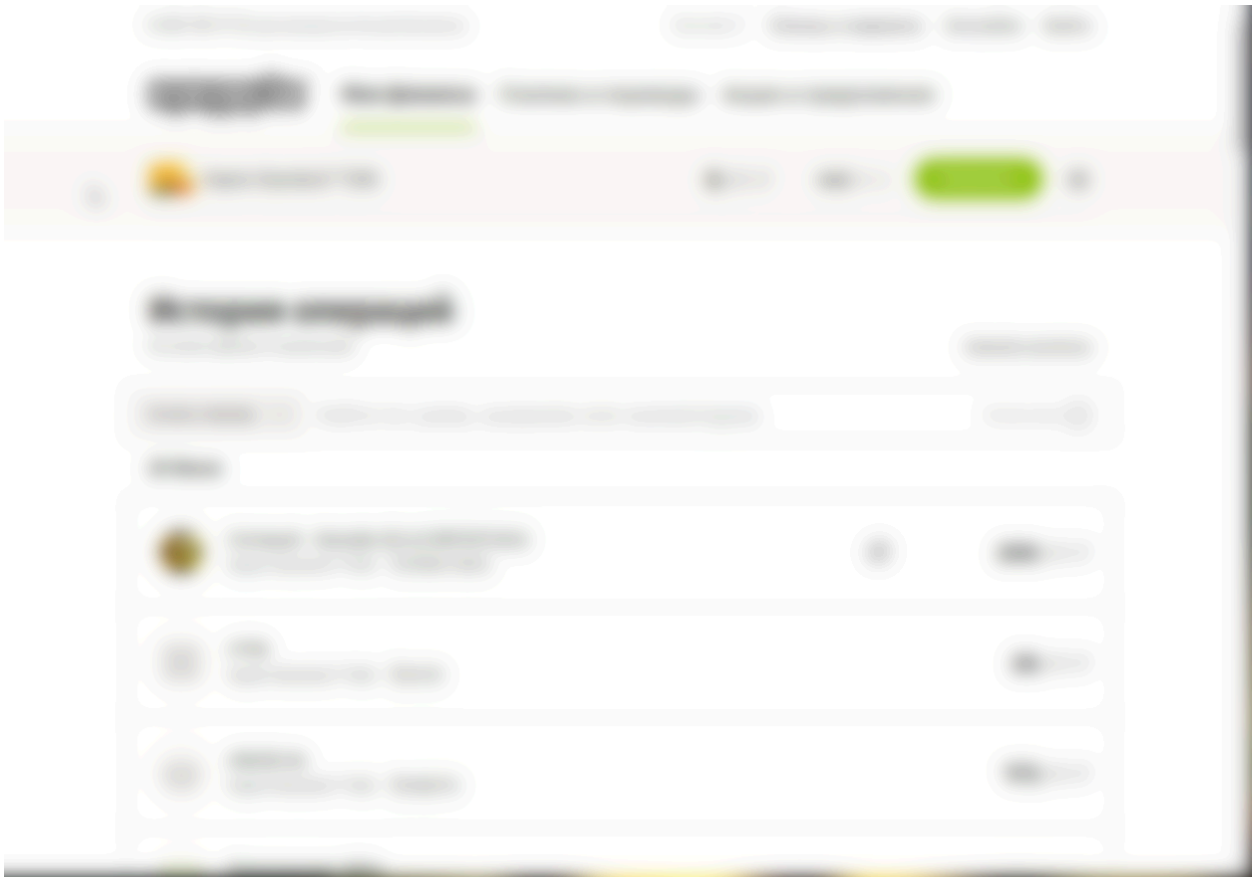




2.2. General results

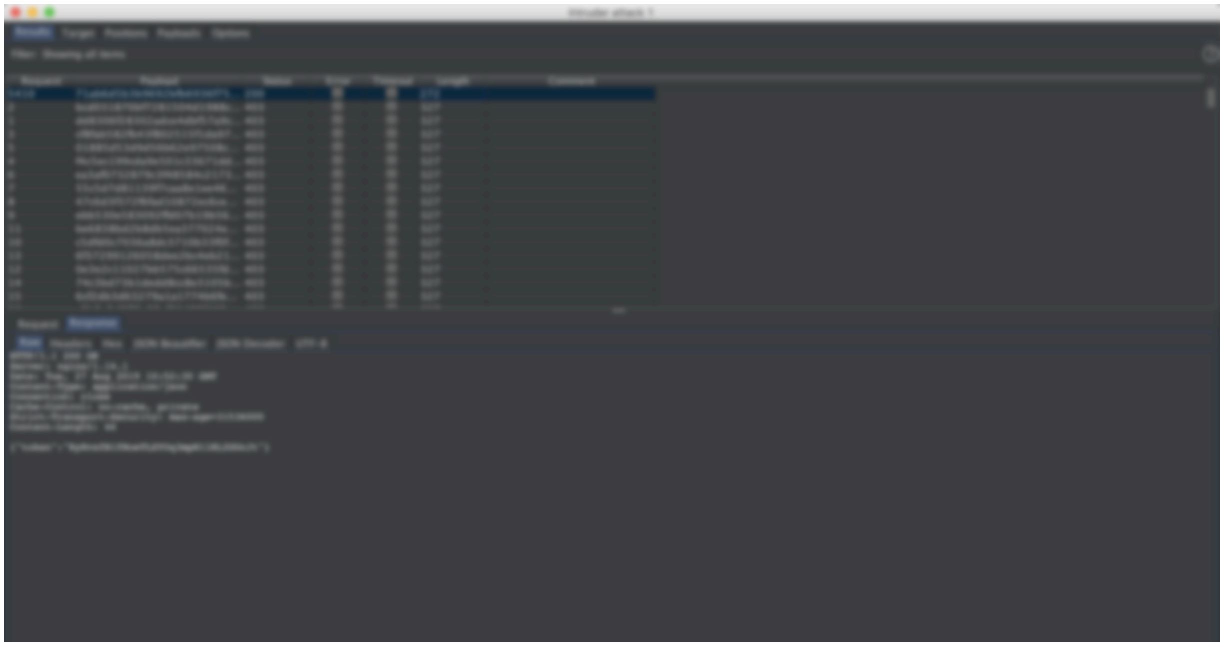
During the audit, some vulnerabilities were discovered. Having exploited those, the Contractor's specialists received unauthorized access:

- full access to user account, including payment functions



- access to the existing users' data (including name, email address, phone number, and account balances. In addition, it has the ability to make changes to the account and perform actions on behalf of the user in the web interface, except for those actions that require confirmation through the second factor.

- user passwords



The full list of the discovered vulnerabilities is represented in Table 4. Detailed descriptions of each vulnerability and recommendations for their elimination are given in Paragraph «Detailed description of the detected vulnerabilities.»

2.3. General recommendations

To minimize the risks of implementing threats to information security and increase the level of security of the Customer’s infrastructure, we **recommend** the following measures:

- apply best practices for secure development by providing SOFTWARE developers with their own or borrowed techniques for writing secure code (For example, OWASP Developer Guide).
- not to reveal debugging information when errors occur during the execution of the application.
- support the latest versions of the software used.to exclude the vulnerabilities found during the analysis.

3 Detailed description of results

This section contains detailed information on the identified vulnerabilities, including recommendations for their remediation, examples of their exploitation, as well as an assessment of the severity of the threat to the infrastructure of the Customer.

Table 4 — List of detected vulnerabilities

Section No.	Name	Resource	Severity level
WLRM-90	Authentication Bypass Using an Alternate Path or Channel in sessions	https://payment.examplecomp.com	Critical
WLRM-186	Users' confidential Information Exposure	https://examplecomp.com	Critical
WLRM-188	Improper Restriction of Excessive Authentication Attempts in the authentication mechanism	https://prod.examplecomp.com	High
WLRM-236	Authorization Bypass Through	api.examplecomp.com	High

	User-Controlled Key API Examplecomp.com		
WLRM-183	Host Header poisoning in sellers section	https://examplecomp.com	Medium
WLRM-172	Using Components with Known Vulnerabilities in jquery 2.2.0	https://examplecomp.com	Low
WLRM-184	Information Exposure (Through an Error Message)	https://examplecomp.com	Low
WLRM-185	Information Exposure (Through an Error Message) in graphql	https://examplecomp.com	Low

3.1 Detailed description of the detected vulnerabilities

3.1.1. WLRM-90 Authentication Bypass Using an Alternate Path or Channel in sessions

Threat severity level: Critical

URI: https://payment.examplecomp.com

Description:

The software implements a user authentication mechanism, but the application has alternative authentication channels that allow you to bypass the main mechanism or take advantage of its shortcomings to gain access to user or administrator rights. As a result of this vulnerability, a potential attacker could gain access to sensitive user data or gain access to application management with administrator rights.

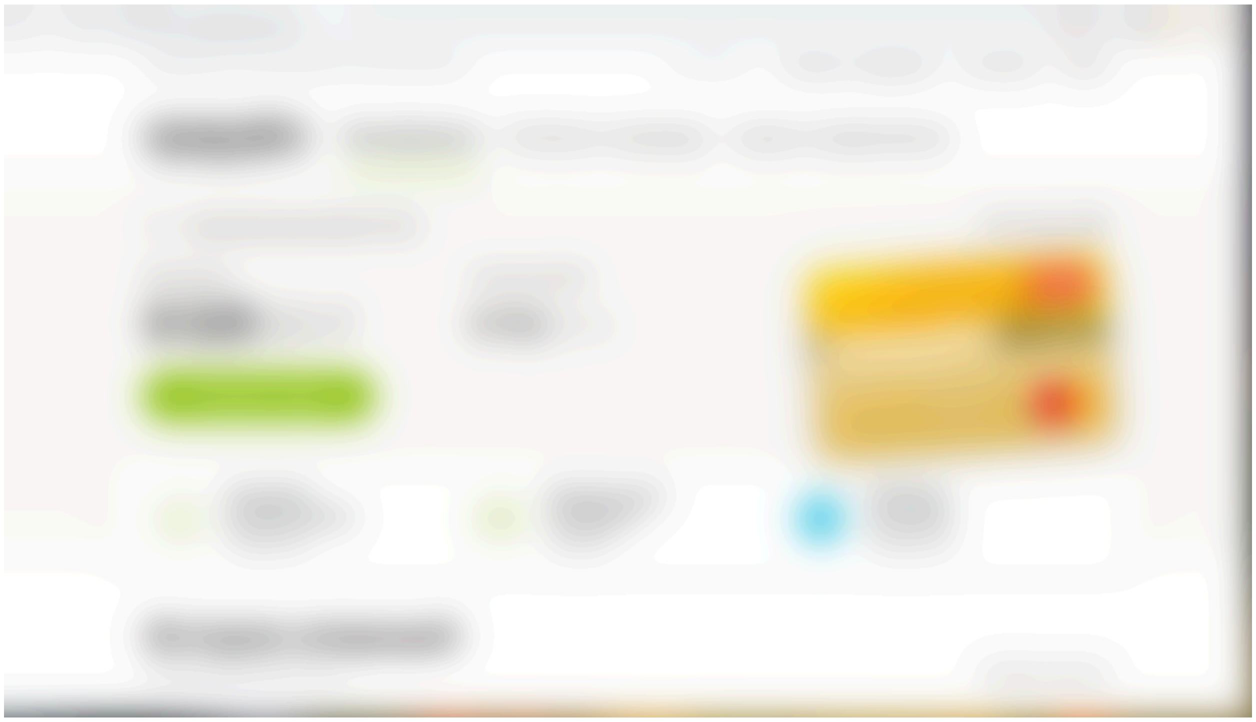
Exploitation:

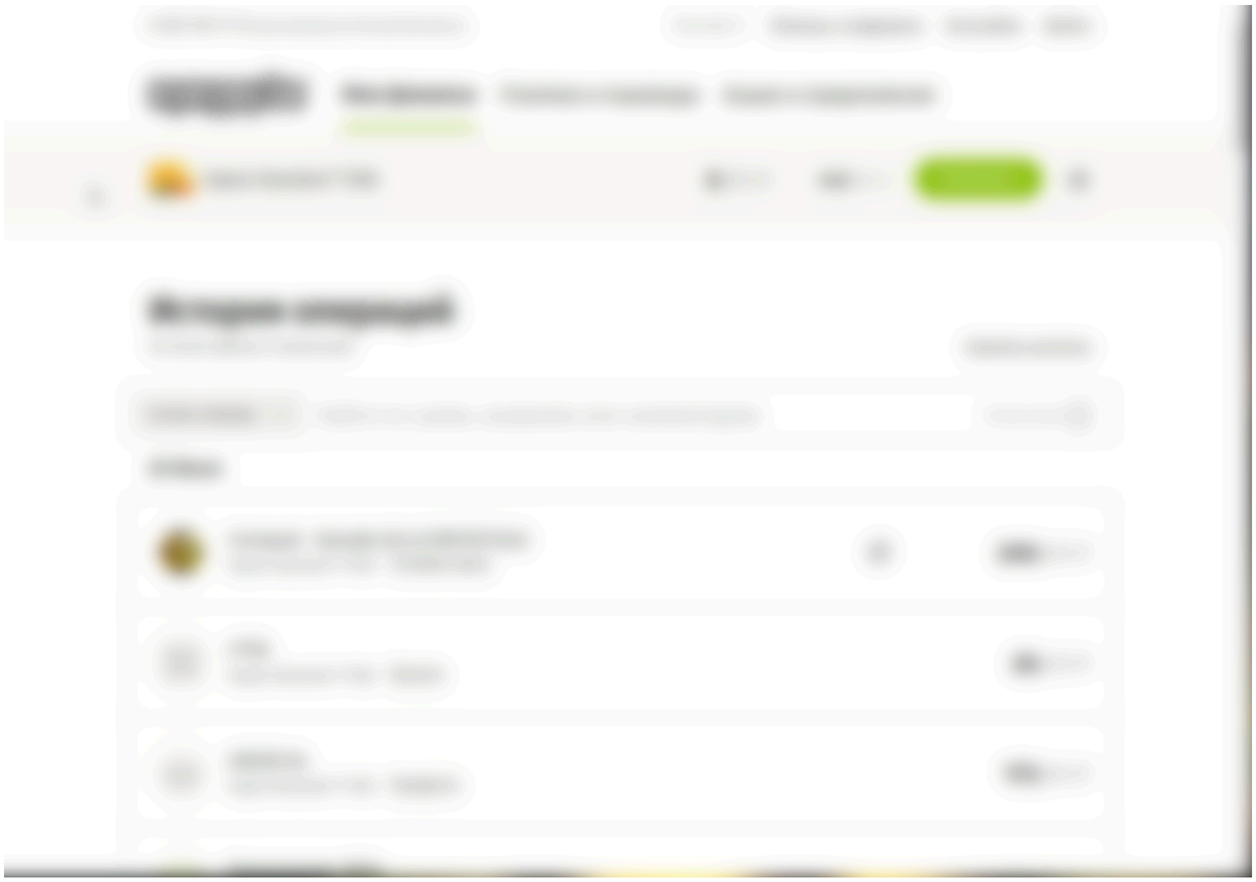
The vulnerability exists on the web server due to configuration or application implementation errors. The vulnerability is that executing a request to the server results in a 500 error, which causes the user session to be reinitialized. At reinitialization to the user by mistake, the session of the third-party client is given in this connection, and the arbitrary user can get access to the session of another third-party arbitrary user.

An example of a request that results in an error is shown below:

```
GET /api/v0001/ping/session?rid=29e84r6h1df00 HTTP/1.1
Host: payment.examplecomp.com
Connection: close
DNT: 1
X-XSRF-TOKEN: c4df59cfeaa5fd28c49955a00a4316f148e
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36
Content-Type: application/json;charset=utf-8
platform: web
channel: web
X-Request-Id: 29e84d41df00
Accept: */*
Referer: https://payment.examplecomp.com/
Cookie: WID=; ROUTEID=; PRINCIPAL=; CONSUMER_NAME=; XSRF-TOKEN=; SID=;
PLAY_ERRORS=; PLAY_SESSION=; max-age=wrt;
```

The result of the query is shown below, where the Executor managed to get access to the user interface of John Joe, as well as to access information about his cards and perform various actions:





During the testing of this vulnerability, 142,124 requests were made to the server payment.examplecomp.com.

During this time, 23,787 user accounts were accessed.

When capturing a third-party user session, the attacker gains access to all the information available in the web interface, such as name, email address, phone number, and account balances. In addition, it has the ability to make changes to the account and perform actions on behalf of the user in the web interface, except for those actions that require confirmation through the second factor.

Recommendations for remediation:

- eliminate the shortcomings of the existing authentication mechanism;
- eliminate any third-party authentication channels that allow potential attackers to access the application without going through the correct authentication process.

3.1.2. WLRM-186 Users' confidential Information Exposure

Threat severity level: Critical

URI: https://examplecomp.com

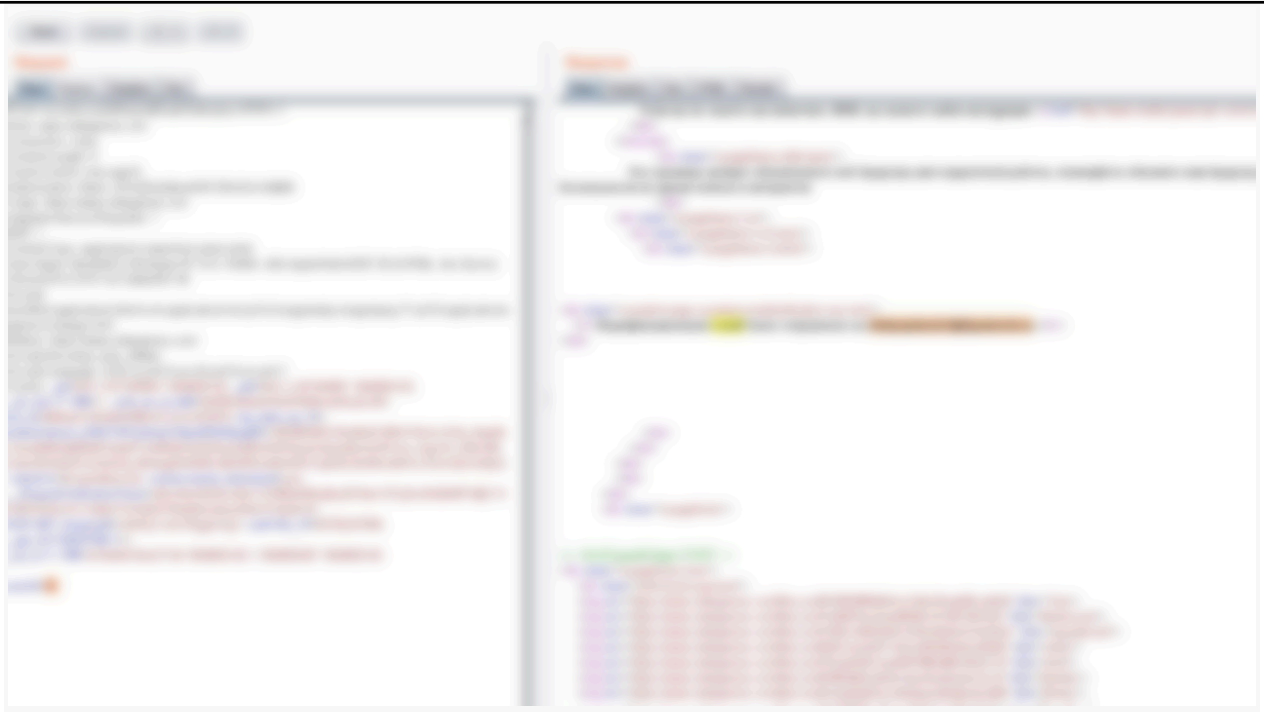
Description:

Information exposure is the intentional or unintentional disclosure of information to an actor who is not explicitly authorized to have access to that information.

Exploitation:

```
POST /api/auth/signin HTTP/1.1
Host: admin-api.examplecomp.com
Connection: close
Content-Length: 70
Accept: application/json, text/plain, */*
Origin: https://admin.examplecomp.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/74.0.3729.157 Safari/537.36
Content-Type: application/json
Referer: https://admin.examplecomp.com
Accept-Encoding: deflate
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

{"userName":"lsadmin","password":"rawsddas","deviceIdentifier":true}
```



Recommendations for remediation:

to eliminate the display of confidential information on the pages of the web application;

3.1.3. WLRM-188 Improper Restriction of Excessive Authentication Attempts in the authentication mechanism

Threat severity level: High

URI: https://prod.examplecomp.com

Description:

The audited web application does not implement a mechanism of protection against brute force of user authentication data. This flaw allows an attacker to perform multiple authentication attempts in a short period of time and gain administrative or user access to the system.

Exploitation:

Application prod.examplecomp.com allows you to search for user credentials. Knowing the user's login, it is possible to select the OTP (One Time Password) code in a short period of time.

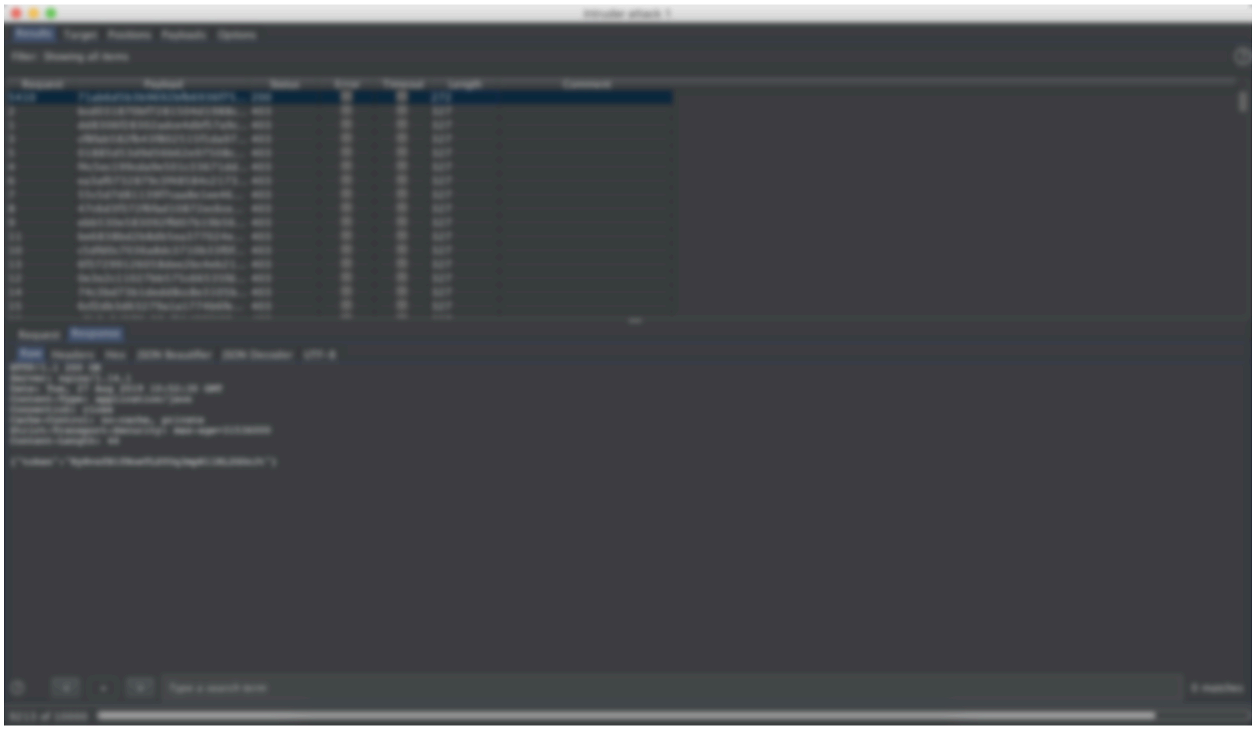
To demonstrate the presence of this vulnerability, a script was written to iterate over all possible values of the OTP code.

Bash-script:

```
for i in  $(seq -f "%04g" 0 9999)
do
    echo -n "29371190066salt$i" | shasum -a 256
done
```

Further, using the Burp Suite Intruder tool, 10 thousand requests were executed in a few minutes with a smooth increase in rps, to eliminate the possibility of denial of service of the system.

The figures below show the results of the queries and the correct OTP code found as a result of the search.



Successfully matched OTR code to the client's account

Recommendations for remediation:

- using CAPTCHA tests on multiple login attempts;
- the introduction of the lockout policy requestor;
- using persistent password policy;
- consider applying a Device Cookie-based protection mechanism (https://www.owasp.org/index.php/Slow_Down_Online_Guessing_Attacks_with_Device_Cookies)

3.1.4. WLRM-236 Authorization Bypass Through User-Controlled Key API Examplecomp.com

Threat severity level: High

URI: api.examplecomp.com

Description:

Application authorization features do not prevent one user from accessing another user's data or resources. This vulnerability occurs because a web application provides the ability, by changing the value of a key, to obtain a direct link to an object, such as a file, directory, or database entry, and does not exercise appropriate access control. In the process of exploiting the vulnerability, an attacker can gain unauthorized access to confidential information of the web application and users by manipulating the request parameters.

Exploitation:

The attacker can view other people's attachments to messages.

Get the message object.
Request:

```
GET /mail/1479 HTTP/1.1
X-Client-Type: android-application
application: Mobile.App;android;3.21.4
X-Android-Push-Token: b7a2b94f-3948-4fe0-a62f-24da1af0a9e0
User-Agent: ExampleComp/3.21.4 Android/5.1.1; A5010/LMY48Z
Host: api.examplecomp.com
Connection: close
Cookie:
.AIAUTH=FC14273A5B6F183DB90477EA52A6140AAE1635D2DA6943E042FE8B43D5616F000914
9BB7E414F3B43B712BEA4353545453FD5B4BE20D4A23EC34886BBD3016B7D2A68E257F7F6777
273A88769E1FDF5EB5; PasswordHash=UXdCUFVtreVJSzFZaUErYT3434V1dZZz090; afid=;
subafid=
Accept-Encoding: gzip, deflate
Via: 1.1 localhost (Apache-HttpClient/UNAVAILABLE (cache))
```

Response:

```
...
"attachments":[{"id":-2103201991,"free":true,"message":"Natasha"},{"id":-210
3201984,"free":false,"message":"Natalia"}]
...
```

Get a link to the photo. In the request, specify the message ID and the ID of the attached file.

Request:

```
GET /mail/1479/attachment/-2103201991 HTTP/1.1
X-Client-Type: android-application
application: Mobile.App;android;3.21.4
X-Android-Push-Token: b7a2b94f-3948-4fe0-a62f-24da1af0a9e0
User-Agent: ExampleComp/3.21.4 Android/5.1.1; A5010/LMY48Z
Host: api.examplecomp.com
Connection: close
Cookie:
.AIAUTH=FC14273A5B6F183DB90477EA52A6140AAE1635D2DA6943E042FE8B43D5616F000914
9BB7E414F3B43B712BEA4353545453FD5B4BE20D4A23EC34886BBD3016B7D2A68E257F7F6777
273A88769E1FDF5EB5; PasswordHash=UXdCUFVtreVJSzFZaUErYT3434V1dZZz090; afid=;
subafid=
Accept-Encoding: gzip, deflate
Via: 1.1 localhost (Apache-HttpClient/UNAVAILABLE (cache))
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: private, s-maxage=0
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/8.5
X-AspNetMvc-Version: 3.0
X-Powered-By: ASP.NET
X-Stage: Live
P3P: CP="NON DSP COR NID IVDo CONo IVAo PSD PSA TELo TAI ADM CUR OUR IND PHY
ONL UNI PUR FIN COM NAV INT CNT PRE"
Access-Control-Allow-Methods: GET,POST,DELETE
Content-Length: 122
Date: Mon, 28 Oct 2019 13:03:28 GMT
Connection: close
{"id":-2103201991,"free":false,"url":"/Attachments/PresentationTemplate/1153
940/03207211-dd58-4a79-8087-fb1623817c8d.JPG"}
```

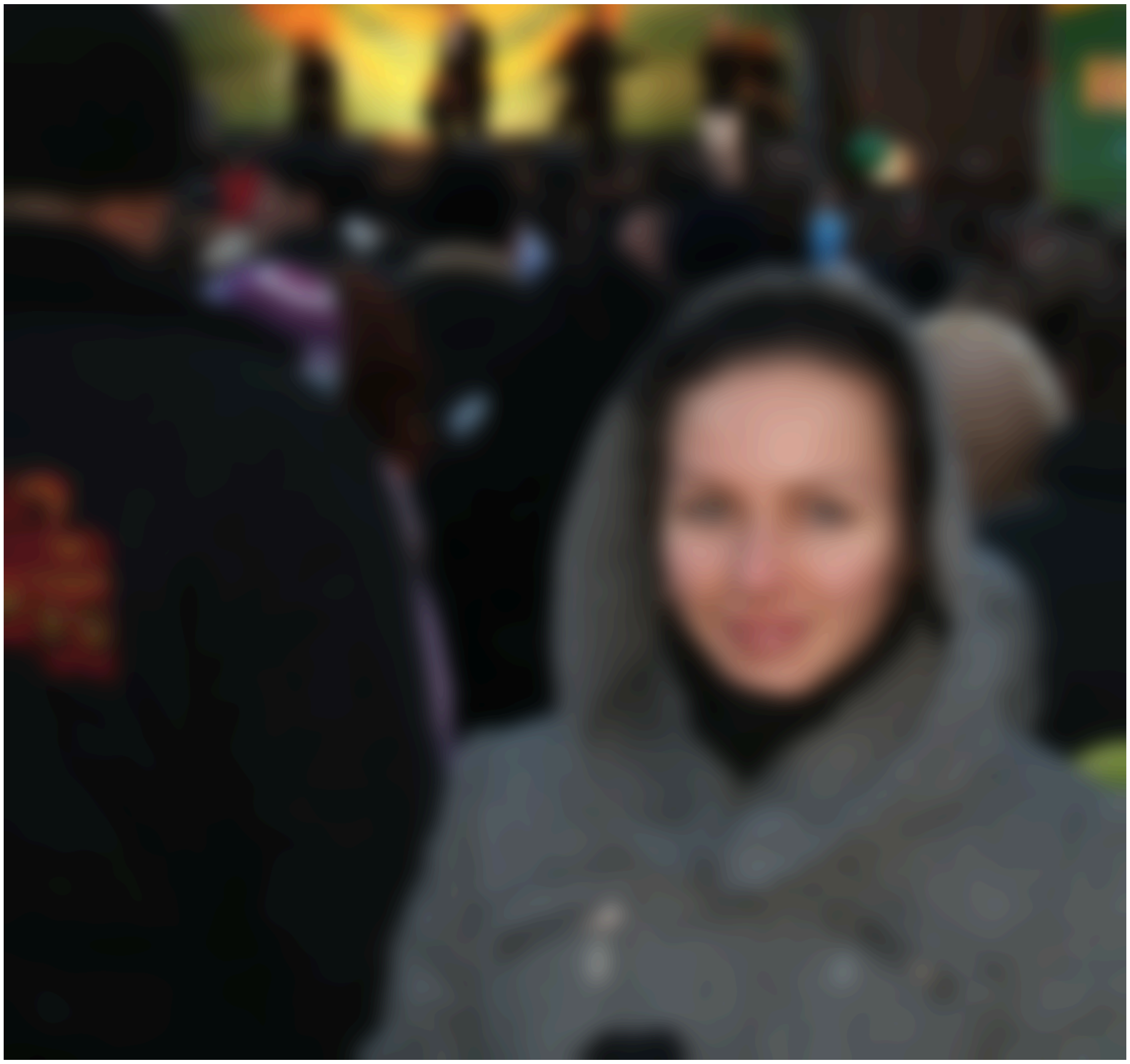
As a result we get the photo:
[http://examplecomp.com/Attachments/PresentationTemplate/1153940/03207211-dd58-4a79-8087-fb1623817c8d.JPG]

[illegible]

Get the message object. It contains information about the ID of the attached files.

[illegible]

Get a link to the attached file


Attached to someone else's message picture

Recommendations for remediation:

- it is necessary to implement correct verification of access control to web application resources;
- use indirect object references;
- it is necessary to implement the mechanism of access control to resources and to delimit access to them according to the rights of the user.

3.1.5. WLRM-183 Host Header poisoning in sellers section

Threat severity level: Medium

URI: https://examplecomp.com

Description:

The application trusts the user-provided “Host” header. Depending on the configuration of the server and other intermediate caching devices, it may be possible to conduct cache poisoning attacks by injecting the host header value into response pages.

You can learn more about this type of attack on the following resources:

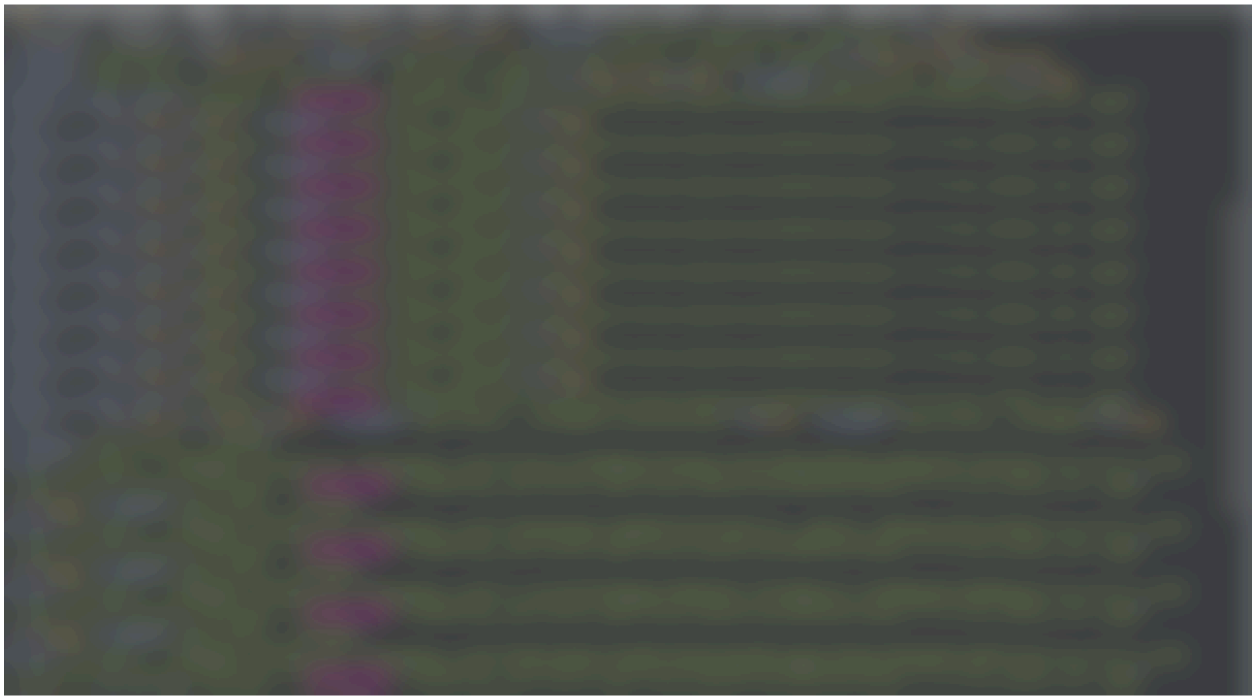
- http://carlos.bueno.org/2008/06/host-header-injection.html
- http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html

Example:

The application reflects the HOST header in the response pages and replaces the domain names of links to the content of the header, which, in the case of a cache server and certain of their settings, can provide attackers with the opportunity to carry out cache poisoning attacks.

An example of a request and response is shown in the figures below

Request with Host header substitution



Page content when spoofing Host header

Recommendations for remediation:

- use the white list of trusted domains during the initial site configuration process.

3.1.6. WLRM-172 Using Components with Known Vulnerabilities in jquery 2.2.0

Threat severity level: Low

URI: https://examplecomp.com

Description:

The audited application uses components with known vulnerabilities from the Common Vulnerabilities and Exposures (CVE) list or other sources, which can compromise the web application using exploits to existing vulnerabilities.

Example:

```
https://cdnjs.clist.com/ajax/libs/jquery/2.2.0/jquery.js

https://github.com/jquery/jquery/issues/2432
```

[redacted]

```
</button></div></form></div
s/jquery/2.2.0/jquery.js"
```

Jquery version

Recommendations for remediation:

- update the software (Hereinafter referred to as the software) and its components to the latest versions;
- take advantage of recommendations from developers of vulnerable software on mitigation of known vulnerabilities.

3.1.7. WLRM-184 Information Exposure (Through an Error Message)

Threat severity level: Low

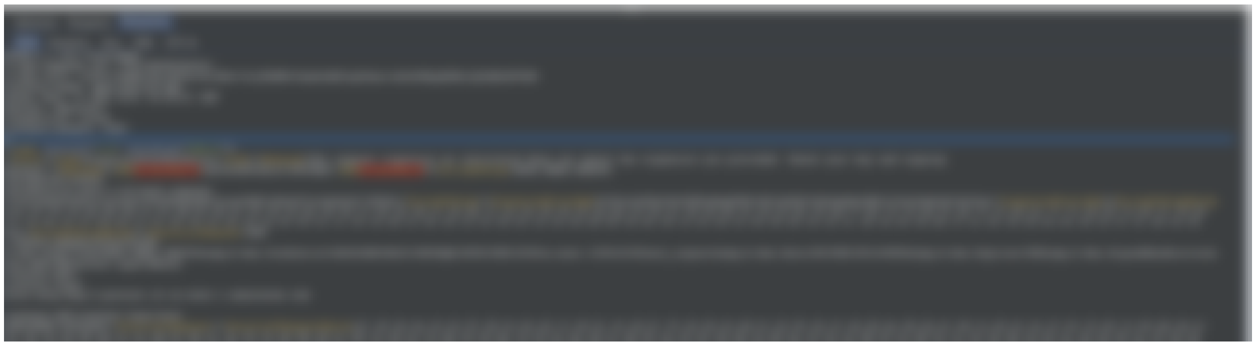
URI: https://examplecomp.com

Description:

The audited web application generates error messages that disclose information about the technologies used, software versions, and data in the web application. Because of this vulnerability, an attacker can obtain information about the implementation of a web application and then use it to find attack vectors, detect and exploit vulnerabilities. The vulnerability that leads to the disclosure of information occurs due to incorrect configuration of the web server.

Example:

The application discloses some of the secret data to access AWS.



Server response, disclosing AWSAccessKeyId.

Recommendations for remediation:

- to exclude debug information on the pages of the web application;
- store information about errors only in the log of the web application unavailable to users.

3.1.8. WLRM-185 Information Exposure (Through an Error Message) in graphql

Threat severity level: Low

URI: https://examplecomp.com

Description:

The audited web application generates error messages that disclose information about the technologies used, software versions, and data in the web application. Because of this vulnerability, an attacker can obtain information about the implementation of a web application and then use it to find attack vectors, detect and exploit vulnerabilities. The vulnerability that leads to the disclosure of information occurs due to incorrect configuration of the web-server.

Example:

The error trace reveals the versions of the dependencies used.

Figure 15 — Disclosure of the version of the software used

Recommendations for remediation:

- to exclude debug information on the pages of the web application;
- store information about errors only in the log of the web application unavailable to users.