

Code Analysis in the SAP Customer Namespace and AI-Supported Security: Identifying Risks, Strengthening Compliance

Christoph Wildensee

As early as 2009, Wiegenstein, Schumacher, Schinzel and Weidemann published the book "Secure ABAP Programming" (SAP Press / Galileo Press)¹, providing a comprehensive guide for building the skills required to identify insecure SAP programming. This concerns both the authorization layer and the actual programming of functionalities that are widely required within organizations. These, however, may also be developed and implemented in such a way that operational security risks arise.

Until now, the focus has primarily been on static code analysis (SCA), i.e. finding certain terms from the statement set and, where applicable, variable anomalies. Unlike fixed statements, variables can be used in multiple ways and reshaped, meaning that sources of error are not so easy to detect. Even the uploading of externally maintained source code — even if only as a simple text file — in order to execute it via temporary storage, with real SAP table changes, and then to let it disappear into oblivion afterwards without logging, already shows that such constructs must be subject to particular scrutiny. The temporary approach is, however, a problem that cannot sensibly be addressed by point-in-time evaluations. Only the use of professional development tools that cannot be disabled during the development phase, that provide indications of problematic programming and also document this, and that may even provide a workflow for approving such code parts, can offer a remedy. Such an approach is referred to as *Static Application Security Testing* (SAST). Examples of relevant providers include SecurityBridge, which offers automated static and dynamic scans directly in the SAP system and in Eclipse — similar to spell-checking for code — and Onapsis, which likewise analyses ABAP code for security vulnerabilities, misconfigurations and compliance violations. Lastly, Xiting should not be omitted with regard to integrating authorization considerations into development. The **traditional static code analysis**, often performed via transactions such as CODE_SCANNER (AFX_CODE_SCANNER) and ABAP Test Cockpit (ATC; based on the SAP Code Inspector SCI with test catalogs), has a **dubious reputation among developers and internal auditors alike. In practice it is often avoided rather than appreciated**, chiefly due to the rigid, rule-based mode of operation of traditional tools. These scans regularly produce an overwhelming number of results which, however, frequently turn out to be 'false positives' — i.e. false alarms that formally violate a rule but do not represent a real risk in the specific application context. For employees this means an enormous amount of manual effort in order to identify the truly critical weaknesses within a flood of incidental findings. In addition, the entry barriers are high: operating and correctly configuring these expert functions requires specialized knowledge that is not available across many teams. The result is a time-consuming analysis that highlights issues but leaves users alone — without concrete remediation ideas.

¹ Original publication: „Sichere ABAP-Programmierung“, 2009, SAP Press, ISBN 978-3-8362-1357-8.

Nevertheless, with a report such as ZABAPDOWN01, which is freely available on the internet, it is at least possible to download the entire source code of a system via transactions such as SA38, OODR and others and to provide it as a text file. Once it is available as a file — containing the three data fields NAME (program name), LINE (line within a program) and CODE (code line with one or more statements, variable assignments, etc.) — it can be stored in an MS Access database under the system name.

AI as a Sparring Partner in Internal Audit

In the complex world of SAP productive systems, customer-specific source code (Y and Z namespace) often represents a large ‘black box’ for internal audit. Where standard processes are safeguarded by SAP’s own controls, individual adaptations open the door to process anomalies, security vulnerabilities and breaches of regulatory requirements such as the GoBD (Principles for the correct management and storage of accounting records and documents in electronic form, as well as for data access by supervisory authorities) and / or GDPR. Without specialized tools for dynamic code analysis (DCA) in the development process, manual review with static code analysis (SCA) becomes a Sisyphean task.

Here, artificial intelligence (AI) does not appear as a replacement but as a communicative, largely ‘knowledgeable’, highly efficient, high-level sparring partner for an internal auditor — someone whom the auditor may deploy at their own discretion — at least provided that the auditor formulates the requirements and conditions in a targeted and comprehensive manner. Such willingness will become ever more important in the future.

The Role of AI: From Pattern Matching to Semantic Analysis

Previous manual reviews were often limited to searching for risk-bearing keywords (e.g. EXEC SQL, EDITOR-CALL, INSERT REPORT), hence also described as static analysis. An AI goes a step further: it ‘understands’ the context, provided the program source code is supplied comprehensively within token limits.

- Logic understanding: Whereas a simple search only notices the presence or absence of a statement, AI can recognize whether an authorization check is in the wrong place or can be bypassed by flawed IF logic or an incorrect handling of return codes (SY-SUBRC).
- Identification of data leakage: AI can trace data flows within the program source code. It recognizes whether personal data (GDPR) are output unmasked in reports or exported to insecure local directories.
- Compliance mapping: AI can check source code directly against regulatory frameworks. For example, it can identify whether posting logic jeopardizes immutability under the GoBD by detecting direct table access (MODIFY, UPDATE, TRUNCATE on SAP standard / system tables) instead of official BAPIs.

The Internal Auditor and AI: A Cooperative Tandem

AI functions as a ‘first line filter’. The internal auditor feeds the AI with code extracts and, in return, receives a qualified risk assessment. The auditor can ask follow-up questions as

often as desired; the AI never gets irritated and is limited only by the manner in which the auditor communicates with it.

- Efficiency gains: AI filters out ‘noise’. The auditor does not have to read 1,000 lines of code, but focuses on the few passages flagged by the AI that actually represent a risk to properness and compliance (e.g. under HGB [Commercial Code / Law / Act], GoBD).
- Knowledge transfer: AI explains why a piece of coding is problematic. This provides the auditor with argumentative support vis-à-vis the business function or development in order to derive measures cooperatively.
- Objectivity: AI checks each program fragment with the same meticulousness — free from time pressure, operational blindness or other limiting factors.

Expected Outcomes: Quality Assurance at a New Level

The use of AI in code audit provides measurable added value:

- Higher audit density: substantially more systems and programs can be reviewed in a shorter time, increasing the overall security of the system landscape.
- Reduction of ‘false positives’: by understanding semantic relationships, AI delivers more precise hits than purely rule-based scripts.
- Early-warning system for departmental ICS: if AI is also used analogously in the responsible business function, it can uncover weaknesses in the internal control system (ICS) when it recognizes that reports systematically bypass controls.
- Audit-proof documentation: the explanations generated by AI can be incorporated into the audit report; findings become transparent for non-developers.

The final decision on criticality remains with the internal auditor. For SAP audit without specialized SCA and / or DCA tools, AI is the decisive lever to move from sample-based review to a deep, risk-oriented analysis of the entire system landscape. It is the sparring partner that not only finds the needle in the haystack but also explains why it must not be there.

Possible Approach to a Supported Analysis

The following program design shows the structure of a categorized code analysis based on certain statements in source code as SCA. On the left, the individual queries and their meanings can be seen. To the right are the systems to be reviewed; the code lines to which the analyses apply are stored there. Thus, these are distinct source-code tables with millions of code lines per system, each analyzed in turn. These can be all systems with customer modifications, processed one after another. In the lower area, results are shown as a grid / table once a query has been executed for a given system.

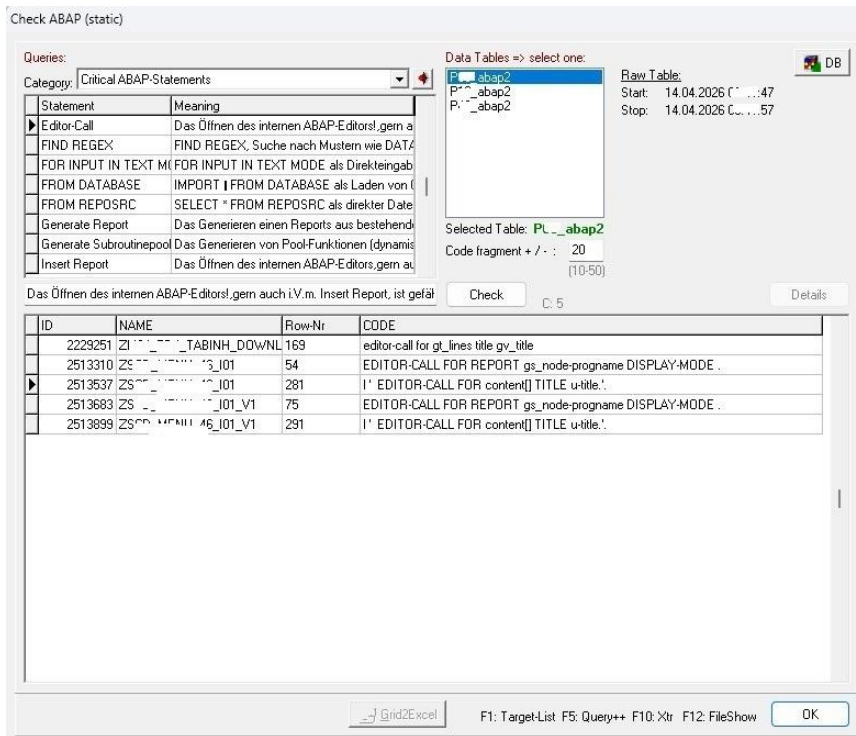


Fig. 1: Hit list of analysis.

If one selects the highlighted entry in the hit list — here ZSC... (row no. 281) — one can view an excerpt of the source code via the Details button, namely ± 10 lines above and below the searched keyword, here EDITOR-CALL. Under Code fragment \pm one can also enter values between 10 and 50, so that at least 20 and at most 100 lines of source code can be inspected. Even at this stage one can estimate whether this is valid problematic code, or whether it is not used because it has, for example, been functionally ‘disabled’.

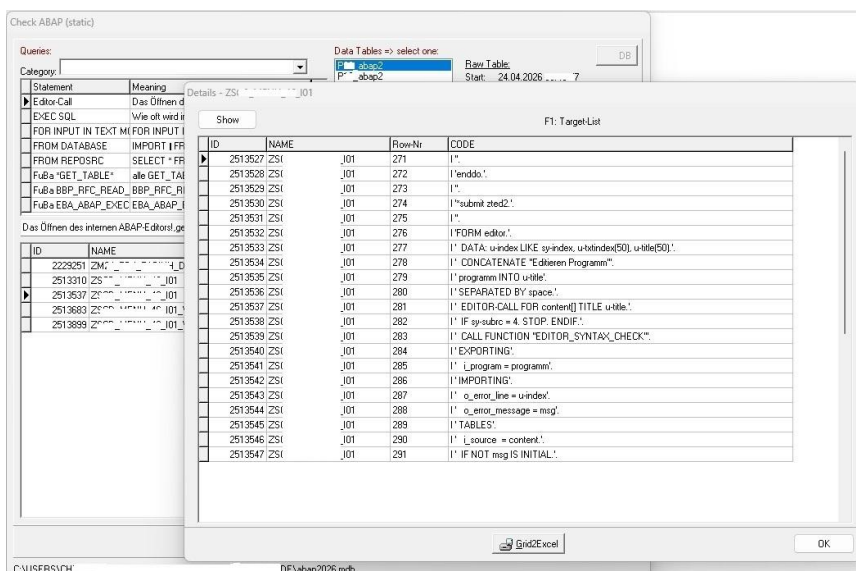


Fig. 2: Details of the selected program; excerpt from the code.

If one recognizes that the code is critical, one can view the target list via F1 and add the program name — together with system assignment and the indication of the list's criticality trigger — to the list.

Target List Mng.

ID	SYSTEM	NAME	QUERY
15	P	_abap2	ZGR.
95	P	_abap2	ZGR.
12	P	_abap2	ZGR.
94	P	_abap2	ZGR.
13	P	_abap2	ZGR.
148	P	_abap2	ZGR.
149	P	_abap2	ZGR.
14	P	_abap2	ZGR.
150	P	_abap2	ZGR.
31	P	_abap2	ZISU
16	P	_abap2	ZM2:
17	P	_abap2	ZM2:
1	P	_abap2	ZM2:
18	P	_abap2	ZM2:
106	P	_abap2	ZNJ_
93	P	_abap2	ZRFF
4	P	_abap2	ZRFF
8	P	_abap2	ZRFF
9	P	_abap2	ZSCI
2	P	_abap2	ZSCI

Fig. 3: Target list.

Once completed, this target list can be transferred to Excel and further processed there. After another verification of results, it is marked whether the program source code is necessary, or whether it should be deleted or severely restricted in terms of access.

In addition, further evaluations can be executed which combine two criteria with AND or OR within the same line.

ABFRAGE	SQLTEXT	FIELD1	FILTER1	AND/OR	FIELD2	FILTER2	SC	INAKTIV
OO Editor Call / Insert Report => *CL_ABAP_SOURCE* OR *CL_OO_FACTORY*	SELECT * FROM #	CODE LIKE	*CL_ABAP_SOURCE*	OR	CODE LIKE	*CL_OO_FACTORY*	Y	<input type="checkbox"/>
OO Editor Call / Insert Report => *CL_ABAP_COMPILER* OR *RS_INSERT_REPORT*	SELECT * FROM #	CODE LIKE	*CL_ABAP_COMPILER*	OR	CODE LIKE	*RS_INSERT_REPORT*	Y	<input type="checkbox"/>
OO Editor Call / Insert Report => *CL_GUI_TEXTEDIT* OR *CL_ABAP_COMPILER_FRONTEND*	SELECT * FROM #	CODE LIKE	*CL_GUI_TEXTEDIT*	OR	CODE LIKE	*CL_ABAP_COMPILER_FRONTEND*	Y	<input type="checkbox"/>
OO Editor Call / Insert Report => *EDITOR_PROGRAM* OR *abaptxt*	SELECT * FROM #	CODE LIKE	*EDITOR_PROGRAM*	OR	CODE LIKE	*abaptxt*	Y	<input type="checkbox"/>
OO Editor Call / Insert Report => *go_editor* OR *set_text_as_stream*	SELECT * FROM #	CODE LIKE	*go_editor*	OR	CODE LIKE	*set_text_as_stream*	Y	<input type="checkbox"/>
REPORT oder FUNCTION	SELECT TOP 100 * I	CODE LIKE	REPORT*	OR	CODE LIKE	*FUNCTION*	Y	<input type="checkbox"/>

Fig. 4: Example of field constellation for AND/OR.

If the list of previous standard queries as well as these combined questions does not allow the desired question to be asked of the database, a search can also be started by specifying a partial string — first under F5 and there under F1 (Search [Part-String]). Results are displayed, which can likewise be added to the target list (last search of part string is stored).

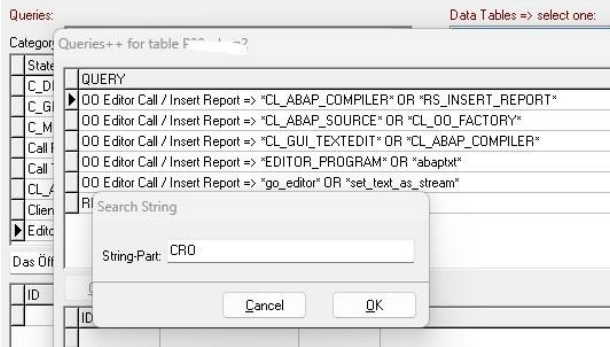


Fig. 5: Arbitrary part-string search.

Any Number of Databases

Figure 2 shows, at the bottom left, a path to a database in which the source-code lines are stored. To avoid the mixing of development, test/integration, and production databases, any number of databases can be created and analyzed simultaneously. The target lists are then likewise separated. This is advantageous in that the database limit of each Access database is thereby lifted (mdb, not accdb). Selection is made via the DB button at the top right.

However, the program does not end here.

AI Integration

If one is unsure whether the code fragment actually contains problematic code or behaves as feared, one selects F10 (Xtr) from the initial display of the identified problem code; this opens the complete source code of the selected program. If an analysis has already been created for this program in the past, it can be retrieved via the Load button. If not — or if a new analysis is intended — an AI-based analysis can be performed via the Analyze button.

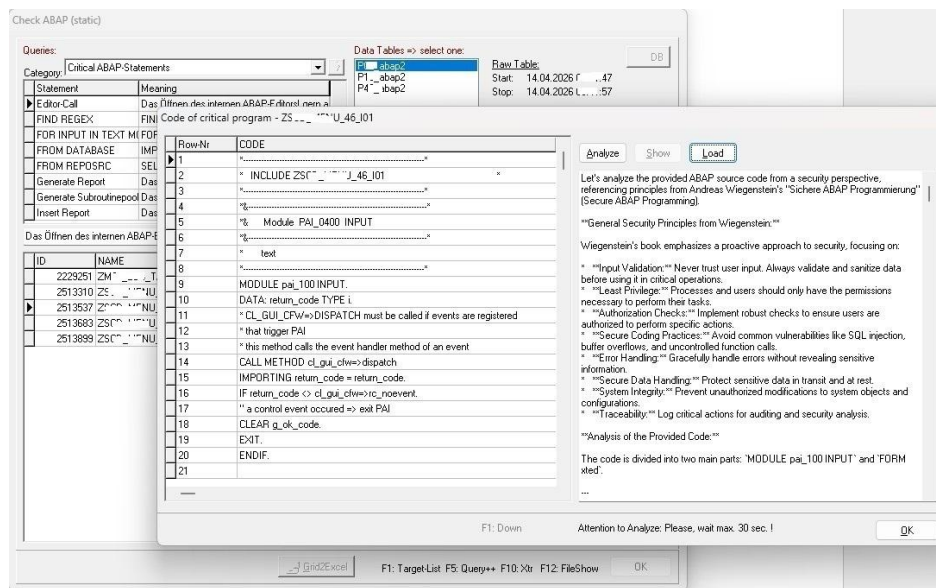


Fig. 6: Entire source code of the selected program, followed by AI assessment.

After starting the AI request via the Analyze button, a connector is launched which establishes an API connection to the AI and sends the question — together with the selected program's source-code data — **to the AI**. The AI processes the request and sends a response back to the connector. The connector provides the AI-generated response to the calling program and then closes. The calling program waits for the connector to close, takes over the response when the connector closes and then displays it in the window.

The first question asked is: 'Analyze the source code generally from security aspects, especially according to those of Andreas Wiegenstein, who wrote the book +Sichere ABAP Programmierung+. Represent the problematic parts with code snippets. =>', with the selected source code added to the question.

Via F1 (Down) the window content can be saved under c:\temp as <system>-<program name>.txt; the AI-generated results are stored and examined later. Helpful is the fact that the AI results iteratively contain pointers to further statements or constructs that are subject to criticism, which subsequently expand the list of static queries.

Any Number of Questions to the AI

It is also useful to determine whether further communication about the case can take place with the AI system. The internal auditor can ask any number of individual questions, and communication with the AI is established automatically.

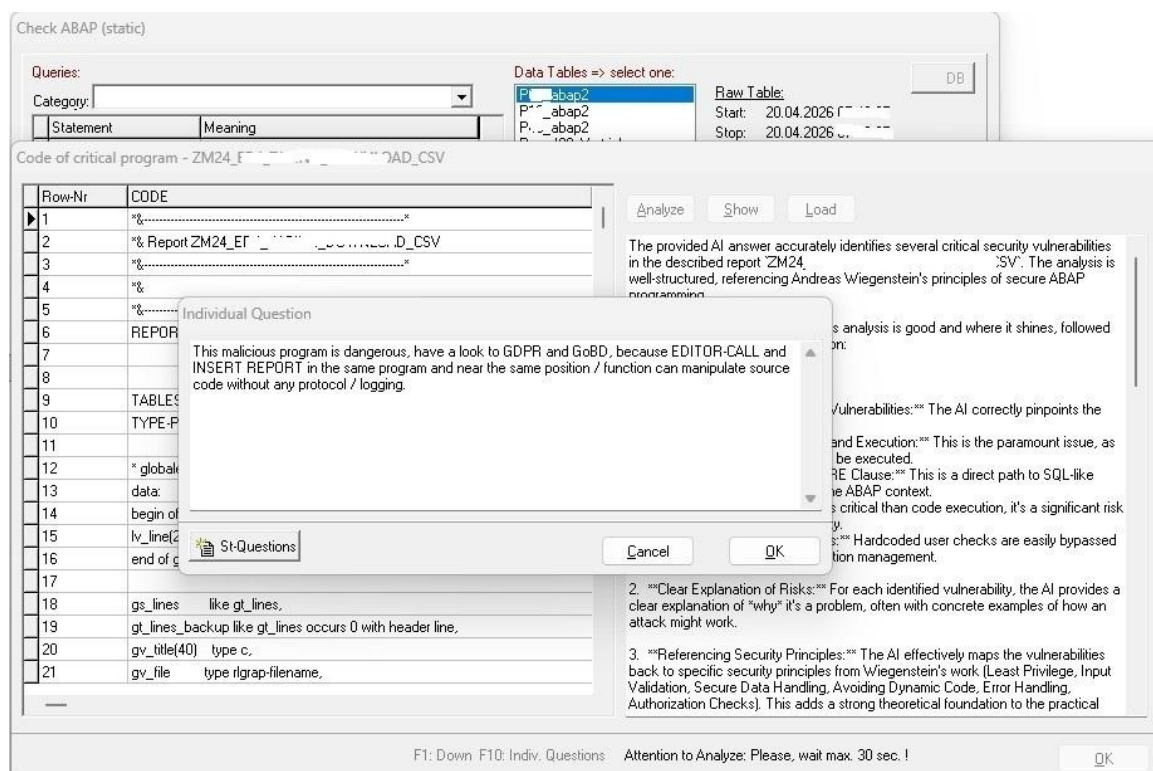


Fig. 7: After the first standard question, option for individual communication with the AI.

With OK, the question is sent to the AI. If the question has appeared for the first time, it can also be stored as a standard question and retrieved again for each further program-related analysis; this saves typing effort.

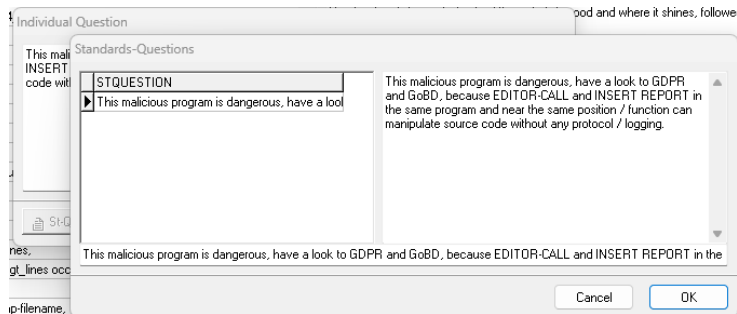


Fig. 8: Standard questions that recur can be retrieved and executed.

The results of each individual analysis per program source code are stored as simple text files. However, they can also be generated by the AI as often as desired. Old versions are not overwritten but historized, so as not to lose approaches for expanding the analysis portfolio and to incorporate all aspects of previous analyses. If the files are stored in a specific directory, this can be selected and marked as the default reading directory.

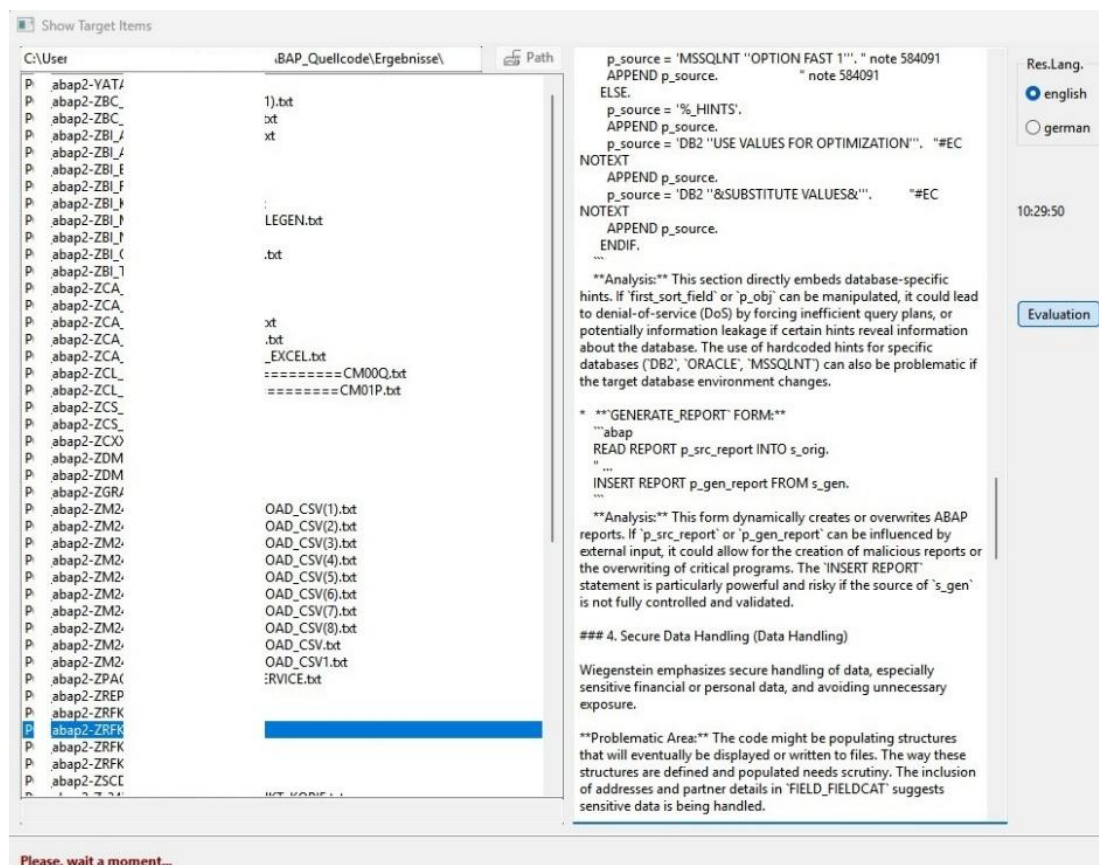


Fig. 9: View of historized AI query results (excerpt).

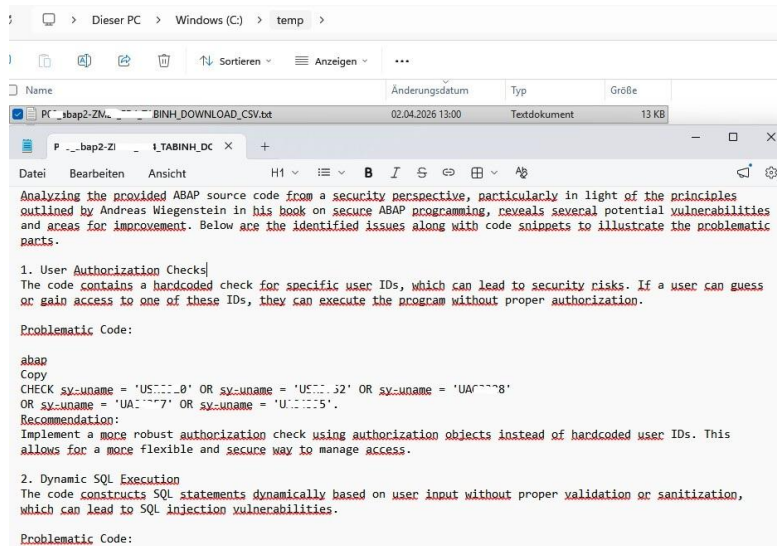


Fig. 10: Provided text file from the AI in File Explorer for the selected program and system.

Data Protection

Data protection is essential in this context. Providing source code to an AI — where the requester cannot fully pre-qualify it due to its scope, neither in terms of business content nor from a data-protection perspective — entails a high risk of leakage of personal data, because source code may contain references in multiple places to user names, user IDs, e-mail addresses, etc. Especially with mass deliveries of code lines, it is hardly reasonable to assume that an internal auditor can first check the GDPR compliance for each individual case, line, or position and take responsibility for it. It is therefore imperative that a GDPR-compliant implementation of bilateral data provision is in place. If such access exists, even if it may be slightly throttled in speed compared with freely accessible providers from the USA or China, nothing stands in the way of GDPR-compliant continuous use.

AI	Communication attributes of the specific connectors	GDPR-compliant
Groq	Adresse=https://api.groq.com/openai/v1/chat/completions Modell=llama-3.3-70b-versatile APIKey=gsk_geWfQKH...	No
Gemini	Adresse=https://generativelanguage.googleapis.com/v1/models Modell=gemini-2.5-flash-lite APIKey=AIzaSyBEeDv....	No
DeepSeek	Adresse=https://api.deepseek.com/chat/completions Modell=deepseek-reasoner APIKey=sk-5126cb...	No
Managed Private Cloud KI / Dedicated SaaS KI	Adresse=https://izzzzzzz-0000000.cognitiveservices.azure.com/ Modell=gpt-5.1-0000000 APIKey=hsdgchjs... APIVersion=2024-12-01-preview	Yes

```

22.04.2026 11:34: - Site: https://api.deepseek.com/chat/completions
22.04.2026 11:34: - Modell: deepseek-reasoner
22.04.2026 11:34: - APIKey: sk-5126cb78.....7c538b
22.04.2026 12:16: - Site: https://api.deepseek.com/chat/completions
22.04.2026 12:16: - Modell: deepseek-reasoner
22.04.2026 12:16: - APIKey: sk-5126cb78.....7c538b
22.04.2026 13:32: - Answer stored.
22.04.2026 16:17: - Answer stored.
22.04.2026 17:39: - Site: https://api.deepseek.com/chat/completions
22.04.2026 17:39: - Modell: deepseek-reasoner
22.04.2026 17:39: - APIKey: sk-5126cb78.....7c538b
22.04.2026 19:54: - Answer stored.
22.04.2026 24:30: - Site: https://api.deepseek.com/chat/completions
22.04.2026 24:30: - Modell: deepseek-reasoner
22.04.2026 24:30: - APIKey: sk-5126cb78.....538b
22.04.2026 27:37: - Site: https://api.deepseek.com/chat/completions
22.04.2026 27:37: - Modell: deepseek-reasoner
22.04.2026 27:37: - APIKey: sk-5126cb78cf.....7c538b
22.04.2026 29:50: - Answer stored.
22.04.2026 30:22: - Answer stored.
23.04.2026 06:43:10 - Site: https://i...y-0000000.cognitiveservices.azure.com
23.04.2026 06:43:10 - Modell: gpt-5.1-0000000
23.04.2026 06:43:10 - APIKey: 6e1m57zk.....AAACOG0Tcm
23.04.2026 06:43:10 - APIVers: 2024-12-01-preview
23.04.2026 06:43:40 - Answer stored.

```

Result Summaries

C:\Users\d	O_ABAP_Queelcode\Ergebnisse\	Evaluation Path
p_i_abap2-VAT)_txt	p_source = MSSQLNT "OPTION FAST !!!," note S84091 APPEND p_source. ELSE.
p_i_abap2-ZBC)_txt	p_source = %_HINTS'.
p_i_abap2-ZBC	_it	APPEND p_source. p_source = DB2 "USE VALUES FOR OPTIMIZATION"', *EC
p_i_abap2-ZBI	.	NEXTX
p_i_abap2-ZBI	.	APPEND p_source.
p_i_abap2-ZBI	.EGEN.txt	p_source = DB2 "&SUBSTITUTE VALUES&", **EC
p_i_abap2-ZBI	.	NEXTX
p_i_abap2-ZBI	btt	APPEND p_source. ENDIF.
p_i_abap2-ZCJ	.	**Analysis** This section directly embeds database-specific hints. If hint_son_fields or p_obj can be manipulated, it could lead to denial-of-service (DoS) by forcing inefficient query plans, or potentially information leakage if certain hints reveal information about the database. The use of hardcoded hints for specific databases (DB2*, ORACLE*, MSSQLNT*) can also be problematic if the target database environment changes.
p_i_abap2-ZCJ	t	* **GENERATE_REPORT FORM** **abap
p_i_abap2-ZCJ	tbl_EXCEL.tbl	READ REPORT p_src_report INTO s_orig.
p_i_abap2-ZCL	=====CM0QO.txt	--
p_i_abap2-ZCL	=====CM0IP.txt	INSERT REPORT p_gen_report FROM s_gen.
p_i_abap2-ZCS	.	**Analysis** This form dynamically creates or overwrites ABAP reports. If p_src_report or p_gen_report can be influenced by external input, it could allow for the creation of malicious reports or the overriding of critical programs. The INSERT REPORT statement is particularly powerful and risky if the source of s_gen is not fully controlled and validated.
p_i_abap2-ZCS	.	### 4. Secure Data Handling (Data Handing)
p_i_abap2-ZCO	.	Wiegstein emphasizes secure handling of data, especially sensitive financial or personal data, and avoiding unnecessary exposure.
p_i_abap2-ZDN	2AD_CSV(1).txt	**Problematic Area:** The code might be populating structures that will eventually be displayed or written to files. The way these structures are defined and populated needs scrutiny. The inclusion of addresses and partner details in FIELD_FIELDCAT suggests sensitive data is being handled.
p_i_abap2-ZDN	2AD_CSV(2).txt	
p_i_abap2-ZGF	2AD_CSV(3).txt	
p_i_abap2-ZM	2AD_CSV(4).txt	
p_i_abap2-ZM	2AD_CSV(5).txt	
p_i_abap2-ZM	2AD_CSV(6).txt	
p_i_abap2-ZM	2AD_CSV(7).txt	
p_i_abap2-ZM	2AD_CSV(8).txt	
p_i_abap2-ZM	2AD_CSV.txt	
p_i_abap2-ZM	2AD_CSV1.txt	
p_i_abap2-ZP	RVICE.txt	
p_i_abap2-ZRE	.	
p_i_abap2-ZRF	.	
p_i_abap2-ZRF	.	
p_i_abap2-ZSC	.	

Please wait a moment...

F1: Download Close

April 2026

[illegible]

Final delivery of results — with all program names as a consolidated overview — is provided in tabular form. The short summaries can be created as often as desired, and the auditor can use the best formulations for report writing. An example result could look as follows:

Fig. 14: Possible result (excerpt).

Further Analyses

The results of the target list(s) and the short results presentation ultimately contain the system and program names of the programs and function modules in the customer namespace that are initially identified as critical. Program names can be reviewed using SAP tables TADIR (catalogue of repository objects) and TRDIR (system table) and transaction SE38 (code and attributes). The fields of table TADIR are OBJECT and OBJ_NAME as well as DEVCLASS (this refers to authorization object S_PROGRAM with fields P_GROUP and P_ACTION in the authorization master records in combination with S_TCODE and calling transactions such as SA38, OODR, etc.). The fields of table TRDIR are CNAM, CDAT, UNAM, UDAT, IDATE and SUBC (type). For function modules, transaction SE37 and table TFDIR (function modules) apply, with fields FUNCNAME, PNAME, PNAME_MAIN and STTEXT. Another perspective is the execution of source code transports through the systems (check of objects S_TCODE, S_PROGRAM, S_TRANSPRT). DEVCLASS is relevant to audit in so far as, if access for calling Y- / Z- programs is spread too widely and little classification and limitation is in effect, there is almost no steering effect, and too many employees are allowed to run customer-namespace programs. This should be prevented. Many programs also control execution depending on the calling user ID, SY-UNAME. If this is the chosen and common way of allowing or blocking program execution, it means that these controls continue to apply when people leave the function and assume other tasks — or even leave the company. When new colleagues are to take over the tasks, access may then no longer exist. This is one of the most common problem areas in this context. The need for adjustment is considerable.

Conclusion

Integrating AI into data generation and enrichment in internal audit engagements is not only possible but advisable. AI-generated texts can be structured and analyzed separately in order to confirm previous results, improve the body of insights and expand the basis of queries. In addition, the entire system landscape can be integrated into the audit without significant additional effort. Source-code audits are inherently important due to developers' creativity, but especially due to the immense use of external consultants and implementers in-house and regularly required process extensions in SAP. Therefore, efforts to identify and eliminate 'malicious' code should be repeated at least every three years. It is thus a particularly suitable topic for establishing continuous auditing. This applies, moreover, to any form of code review. Professional tools are expensive and not always desired in-house. The company itself offers AI models with GDPR-compliant use and significantly higher usage limits. Internal audit should use these permanently, strengthen its own capacities, and expand its competencies in order to independently design such solutions. And the application is infinitely expandable. Ultimately, the objective is to ensure compliance with the regulatory requirements defined in the developer guidelines (e.g., separate role assignments for REQUIREMENT / DEVELOPMENT / TESTING / TECHNICAL APPROVAL / TRANSPORT, command exclusion lists...) and that highly privileged access rights are properly managed. This also eliminates the unspoken accusation that developers are abusing their special privileges.

[For stylistic reasons and to simplify reading, the masculine form is occasionally used. The feminine form is, of course, equally intended and should be borne in mind.]

References

Wiegenstein et al (2009): Sichere ABAP-Programmierung, SAP Press, 2009.

Wildensee (2016): Gefährdung der Ordnungsmäßigkeit rechnungslegungsrelevanter SAP-Systeme durch irreguläre Quellcode-Transfers,
<https://www.risknet.de/elibrary/paper/gefaehrdung-der-ordnungsmaessigkeit-rechnungslegungsrelevanter-sap-systeme-durch-irregulaere-quellcode-transfers> .

Wildensee (2026): Project SCA; <http://www.wildensee.de/sca.zip> .

Appendix

Technical elements (Lazarus; uses ... fphttpclient, opensslsockets, fpjson, jsonparser)

The connector is called in the administrative program as follows:

```
[...]
if (anzds1 > 0) then // number of source-code records of the selected program
begin
// call connector here if present in the main program path
...
if
(FileExists(Uppercase(PChar(ExtractFilePath(ParamStr(0))+ 'ABAPSTA1.EXE'))))
then
begin
...
ShellExecAndWait(ExtractFilePath(ParamStr(0))+ 'ABAPSTA1.EXE ', '',
SW_SHOWMINIMIZED);
end;
...
end;
[...]
// ShellExecAndWait ensures that the main program continues only after the
// connector abapsta1.exe has terminated again after communication.
[...]
```

Connector Azure:

```
function TForm1.SendToAI(const Frage: string): string;
var
  HTTPClient: TFPHttpClient;
  ResponseStream: TStringStream;
  RequestBodyStream: TMemoryStream;
  FullURL, JsonBody, SafeFrage: string;
  UTF8Bytes: RawByteString;
begin
  Result := '';
  // build URL without risk of double slashes
```

```

FullURL := Trim(Self.Site1) + '/openai/deployments/' + Trim(Self.Model1) +
'/chat/completions?api-version=' + Self.APIVers;
// working JSON body
SafeFrage := Frage.Replace('\', '\\').Replace('"', '\"').Replace(#13,
'\r').Replace(#10, '\n');
JsonBody := '{"messages":[{"role":"user","content":"' + SafeFrage +
'"}],"max_completion_tokens":4000}';
UTF8Bytes := UTF8Encode(JsonBody);
HTTPClient := TFPHttpClient.Create(nil);
ResponseStream := TStringStream.Create('');
RequestBodyStream := TMemoryStream.Create;
try
if Length(UTF8Bytes) > 0 then
RequestBodyStream.WriteBuffer(UTF8Bytes[1], Length(UTF8Bytes));
RequestBodyStream.Position := 0;
HTTPClient.AddHeader('api-key', Trim(Self.APIKey1));
HTTPClient.AddHeader('Content-Type', 'application/json; charset=utf-8');
HTTPClient.RequestBody := RequestBodyStream;
try
HTTPClient.Post(FullURL, ResponseStream);
if ResponseStream.DataString <> '' then
begin
with GetJSON(ResponseStream.DataString) do
try
if Assigned(FindPath('choices')) then
Result := FindPath('choices[0].message.content').AsString
else
Result := 'Azure-Err: ' + ResponseStream.DataString;
finally
Free;
end;
end;
except
on E: Exception do
Result := 'Err: ' + E.Message + ' \
Body: ' + ResponseStream.DataString;
end;
finally
RequestBodyStream.Free;
ResponseStream.Free;
HTTPClient.Free;
end;
end;
end;

```

Author

Diplom-Betriebswirt Christoph T. Wildensee, Ph.D. (Business Administration / Informatics), Master of Financial Technical Analysis, CISM, CRISC, CDPSE, is a well-known Auditor and Data / Process Analyst at enercity AG, Hannover, Germany. Christoph's special focus is on identifying errors and potential for optimization within IT systems relevant to financial accounting.