Analysis Results

bsincomevalidation.zip

Report Date	2026-02-12 05:45:28
Report Author	dctuser
Classification Method	By severity
Product Version	10.16
Rules Version	10.127506

Confidentiality Note

01 Project Information

Project Name

bsincomevalidation.zip

UUID

3336d4ca-ec4a-48ce-8959-d604170877bc

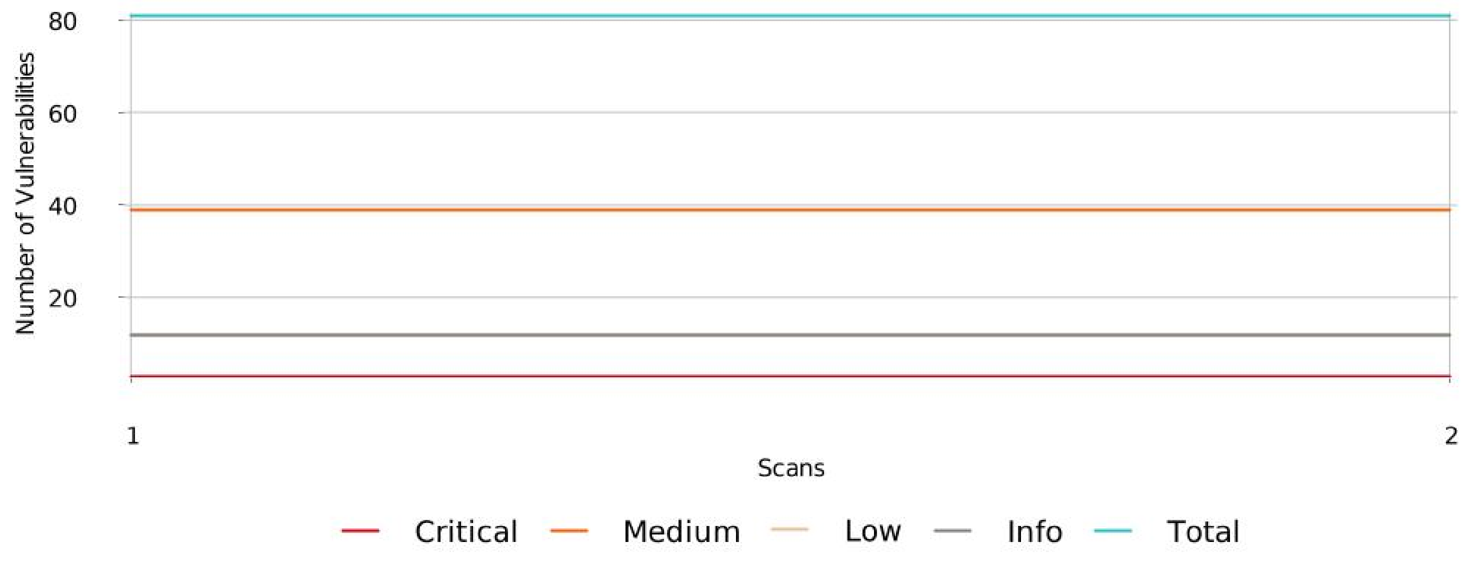
Project in DerScanner

Dynamics by vulnerabilities

Vulnerabilities are divided by severity level: critical, medium, low and info.

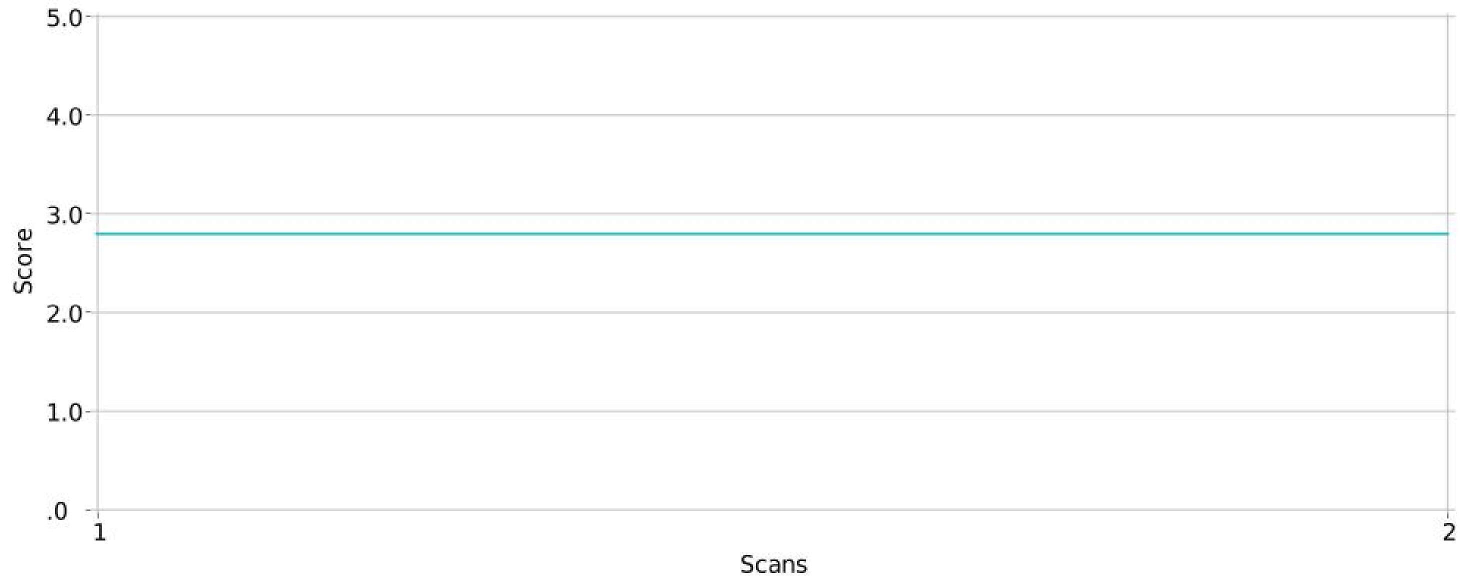
CRITICAL LEVEL	MEDIUM LEVEL	LOW LEVEL	INFORMATION
Likely to lead to compromise confidential data and violation of the integrity of the system.	May be less likely to lead to compromising confidential data and violating the integrity of the system, or are less serious security	Can become a potential security risk.	Signal a violation of good programming practice.

First of all, pay attention to vulnerabilities of critical and medium levels.



Dynamics by rating

The app score is calculated on a scale from 0 to 5. Score is calculated based on the number of critical and medium level vulnerabilities. The impact of critical vulnerabilities is greater than that of medium level vulnerabilities, and does not take into account the amount of code. Medium level vulnerabilities are taken into account based on their frequency and total number of source code lines.



Scan History

	Date and Time	Status	Languages	Lines of Code	Number of Vulnerabilities					Score
					Critical	Medium	Low	Info	Total	
2/2	2026-02-12 05:41:56	completed	HTML5, JavaScript, T-SQL, PL/SQL, Python, Config files	0	3	39	12	27	81	2.8/5.0
1/2	2026-02-12 05:35:54	completed	Config files, Python, PL/SQL, T-SQL, JavaScript, HTML5	35 538	3	39	12	27	81	2.8/5.0

02

Scan Information

2/2 2026-02-12 05:41:56
10.127506

Scan Statistics

Status	completed
Score	2.8/5.0
Duration	0:00:00
Lines of Code	0

Vulnerabilities

Critical	Medium	Low	Info	Total
3	39	12	27	81

Language	Status	Duration	Lines of Code	Number of Vulnerabilities				
				Critical	Medium	Low	Info	Total
HTML5	completed	0:00:00	0	0	0	0	0	0
JavaScript	completed	0:00:00	0	3	0	2	0	5
T-SQL	completed	0:00:00	0	0	0	0	0	0
PL/SQL	completed	0:00:00	0	0	0	0	0	0
Python	completed	0:00:00	0	0	36	6	27	69
Config files	completed	0:00:00	0	0	3	4	0	7

Language Statistics

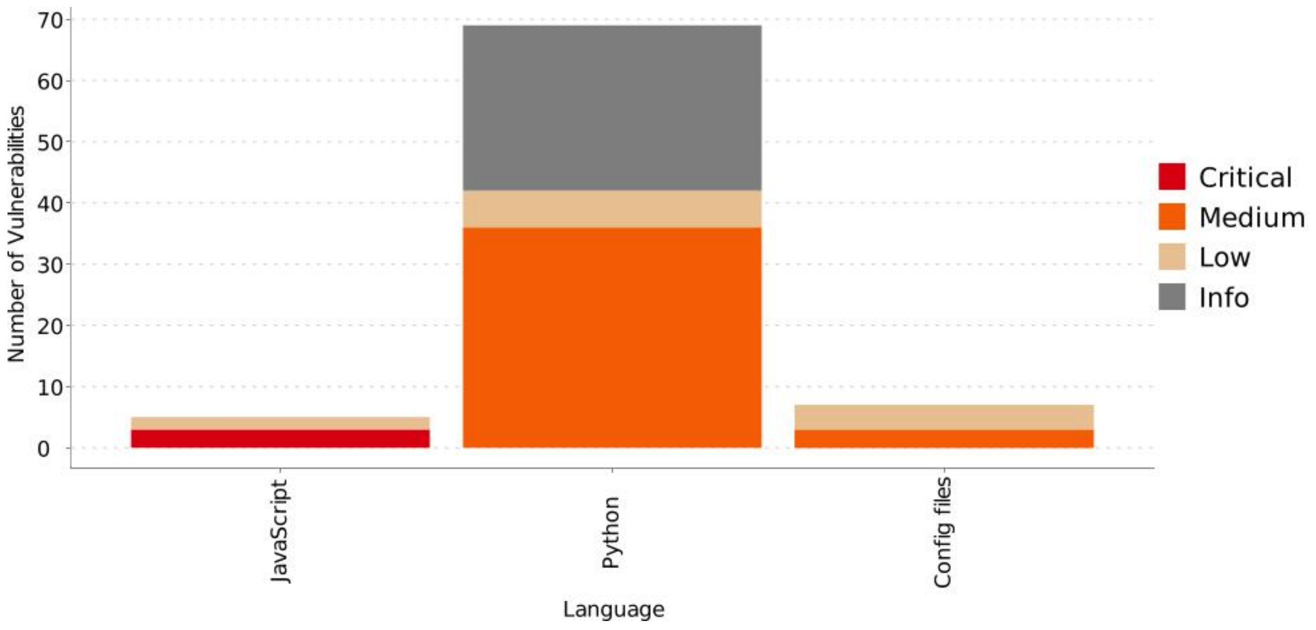
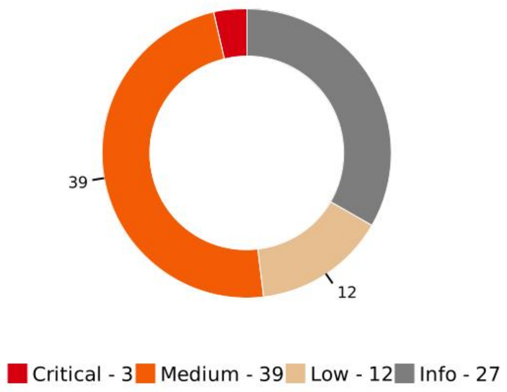
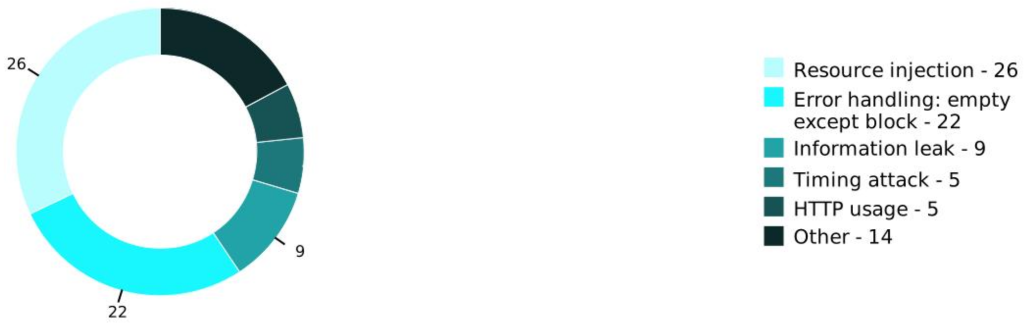


Diagram of identified vulnerabilities



Vulnerable Dependencies



Vulnerability List

Vulnerabilities are displayed accordingly to export settings: **42 selected**

Actual: **42 of 81**

Critical vulnerabilities		**3**
Persistent XSS	JavaScript	3
● bsincomevalidation/src/ui/db_viewer.html:408		Not processed
● bsincomevalidation/src/ui/db_viewer.html:450		Not processed
● bsincomevalidation/src/ui/db_viewer.html:483		Not processed
Medium-level vulnerabilities		**39**
Persistent authentication	Config files	3
● bsincomevalidation/src/ui/taxpayer_search_app.py:29		Not processed
● bsincomevalidation/src/ui/taxpayer_search_app.py:244		Not processed
● bsincomevalidation/src/ui/upload_app.py:43		Not processed
DOS attack via regular expressions possible	Python	1
● bsincomevalidation/src/halyk_ind/transactions.py:79		Not processed
Information leak	Python	9
● bsincomevalidation/src/bcc/footer.py:211#215		Not processed
● bsincomevalidation/src/bcc_ind/parser.py:341		Not processed
● bsincomevalidation/src/halyk_business/footer.py:193		Not processed
● bsincomevalidation/src/halyk_ind/parser.py:321		Not processed
● bsincomevalidation/src/halyk_ind/parser.py:324		Not processed
● bsincomevalidation/src/halyk_ind/parser.py:328		Not processed

* Rejected vulnerabilities are not taken into account

Medium-level vulnerabilities

Information leak	Python	
● bsincomevalidation/src/utils/income_calc.py:296#300		Not processed
● bsincomevalidation/src/utils/income_calc.py:309#313		Not processed
● bsincomevalidation/src/utils/income_calc.py:322#326		Not processed

Resource injection	Python	26
● bsincomevalidation/src/alatau_city_bank/batch_parse.py:98		Not processed
● bsincomevalidation/src/alatau_city_bank/batch_parse.py:137		Not processed
● bsincomevalidation/src/api/app.py:843		Not processed
● bsincomevalidation/src/api/storage.py:134		Not processed
● bsincomevalidation/src/api/storage.py:149		Not processed
● bsincomevalidation/src/api/storage.py:157		Not processed
● bsincomevalidation/src/api/storage.py:167		Not processed
● bsincomevalidation/src/bcc/batch_parse.py:84		Not processed
● bsincomevalidation/src/bcc/batch_parse.py:179		Not processed
● bsincomevalidation/src/bcc/footer.py:189		Not processed
● bsincomevalidation/src/bcc/parser.py:40		Not processed
● bsincomevalidation/src/eurasian_bank/batch_parse.py:79		Not processed
● bsincomevalidation/src/eurasian_bank/batch_parse.py:148		Not processed
● bsincomevalidation/src/forte_bank/batch_parse.py:115		Not processed
● bsincomevalidation/src/forte_bank/batch_parse.py:192		Not processed
● bsincomevalidation/src/forte_bank/footer.py:195		Not processed
● bsincomevalidation/src/freedom_bank/batch_parse.py:107		Not processed
● bsincomevalidation/src/freedom_bank/batch_parse.py:189		Not processed
● bsincomevalidation/src/halyk_business/footer.py:143		Not processed
● bsincomevalidation/src/halyk_business/footer.py:158		Not processed
● bsincomevalidation/src/halyk_business/footer.py:200		Not processed

* Rejected vulnerabilities are not taken into account

Medium-level vulnerabilities

Resource injection	Python
● bsincomevalidation/src/halyk_ind/parser.py:55	Not processed
● bsincomevalidation/src/kaspi_pay/parser.py:42	Not processed
● bsincomevalidation/src/utils/convert_pdf_json_page.py:208	Not processed
● bsincomevalidation/src/utils/convert_pdf_json_pages.py:65	Not processed
● bsincomevalidation/src/utils/path_security.py:111	Not processed

Low-level vulnerabilities 0

Info-level vulnerabilities 0

Analysis Results

Persistent XSS (JavaScript)

Description

Persistent XSS or server XSS attack is possible.
Cross-site scripting is one of the most common types of attacks on web applications. XSS attacks take seventh place in the “OWASP Top 10 2017” list of ten most significant vulnerabilities in web applications.
The main phase of any XSS attack is an imperceptible for the victim execution of a malicious code in the context of the vulnerable application. For this purpose, the functionality of the client application (browser) is used that allows to automatically execute scripts embedded in web page code. In most cases, these malicious scripts are implemented in JavaScript.
Consequences of an XSS attack vary from violations of application functionality to complete loss of user data confidentiality. The malicious code during the XSS attack can steal user HTTP-cookie, which gives an attacker the ability to make requests to the server on behalf of the user.
OWASP suggests the following classification of XSS attacks:

- Server type XSS occurs when data from an untrusted source is included in the response returned by the server. The source of such data can be both user input and server database (where it had been previously injected by an attacker who exploited vulnerabilities in the server-side application).
- Client type XSS occurs when the raw data from the user input contains code that changes the Document Object Model (DOM) of the web page received from the server. The source of such data can be both the DOM and the data received from the server (e.g., in response to an AJAX request).

Typical server type attack scenario:

1. Unvalidated data, usually from a HTTP request, gets into the server part of the application.
2. The server dynamically generates a web page that contains the unvalidated data.
3. In the process of generating a web page, server does not prevent the inclusion of an executable code that can be executed in the client (browser), such as JavaScript code language, HTML-tags, HTML-attributes, Flash, ActiveX, etc., in the page code.
4. The victim’s client application displays the web page that contains the malicious code injected via data from an untrusted source.
5. Since malicious code is injected in the web page coming from the known server, the client part of the application (browser) executes it with the rights set for the application.
6. This violates the same-origin policy, according to which the code from the one source must not get an access to resources from another source.

Client type attacks are executed in a similar way with the only difference that the malicious code is injected during the phase of the client application work with the document object model received from the server.

Example

In the following example, the application obtains user identifier (name) from the URL and displays it:
var name = localStorage.getItem("name");
document.write(name);
The code operates correctly if name does not contain special characters. If name contains code it will be executed in the victim’s web browser.
If the name values do not contain special characters, the code behaves correctly. But if the name value is derived from data from an untrusted source (e.g., user input), the attacker can store the malicious code in the database. Such attacks are particularly dangerous because they may affect a large number of users.

Recommendations

- Implement a validation mechanism. Whitelist is more secure but less flexible than the blacklist. The blacklist must at least include the characters “&”, “<”, “>”, and quotation marks.
- Many web application servers provide their own mechanisms of protection against XSS, but they may not be considered sufficient. There is no guarantee that the application will run in conjunction with the server that updates these mechanisms timely and completely.

Links

1. OWASP: Cross-site Scripting (XSS)
2. CWE-79: Improper Neutralization of Input During Web Page Generation
3. Types of Cross-Site Scripting - OWASP
4. OWASP: XSS Prevention Cheat Sheet
5. OWASP Top 10-2017 A7-Cross-Site Scripting (XSS)
6. CWE CATEGORY: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
7. CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
8. CWE-81: Improper Neutralization of Script in an Error Message Web Page
9. CWE-83: Improper Neutralization of Script in Attributes in a Web Page
10. Cross-site Scripting (XSS) Affecting jquery-mobile package

Vulnerability Entries

bsincomevalidation/src/ui/db_viewer.html:408

Level	Critical

Status Not processed

```
405    });
406
407    html += '</tbody></table>';

408    document.getElementById('statementsContainer').innerHTML = html;

409 } catch (error) {
410    showError('Ошибка загрузки выписок: ' + error.message);
411 }
```

Trace

row

bsincomevalidation/src/ui/db_viewer.html:394

```
391 html += '</tr></thead><tbody>';
392
393 data.forEach(row => {

394    const period = `${escapeHtml(row.period_from || 'Н/Д')} по ${escapeHtml(row.
period_to || 'Н/Д')}`;

395    const created = new Date(row.created_at).toLocaleDateString();
396    html += `<tr>
397       <td>${escapeHtml(row.bank || '')}</td>
```

html

bsincomevalidation/src/ui/db_viewer.html:408

```
405    });
406
407    html += '</tbody></table>';

408    document.getElementById('statementsContainer').innerHTML = html;

409 } catch (error) {
410    showError('Ошибка загрузки выписок: ' + error.message);
411 }
```

bsincomevalidation/src/ui/db_viewer.html:450

Level	Critical
Status	Not processed

```
447    });
448
449    html += '</tbody></table>';
450    document.getElementById('transactionsContainer').innerHTML = html;
451 } catch (error) {
452    showError('Ошибка загрузки транзакций: ' + error.message);
453 }
```

Trace

row

bsincomevalidation/src/ui/db_viewer.html:434

```
431 html += '</tr></thead><tbody>';
432
433 data.forEach(row => {
434    const date = escapeHtml(row.operation_date || 'Н/Д');
435    const debit = row.debit_amount ? `<span class="amount-cell negative">${escapeHtml
(row.debit_amount.toLocaleString())}</span>` : '-';
436    const credit = row.credit_amount ? `<span class="amount-cell positive">${escapeHtml
(row.credit_amount.toLocaleString())}</span>` : '-';
437
```

html

bsincomevalidation/src/ui/db_viewer.html:450

```
447    });
448
449    html += '</tbody></table>';
450    document.getElementById('transactionsContainer').innerHTML = html;
```

```
451 } catch (error) {
452     showError('Ошибка загрузки транзакций: ' + error.message);
453 }
```

bsincomevalidation/src/ui/db_viewer.html:483

Level	Critical
Status	Not processed

```
480     });
481
482     html += '</tbody></table>';

483     document.getElementById('clientsContainer').innerHTML = html;

484 } catch (error) {
485     showError('Ошибка загрузки клиентов: ' + error.message);
486 }
```

Trace

row

bsincomevalidation/src/ui/db_viewer.html:471

```
468 html += '</tr></thead><tbody>';
469
470 data.forEach(row => {

471     const created = new Date(row.created_at).toLocaleDateString();

472     html += `<tr>
473         <td><strong>${escapeHtml(row.iin_bin || '')}</strong></td>
474         <td>${escapeHtml(row.full_name || '')}</td>
```

html

bsincomevalidation/src/ui/db_viewer.html:483

```
480     });
```

```

    html += '</tbody></table>';
    document.getElementById('clientsContainer').innerHTML = html;
} catch (error) {
    showError('Ошибка загрузки клиентов: ' + error.message);
}
```

Persistent authentication (Config files)

Description

The application uses a permanent Bearer authentication token. At the end of the life of the token, the application may not work correctly. If the token has a long lifetime, after getting it, any subject will be able to authorize into the service.

Example

In the following example, the application uses a persistent Bearer token:

```
{
  "key": "Authorization",
  "value": "Bearer ...Token..."
}
```

Recommendations

- Dynamically add token to request header.
- Limit the validity of the authentication token.

Links

1. FormsAuthentication.RedirectFromLoginPage Method - msdn.microsoft.com
2. OWASP Top 10 2017 A2-Broken Authentication
3. CWE CATEGORY: OWASP Top Ten 2017 Category A2 - Broken Authentication

Vulnerability Entries

bsincomevalidation/src/ui/taxpayer_search_app.py:29

Level	Medium
Status	Not processed

```
26 # Константы
27 # SECURITY: Use environment variables instead of hardcoded values
28 import os
29 DEFAULT_PORTAL_TOKEN = os.environ.get("TAXPAYER_API_PORTAL_TOKEN", "")
30
31
32 def init_session_state() -> None:
```

bsincomevalidation/src/ui/taxpayer_search_app.py:244

Level	Medium
Status	Not processed

```
241 try:
242     client = TaxpayerAPIClient(
243         portal_host=st.session_state.portal_host,
244         portal_token=st.session_state.portal_token
245     )
246
247     taxpayer_type_enum = TaxpayerType[taxpayer_type]
```

bsincomevalidation/src/ui/upload_app.py:43

Level	Medium
Status	Not processed

```
    "TAXPAYER_API_PORTAL_HOST",
    "https://portal.kgd.gov.kz"  # Default, should be overridden via env var
)
TAXPAYER_API_PORTAL_TOKEN = os.environ.get(
    "TAXPAYER_API_PORTAL_TOKEN",
    ""  # Must be set via environment variable in production
)
```

DOS attack via regular expressions possible (Python)

Description

The regexp used is from an unreliable source, which can be computationally intensive for some inputs. Regular expression denial of service (ReDOS) attack is possible.
Regular expressions are widely used in applications to validate the user-supplied data. Expressions containing structures like (( )+)+ cause execution of a significant amount of iterations. By inputting a certain type of string an attacker can disrupt the application operation. All implementations of regular expressions have such vulnerabilities.

Example

The following regular expression performs a cycle of 32,768 iterations on the input string aaaaaaaaaaaaaaaX:

```
^(a+)+$
```

Recommendations

- Do not use the data that is obtained from an untrusted source has not passed validation in regular expressions.
- Avoid regular expressions containing nested groups of repeating characters.
- Check the regular expression for the possibility of ReDOS attack using safeRegex.

Links

1. OWASP: Regular expression Denial of Service
2. Runaway Regular Expressions: Catastrophic Backtracking – regular-expressions.info
3. saferegex – Tool for testing regular expressions for ReDoS vulnerabilities

4. re - Regular expression operations
5. CWE-400

Vulnerability Entries

bsincomevalidation/src/halyk_ind/transactions.py:79

Level	Medium
Status	Not processed

```
76    "fee":          r"\bкомисси[яи]\b",
77    "account":      r"\bNº\s*карточки/счета\b|\bкарточки/счета\b",
78 }

79 RX = {k: re.compile(v, re.I) for k, v in ANCHORS.items()}

80 _MAX_REGEX_INPUT = 2000  # ReDoS mitigation: truncate before regex
81 _MAX_REGEX_INPUT_LEN = 500  # Limit input to prevent ReDoS
82
```

Information leak (Python)

Description

System configuration information leak is possible. This can help an attacker to create a plan of an attack. Debug information and error messages can be written to the log, displayed to the console, or sent to the user depending on the system settings. In some cases, an attacker can make a conclusion about the system vulnerabilities from the error message. For example, a database error can indicate insecurity against SQL injection attacks. Information about the version of the operating system, server applications and system configurations can also be of value to the attacker.

Example

In the following example, the application turns on debug mode:

```
from django.conf import settings
settings.configure(DEBUG=True)
```

Recommendations

- Exclude detailed information about the system and its configuration from the error messages.

Links

1. OWASP Top 10 2013-A5-Security Misconfiguration
2. CWE-497: Exposure of System Data to an Unauthorized Control Sphere
3. OWASP Top 10 2017-A3-Sensitive Data Exposure
4. OWASP Top 10 2017-A6-Security Misconfiguration
5. CWE CATEGORY: OWASP Top Ten 2017 Category A5 - Broken Access Control
6. CWE CATEGORY: OWASP Top Ten 2017 Category A6 - Security Misconfiguration
7. CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
8. CWE-209: Generation of Error Message Containing Sensitive Information
9. CWE-489

Vulnerability Entries

bsincomevalidation/src/bcc/footer.py:211#215

Level	Medium
Status	Not processed

```
208 DEBUG_MODE = os.environ.get("DEBUG_PARSER", "false").lower() == "true"
209
210 if DEBUG_MODE:
211     print({
212         "total_debit": res["total_debit"],
213         "total_credit": res["total_credit"],
214         "closing_balance": res["closing_balance"],
215     })
216 else:
217     print(f"Summary: debit={res['total_debit']}, credit={res['total_credit']}, balance={res
['closing_balance']}")
218 })
```

bsincomevalidation/src/bcc_ind/parser.py:341

Level Medium

Status Not processed

```
338 DEBUG_MODE = os.environ.get("DEBUG_PARSER", "false").lower() == "true"
339 if DEBUG_MODE:
340     with pd.option_context("display.max_colwidth", None):
341         print(df.head(3)[["Описание операции"]].to_string(index=False))
342 else:
343     print(f"Parsed {len(df)} rows (operation details hidden)")
344
```

Trace

Path

bsincomevalidation/src/bcc_ind/parser.py:312

```
309 ap.add_argument("--calib-pages", default="1,2,3", help="Pages to try for calibration (e.g.
'1,2,3').")
310 args = ap.parse_args()
311
312 pdf = Path(args.pdf)
313 if not pdf.exists():
314     raise SystemExit(f"File not found: {pdf}")
315
```

print

bsincomevalidation/src/bcc_ind/parser.py:341

```
338 DEBUG_MODE = os.environ.get("DEBUG_PARSER", "false").lower() == "true"
339 if DEBUG_MODE:
340     with pd.option_context("display.max_colwidth", None):
341         print(df.head(3)[["Описание операции"]].to_string(index=False))
342 else:
343     print(f"Parsed {len(df)} rows (operation details hidden)")
344
```

bsincomevalidation/src/halyk_business/footer.py:193

Level	Medium
Status	Not processed

```
190 import os
191 DEBUG_MODE = os.environ.get("DEBUG_PARSER", "false").lower() == "true"
192 if DEBUG_MODE:
193     print(df.to_string(index=False))
194 else:
195     print(f"Summary: {len(df)} rows parsed (details hidden)")
196
```

bsincomevalidation/src/halyk_ind/parser.py:321

Level	Medium
Status	Not processed

```
318
319 if DEBUG_MODE:
320     print("=== HEADER ===")
321     print(header_df.to_string(index=False))
322
323     print("\n=== TX (first 20) ===")
324     print(tx_df.head(20).to_string(index=False))
```

bsincomevalidation/src/halyk_ind/parser.py:324

Level	Medium

Status Not processed

```
321 print(header_df.to_string(index=False))
322
323 print("\n=== TX (first 20) ===")
324 print(tx_df.head(20).to_string(index=False))
325
326 print("\n=== FOOTER ===")
327 if not footer_df.empty:
```

bsincomevalidation/src/halyk_ind/parser.py:328

Level Medium

Status Not processed

```
325
326     print("\n=== FOOTER ===")
327     if not footer_df.empty:
328         print(footer_df.to_string(index=False))
329     else:
330         print("<empty>")
331 else:
```

bsincomevalidation/src/utils/income_calc.py:296#300

Level Medium

Status Not processed

```
293 if DEBUG_MODE:
294     print("\n[income_calc] examples excluded by KNP:")
295     # Only show non-sensitive columns in debug mode
296     print(
297         df.loc[df["ip_is_non_business_by_knp"],
298             [col_op_date, col_knp, col_credit]]
299         .head(max_examples)
300     )
301 else:
302     print(f"\n[income_calc] {n_knp} transactions excluded by KNP (details hidden)")
303
```

bsincomevalidation/src/utils/income_calc.py:309#313

Level Medium

Status Not processed

```
306 if DEBUG_MODE:
307     print("\n[income_calc] examples excluded by keywords:")
308     # Only show non-sensitive columns in debug mode
309     print(
310         df.loc[df["ip_is_non_business_by_keywords"],
311                [col_op_date, col_knp, col_credit]]
312         .head(max_examples)
313     )
314 else:
315     print(f"\n[income_calc] {n_kw} transactions excluded by keywords (details hidden)")
316
```

bsincomevalidation/src/utils/income_calc.py:322#326

Level Medium

Status Not processed

```
319 if DEBUG_MODE:
320     print("\n[income_calc] examples KEPT due to KNP=099 + 'возмещение/гарант':")
321     # Only show non-sensitive columns in debug mode
322     print(
323         df.loc[override_keep_mask,
324                [col_op_date, col_knp, col_credit]]
325         .head(max_examples)
```

```
    )

else:
    print(f"\n[income_calc] {n_override} transactions kept due to override rules (details hidden)")

```

Resource injection (Python)

Description

An attacker can gain access to reading and changing protected system resources of the application has the ability to change the resource identifier.
Resource injection occurs when an atacker can specify identifier that will be used to access the system resource (for example, the port number to connect to the network resource). This allows him, in particular, to transfer valuable data to thied-party server.

Example

In the following example, the program executes a HTTP request in order to find out the price of tickets:

```
host_name = request.GET['host_example']
dbc = db.connect(host = host_name, port = 1235, dbname = ticket_data_base)
c = dbc.cursor()
###
result = c.execute('SELECT * FROM price_list')
###
```

If the request contains special characters, the attacker can receive confidential data.

Recommendations

- Create a whitelist of valid resource IDs and allow a user to select from this list and not to set his/her own value.
- If maintaining a whitelist is too difficult because of the large number of valid IDs, create a whitelist of characters allowed in identifiers. Blacklist in this case is ineffective, as it is likely to initially be incomplete, or sooner or later cease to be relevant.
- If nonetheless blacklist is chosen as a validation mechanism, make sure that it takes into account all the possible encodings and special character values (different for different operating systems). Changing the list should be simple when changing the requirements for

validation.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP Top 10 2013-A1-Injection
3. OWASP Top 10 2013-A4-Insecure Direct Object References
4. CWE-99: Improper Control of Resource Identifiers (‘Resource Injection’)
5. Dangerous Python Functions, Part 2
6. Create, use, and remove temporary files securely
7. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
8. CWE-1030

Vulnerability Entries

bsincomevalidation/src/alatau_city_bank/batch_parse.py:98

Level	Medium
Status	Not processed

```
95             xref = {"error": f"{type(e).__name__}: {e}"}
96         out["XRef"] = xref
97

98 with open(out_path, "w", encoding="utf-8") as f:

99     json.dump(out, f, ensure_ascii=False, indent=2)
100
101 print(f"[json] Dumped PDF internals to {out_path}")
```

bsincomevalidation/src/alatau_city_bank/batch_parse.py:137

Level	Medium
Status	Not processed

```
134 if not json_path.exists() or not json_path.is_file():
135     raise ValueError(f"Invalid JSON path: {json_path}")
136

137 with open(json_path, "r", encoding="utf-8") as f:
```

```
138     pdf_json = json.load(f)
139
140 closing_date = header_df.iloc[0].get("closing_balance_date")
```

bsincomevalidation/src/api/app.py:843

Level	Medium
Status	Not processed

```
840     resolved_path = ui_file.resolve()
841     # Ensure path is within project or expected locations
842     if resolved_path.exists() and resolved_path.is_file():
843         with open(resolved_path, 'r', encoding='utf-8') as f:
844             return Response(content=f.read(), media_type="text/html")
845 except (OSError, ValueError) as e:
846     # Skip invalid paths
```

bsincomevalidation/src/api/storage.py:134

Level	Medium
Status	Not processed

```
131
132 # Save project metadata
133 project_file = self._get_project_file(project_id)
134 with open(project_file, 'w', encoding='utf-8') as f:
135     json.dump(project.to_dict(), f, ensure_ascii=False, indent=2)
136
137 # Create directory for project files
```

bsincomevalidation/src/api/storage.py:149

Level	Medium

Status Not processed

```
146 if not project_file.exists():
147     return None
148
149 with open(project_file, 'r', encoding='utf-8') as f:
150     data = json.load(f)
151
152 return Project.from_dict(data)
```

bsincomevalidation/src/api/storage.py:157

Level Medium

Status Not processed

```
154 def update_project(self, project: Project):
155     """Update existing project"""
156     project_file = self._get_project_file(project.project_id)
157     with open(project_file, 'w', encoding='utf-8') as f:
158         json.dump(project.to_dict(), f, ensure_ascii=False, indent=2)
159
160 def get_projects_by_iin(self, iin: str) -> List[Project]:
```

bsincomevalidation/src/api/storage.py:167

Level Medium

Status Not processed

```
164 # Scan all project files
165 for project_file in self.base_dir.glob("project_*.json"):
166     try:
167         with open(project_file, 'r', encoding='utf-8') as f:
```

```
168         data = json.load(f)
169
170       if data.get('iin') == iin:
```

bsincomevalidation/src/bcc/batch_parse.py:84

Level	Medium
Status	Not processed

```
81    # we skip XRef for now – not needed for metadata validation
82
83 validated = validate_path_for_write(json_path, meta_dir)
84 with open(validated, "w", encoding="utf-8") as f:
85    json.dump(out, f, ensure_ascii=False, indent=2)
86
87 return json_path
```

bsincomevalidation/src/bcc/batch_parse.py:179

Level	Medium
Status	Not processed

```
176 # Security: Validate path before opening
177 from src.utils.path_security import validate_path
178 validated_path = validate_path(meta_json_path, pdf_meta_dir)
179 with open(validated_path, "r", encoding="utf-8") as f:
180    pdf_json = json.load(f)
181
182 period_end = header_df.iloc[0].get("Период (конец)")
```

bsincomevalidation/src/bcc/footer.py:189

Level	Medium

Status Not processed

```
186 validated_path = validate_path(jsonl_path)
187
188 pages: List[Dict[str, Any]] = []
189 with open(validated_path, "r", encoding="utf-8") as f:
190     for line in f:
191         line = line.strip()
192         if line:
```

bsincomevalidation/src/bcc/parser.py:40

Level Medium

Status Not processed

```
37
38 pages: List[Dict[str, Any]] = []
39
40 with open(validated_path, "r", encoding="utf-8") as f:
41     for line in f:
42         line = line.strip()
43         if not line:
```

bsincomevalidation/src/eurasian_bank/batch_parse.py:79

Level Medium

Status Not processed

```
76
77 from src.utils.path_security import validate_path_for_write
78 validated = validate_path_for_write(json_path, meta_dir)
79 with open(validated, "w", encoding="utf-8") as f:
```

```
80    json.dump(out, f, ensure_ascii=False, indent=2)
81
82 return json_path
```

bsincomevalidation/src/eurasian_bank/batch_parse.py:148

Level	Medium
Status	Not processed

```
145 try:
146     from src.utils.path_security import validate_path
147     validated = validate_path(meta_json_path, pdf_meta_dir)
148     with open(validated, "r", encoding="utf-8") as f:
149         pdf_json = json.load(f)
150
151     period_end = header_df.iloc[0].get("period_end")
```

bsincomevalidation/src/forte_bank/batch_parse.py:115

Level	Medium
Status	Not processed

```
112     )
113
114 validated = validate_path_for_write(json_path, meta_dir)
115 with open(validated, "w", encoding="utf-8") as f:
116     json.dump(out, f, ensure_ascii=False, indent=2, default=str)
117
118 return json_path
```

bsincomevalidation/src/forte_bank/batch_parse.py:192

Level	Medium

Status Not processed

```
189 try:
190     from src.utils.path_security import validate_path
191     validated = validate_path(meta_json_path, pdf_meta_dir)

192     with open(validated, "r", encoding="utf-8") as f:

193         pdf_json = json.load(f)
194
195     period_end = header_df.iloc[0].get("period_end")
```

bsincomevalidation/src/forte_bank/footer.py:195

Level Medium

Status Not processed

```
192 from src.utils.path_security import validate_path
193 validated = validate_path(jsonl_path)
194 pages: List[Dict[str, Any]] = []

195 with open(validated, "r", encoding="utf-8") as f:

196     for line in f:
197         line = line.strip()
198         if line:
```

bsincomevalidation/src/freedom_bank/batch_parse.py:107

Level Medium

Status Not processed

```
104 # make sure there are no Decimal (or other non-JSON) types
105 out_clean = _json_safe(out)
106

107 with open(json_path, "w", encoding="utf-8") as f:
```

```
108    json.dump(out_clean, f, ensure_ascii=False, indent=2)
109
110 return json_path
```

bsincomevalidation/src/freedom_bank/batch_parse.py:189

Level	Medium
Status	Not processed

```
186 try:
187    from src.utils.path_security import validate_path
188    validated = validate_path(meta_json_path, pdf_meta_dir)
189    with open(validated, "r", encoding="utf-8") as f:
190        pdf_json = json.load(f)
191
192    period_end = header_df.iloc[0].get("period_end")
```

bsincomevalidation/src/halyk_business/footer.py:143

Level	Medium
Status	Not processed

```
140 from src.utils.path_security import validate_path
141 validated = validate_path(source_path)
142 last_obj: Dict[str, Any] = {}
143 with open(validated, "r", encoding="utf-8") as f:
144    for line in f:
145        line = line.strip()
146        if not line:
```

bsincomevalidation/src/halyk_business/footer.py:158

Level	Medium

Status Not processed

```
155
156 from src.utils.path_security import validate_path
157 validated = validate_path(source_path)
158 with open(validated, "r", encoding="utf-8") as f:
159     return f.read()
160
161
```

bsincomevalidation/src/halyk_business/footer.py:200

Level Medium

Status Not processed

```
197 if args.out_json:
198     from src.utils.path_security import sanitize_filename
199     out_json_safe = Path(args.out_json).resolve()
200     with open(out_json_safe, "w", encoding="utf-8") as f:
201         json.dump(res, f, ensure_ascii=False, indent=2)
202     print(f" JSON saved → {args.out_json}")
203
```

bsincomevalidation/src/halyk_ind/parser.py:55

Level Medium

Status Not processed

```
52 from src.utils.path_security import validate_path
53 validated = validate_path(jsonl_path)
54 pages: List[Dict[str, Any]] = []
55 with open(validated, "r", encoding="utf-8") as f:
```

```
56    for line in f:
57        line = line.strip()
58        if not line:
```

bsincomevalidation/src/kaspi_pay/parser.py:42

Level Medium

Status Not processed

```
39 from src.utils.path_security import validate_path
40 validated = validate_path(path)
41 pages: List[Dict[str, Any]] = []
42 with open(validated, "r", encoding="utf-8") as f:
43    for line in f:
44        line = line.strip()
45        if not line:
```

bsincomevalidation/src/utils/convert_pdf_json_page.py:208

Level Medium

Status Not processed

```
205        out["XRef"] = xref
206
207 validated = validate_path_for_write(out_path, out_dir if not args.out else _PROJECT_ROOT)
208 with open(validated, "w", encoding="utf-8") as f:
209    json.dump(out, f, ensure_ascii=False, indent=2)
210
211 print(f"Dumped to {out_path}")
```

bsincomevalidation/src/utils/convert_pdf_json_pages.py:65

Level Medium

Status Not processed

```
62       words_per_page.append(words)
63
64 # 2) Raw content streams (pikepdf)
65 with pikepdf.open(str(pdf_path)) as pdf, open(out_path, "w", encoding="utf-8") as out:
66     for i, page in enumerate(pdf.pages):
67         contents = page.get("/Contents", None)
68         raw_bytes = b""
```

bsincomevalidation/src/utils/path_security.py:111

Level Medium

Status Not processed

```
108     ValueError: If path is invalid or escapes base_dir
109 """
110 validated_path = validate_path(file_path, base_dir)
111 return open(validated_path, mode, **kwargs)
```

WAF Setup Instructions

Persistent XSS

Description

Persistent XSS or server XSS attack is possible.
Cross-site scripting is one of the most common types of attacks on web applications. XSS attacks take seventh place in the “OWASP Top 10 2017” list of ten most significant vulnerabilities in web applications.
The main phase of any XSS attack is an imperceptible for the victim execution of a malicious code in the context of the vulnerable application. For this purpose, the functionality of the client application (browser) is used that allows to automatically execute scripts embedded in web page code. In most cases, these malicious scripts are implemented in JavaScript.
Consequences of an XSS attack vary from violations of application functionality to complete loss of user data confidentiality. The malicious code during the XSS attack can steal user HTTP-cookie, which gives an attacker the ability to make requests to the server on behalf of the user.
OWASP suggests the following classification of XSS attacks:

- Server type XSS occurs when data from an untrusted source is included in the response returned by the server. The source of such data can be both user input and server database (where it had been previously injected by an attacker who exploited vulnerabilities in the server-side application).
- Client type XSS occurs when the raw data from the user input contains code that changes the Document Object Model (DOM) of the web page received from the server. The source of such data can be both the DOM and the data received from the server (e.g., in response to an AJAX request).

Typical server type attack scenario:

1. Unvalidated data, usually from a HTTP request, gets into the server part of the application.
2. The server dynamically generates a web page that contains the unvalidated data.
3. In the process of generating a web page, server does not prevent the inclusion of an executable code that can be executed in the client (browser), such as JavaScript code language, HTML-tags, HTML-attributes, Flash, ActiveX, etc., in the page code.
4. The victim’s client application displays the web page that contains the malicious code injected via data from an untrusted source.
5. Since malicious code is injected in the web page coming from the known server, the client part of the application (browser) executes it with the rights set for the application.
6. This violates the same-origin policy, according to which the code from the one source must not get an access to resources from another source.

Client type attacks are executed in a similar way with the only difference that the malicious code is injected during the phase of the client application work with the document object model received from the server.

Vulnerability Entries

1. bsincomevalidation/src/ui/db_viewer.html:408
2. bsincomevalidation/src/ui/db_viewer.html:450
3. bsincomevalidation/src/ui/db_viewer.html:483

Information leak

Description

System configuration information leak is possible. This can help an attacker to create a plan of an attack. Debug information and error messages can be written to the log, displayed to the console, or sent to the user depending on the system settings. In some cases, an attacker can make a conclusion about the system vulnerabilities from the error message. For example, a database error can indicate insecurity against SQL injection attacks. Information about the version of the operating system, server applications and system configurations can also be of value to the attacker.

Vulnerability Entries

1. bsincomevalidation/src/bcc/footer.py:211#215
2. bsincomevalidation/src/bcc_ind/parser.py:341
3. bsincomevalidation/src/halyk_business/footer.py:193
4. bsincomevalidation/src/halyk_ind/parser.py:321
5. bsincomevalidation/src/halyk_ind/parser.py:324
6. bsincomevalidation/src/halyk_ind/parser.py:328
7. bsincomevalidation/src/utils/income_calc.py:296#300
8. bsincomevalidation/src/utils/income_calc.py:309#313
9. bsincomevalidation/src/utils/income_calc.py:322#326

Resource injection

Description

An attacker can gain access to reading and changing protected system resources of the application has the ability to change the resource identifier.
Resource injection occurs when an atacker can specify identifier that will be used to access the system resource (for example, the port number to connect to the network resource). This allows him, in particular, to transfer valuable data to thied-party server.

Vulnerability Entries

1. bsincomevalidation/src/alatau_city_bank/batch_pars
2. bsincomevalidation/src/alatau_city_bank/batch_pars
3. bsincomevalidation/src/api/app.py:843
4. bsincomevalidation/src/api/storage.py:134
5. bsincomevalidation/src/api/storage.py:149
6. bsincomevalidation/src/api/storage.py:157
7. bsincomevalidation/src/api/storage.py:167
8. bsincomevalidation/src/bcc/batch_parse.py:84
9. bsincomevalidation/src/bcc/batch_parse.py:179
10. bsincomevalidation/src/bcc/footer.py:189
11. bsincomevalidation/src/bcc/parser.py:40
12. bsincomevalidation/src/eurasian_bank/batch_parse.
13. bsincomevalidation/src/eurasian_bank/batch_parse.
14. bsincomevalidation/src/forte_bank/batch_parse.py:115
15. bsincomevalidation/src/forte_bank/batch_parse.py:192
16. bsincomevalidation/src/forte_bank/footer.py:195
17. bsincomevalidation/src/freedom_bank/batch_parse.
18. bsincomevalidation/src/freedom_bank/batch_parse.
19. bsincomevalidation/src/halyk_business/footer.py:143
20. bsincomevalidation/src/halyk_business/footer.py:158
21. bsincomevalidation/src/halyk_business/footer.py:200
22. bsincomevalidation/src/halyk_ind/parser.py:55
23. bsincomevalidation/src/kaspi_pay/parser.py:42
24. bsincomevalidation/src/utils/convert_pdf_json_page.
25. bsincomevalidation/src/utils/convert_pdf_json_pages
26. bsincomevalidation/src/utils/path_security.py:111

Scan Settings

2/2 2026-02-12 05:41:56

Select files for Analysis

/*; !/*.json; !**/*.resx;

Languages

- ABAP
- Apex
- [✓ C#]
- C/C++
- COBOL
- [✓ Config files]
- [✓ Dart]
- [✓ Go]
- Groovy
- [✓ HTML5]
- [✓ Java, Scala, Kotlin]
- [✓ JavaScript, TypeScript]
- LotusScript
- Objective-C
- Delphi, Pascal
- [✓ PHP]
- [✓ PL/SQL, T-SQL]
- [✓ Python]
- Perl
- [✓ Ruby]
- [✓ Rust]
- [✓ Solidity]
- [✓ Swift]
- TSX
- VB.NET, VBA, VBScript, VB6
- Vyper
- 1C

Java/Scala/Kotlin Specific Settings

- [✓ Prebuilt project with .class files]
- Source code (will be built by DerScanner)
- Source code (no build needed, only for Java)
- Source code (build with own tools)

JavaScript Specific Settings

- Analyze standard libraries

General Analysis Settings

- ✓ Analyze libraries and nested archives
- ☐ Use extra rules
- ✓ Incremental analysis
- ✓ Preprocessing

Source Code Charset	UTF-8
Filename Charset	UTF-8

Rule Sets —

03 Export Settings

Project Information

Project Statistics

- [x] Dynamics by rating
- [x] Dynamics by vulnerabilities

Scan History

- ○ Do not export scan history
- ● Export entire scan history
- ○ Export the latest scans ...

Vulnerability Classification

By severity

Scan Information

- [x] Detected vulnerabilities chart
- [x] Vulnerable Dependencies
- [x] Statistics on languages
- [ ] Analyzed files statistics
- [x] Error information
- [x] Scan settings
- [x] Version Properties

Vulnerability Filter

Severity

- [x] Critical
- [x] Medium
- [ ] Low
- [ ] Info

Vulnerable Dependencies

- [x] In standard libraries

- ✓ In .class files that cannot be decompiled
- ✓ Vulnerabilities without WAF configuration guide

Languages

✓ ABAP	✓ Java	✓ Scala
✓ Android	✓ JavaScript	✓ Solidity
✓ Apex	✓ Kotlin	✓ Swift
✓ C#	✓ LotusScript	✓ T-SQL
✓ C/C++	✓ Objective-C	✓ TSX
✓ COBOL	✓ Pascal	✓ TypeScript
✓ Config files	✓ PHP	✓ VB.NET
✓ Dart	✓ PL/SQL	✓ VBA
✓ Delphi	✓ Python	✓ VBScript
✓ Go	✓ Perl	✓ Visual Basic 6
✓ Groovy	✓ Ruby	✓ Vyper
✓ HTML5	✓ Rust	✓ 1C

Vulnerability List

Vulnerability Statuses

- ✓ Not processed
- ✓ Confirmed
- ☐ Rejected

List of Vulnerability Entries

- ○ Do not export
- ● Export all entries
- ○ Export no more than entries ...

Analysis Results

Vulnerability Statuses

- ✓ Not processed

- [x] Confirmed
- [ ] Rejected

Vulnerability Entries

- ○ Do not export
- ● Export all entries
- ○ Export no more than entries ...

Source Code

- ○ Do not export source code
- ○ Export entire vulnerable source code file
- ● Export context in the number of lines of code 3

Trace

- ○ Do not export trace items
- ○ Export all items
- ● Export only the first and last items

Additional information

- [x] Vulnerability comment
- [x] Vulnerability actions

WAF Setup Instructions

Guide for Vulnerability Statuses

- [x] Not processed
- [x] Confirmed
- [ ] Rejected

Guide for WAF

- [x] Imperva SecureSphere
- [x] ModSecurity
- [x] F5

General Report Settings

- ✓ Table of contents
- ✓ Report export settings
- ✓ Display vulnerability statuses
- ✓ Display results correlation tags