



Code-Analyse im SAP-Kundennamensraum und KI-gestützte Sicherheit: Risiken erkennen, Compliance stärken

Bereits im Jahr 2009 boten die Herren Wiegenstein/Schumacher/Schinzler/Weidemann mit dem Buch „**Sichere ABAP-Programmierung**“, erschienen bei SAP Press/Galileo Press, einen umfangreichen Leitfaden zum Qualifikationsaufbau, um generell unsichere SAP-Programmierung zu identifizieren. Dies betrifft sowohl die Berechtigungsseite als auch die tatsächliche Programmierung von Funktionalitäten, die im Unternehmen vielfältig benötigt werden.

Diese können aber auch so entwickelt und implementiert werden, dass hieraus operative Sicherheitsrisiken entstehen. Vorrangig ging es bisher um statische Code-Analyse (SCA), d. h. um das Finden bestimmter Begrifflichkeiten des Befehlsatzes und ggf. auch von Variablenanomalien, die allerdings im Gegensatz zu den feststehenden Befehlen multipel einsetzbar und umgestaltbar sind, sodass Fehlerquellen nicht so einfach zu entdecken sind. Schon das Hochladen von extern gehaltenem Quellcode, und sei es in einer einfachen Textdatei, um diesen nur mit temporärer Speicherung mit maßgeblichen Tabellenänderungen auszuführen und

Code-Analyse im SAP-Kundennamensraum und KI-gestützte Sicherheit: Risiken erkennen, Compliance stärken

Christoph Wildensee

Bereits im Jahr 2009 boten die Herren Wiegenstein / Schumacher / Schinzel / Weidemann mit dem Buch „**Sichere ABAP-Programmierung**“, erschienen bei SAP Press / Galileo Press, einen umfangreichen Leitfaden zum Qualifikationsaufbau, um generell unsichere SAP-Programmierung zu identifizieren. Dies betrifft sowohl die Berechtigungsseite als auch die tatsächliche Programmierung von Funktionalitäten, die im Unternehmen vielfältig benötigt werden. Diese können aber auch so entwickelt und implementiert werden, dass hieraus operative Sicherheitsrisiken entstehen. Vorrangig ging es bisher um statische Code-Analyse (SCA), d.h. um das Finden bestimmter Begrifflichkeiten des Befehlssatzes und ggf. auch von Variablenanomalien, die allerdings im Gegensatz zu den feststehenden Befehlen multipel einsetzbar und umgestaltbar sind, sodass Fehlerquellen nicht so einfach zu entdecken sind. Schon das Hochladen von extern gehaltenem Quellcode, und sei es in einer einfachen Textdatei, um diesen nur mit temporärer Speicherung mit maßgeblichen Tabellenänderungen auszuführen und danach ohne Protokollierung im Nirvana verschwinden zu lassen, zeigt bereits, dass solche Konstrukte einer besonderen Betrachtung unterworfen sein müssen. Der temporäre Ansatz ist dabei allerdings ein Problem, dem man mit zeitpunktbezogenen Auswertungen nicht sinnvoll begegnen kann. Nur der Einsatz professioneller Entwicklungstools, die sich in der Entwicklungsphase nicht ausschalten lassen, Hinweise für problematische Programmierungen geben und dies auch dokumentieren, ggf. sogar einen Workflow zur Freigabe solcher Code-Parts bereitstellen, können hier für Abhilfe sorgen. Ein solches Vorgehen wird als *Static Application Security Testing* (SAST) bezeichnet. Beispiele für entsprechende Anbieter sind SecurityBridge, sie bieten automatisierte statische und dynamische Scans direkt im SAP-System und in Eclipse ähnlich einer Rechtschreibprüfung für Code an, und Onapsis, auch hier wird ABAP-Code auf Sicherheitslücken, Fehlkonfigurationen und Compliance-Verstöße analysiert. Xiting sollte man zuletzt hinsichtlich der Integration von Berechtigungsfragen in der Entwicklung ebenfalls nicht unterschlagen. Die bisher herkömmliche **statische Code-Analyse**, oft durchgeführt über Transaktionen wie z.B. CODE_SCANNER (AFX_CODE_SCANNER) und ABAP Test Cockpit (ATC; als Basis hierfür der SAP Code Inspector SCI mit Prüfkatalogen), **genießt unter Entwicklern und auch Revisoren einen zweifelhaften Ruf. In der Praxis wird sie häufig eher gemieden als geschätzt**, was vor allem an der starren, regelbasierten Arbeitsweise traditioneller Werkzeuge liegt. Diese Scans produzieren regelmäßig eine überwältigende Anzahl an Ergebnissen, die sich jedoch oft als „False Positives“ entpuppen – also Fehlalarme, die zwar formal gegen eine Regel verstoßen, im spezifischen Kontext der Anwendung jedoch kein reales Risiko darstellen. Für die Beschäftigten bedeutet dies einen enormen manuellen Aufwand, um in einer Flut von Zufallsfunden die tatsächlich kritischen Schwachstellen zu identifizieren. Zudem sind die Einstiegshürden hoch: Die Bedienung und korrekte Konfiguration dieser Experten-Funktionen erfordert spezialisiertes Wissen, das in vielen Teams nicht flächendeckend vorhanden ist. Das Ergebnis ist eine zeitaufwendige Analyse, die Probleme aufzeigt, den Nutzern aber ohne konkrete Lösungsideen mit der Behebung allein lässt. Trotzdem ist es mit einem Report wie ZABAPDOWN01, der im Internet frei zugänglich ist, zumindest möglich, den gesamten Quellcode eines Systems über Transaktionen wie SA38, OODR u.a. herunterzuladen und als Textdatei zur Verfügung zu stellen. Steht dieser als Datei bereit, wobei die drei Datenfelder NAME (Programmname), ZEILE (Zeile innerhalb eines Programms) und CODE (Code-Zeile mit einem oder mehreren Befehlen, Variablenzuweisungen usw.) vorliegen, kann er in einer MS Access-Datenbank unter dem Systemnamen gespeichert werden.

KI als Sparringspartner in der Revision

In der komplexen Welt von SAP-Produktivsystemen stellt kundeneigener Quellcode (Y- und Z-Namensraum) für die Revision oft eine „Blackbox“ mit großem Umfang dar. Wo Standardprozesse durch SAP-eigene Kontrollen abgesichert sind, öffnen individuelle Anpassungen Tür und Tor für Prozessanomalien, Sicherheitslücken und Verstöße gegen regulatorische Anforderungen wie nach

GoBD und / oder DSGVO (GDPR). Ohne spezialisierte Tools zur dynamischen Code-Analyse im Entwicklungsprozess (DCA) gleicht die manuelle Prüfung mit statischer Code-Analyse (SCA) wie oben anklingend einer Sisyphusarbeit. Hier tritt Künstliche Intelligenz nicht als Ersatz, sondern als „kommunikationswilliger“, weitgehend „wissender“, hocheffizienter High-Level-Sparringspartner eines Revisors auf, der die KI beliebig einsetzen kann, sofern er Anforderungen und Bedingungen zielgerichtet und umfassend formuliert. Ein solcher Kooperationswille der Beschäftigten in Teams und mit KI-Unterstützung wird in der Zukunft immer wichtiger.

Die Rolle der KI: Vom Pattern-Matching zur semantischen Analyse

Bisherige manuelle Prüfungen beschränkten sich oft auf die Suche nach risikobehafteten Schlüsselwörtern (z.B. EXEC SQL, EDITOR-CALL, INSERT REPORT, RPY_PROGRAM*, cl_wb_source*, cl_oo_fact* usw.), daher auch als statische Analyse bezeichnet. Eine KI geht einen Schritt weiter: Sie „versteht“ den Kontext, sofern der Programm-Quellcode innerhalb der token-Grenzen umfänglich bereitgestellt wird.

- **Logik-Verständnis:** Während eine einfache Suche nur das Fehlen eines Befehls bemerkt, erkennt die KI, ob eine Berechtigungsprüfung an der *falschen Stelle* steht oder durch eine fehlerhafte IF-Logik oder Rückgabecode (sy-subrc) umgangen werden kann.
- **Identifikation von Datenabflüssen:** Die KI kann Datenströme innerhalb des Programms nachvollziehen. Sie erkennt, ob personenbezogene Daten (DSGVO) in Reports unmaskiert ausgegeben oder in unsichere lokale Verzeichnisse exportiert werden.
- **Compliance-Mapping:** Die KI kann Quellcode direkt gegen regulatorische Frameworks prüfen. Sie erkennt beispielsweise, ob Buchungslogiken die Unveränderbarkeit gemäß GoBD gefährden, indem sie direkte Tabellenzugriffe (MODIFY oder UPDATE auf SAP-Standard- / Systemtabellen) statt offizieller BAPIs identifiziert.

Der Revisor und die KI: Ein kooperatives Tandem

Die KI fungiert als „Vorstopper“. Der Revisor füttert die KI mit Code-Extrakten und erhält im Gegenzug eine qualifizierte Risikoeinschätzung. Und der Revisor kann beliebig oft nachfragen, die KI ist nie genervt und nur limitiert in der Art, wie der Revisor mit ihr kommuniziert.

- **Effizienzsteigerung:** Die KI sortiert „Rauschen“ aus. Der Revisor muss nicht 1.000 Zeilen Code lesen, sondern konzentriert sich auf die wenigen von der KI markierten Passagen, die tatsächlich ein Risiko für die Ordnungsmäßigkeit (z.B. nach § 239 HGB / § 146 AO, GoBD, ggf. zusätzlich auch nach §§ 202a, 202c, 263a, 303a, 303b StGB) darstellen.
- **Wissenstransfer:** Die KI erklärt, *warum* ein Coding problematisch ist. Dies dient dem Revisor als Argumentationshilfe gegenüber der Fachabteilung oder der Entwicklung, um Maßnahmen kooperativ abzuleiten.
- **Objektivität:** Die KI prüft jedes Programmfragment mit der gleichen Akribie, frei von Zeitdruck, Betriebsblindheit oder anderen limitierenden Faktoren.

Erwartbare Ergebnisse: Qualitätssicherung auf neuem Niveau

Der Einsatz von KI in der Code-Revision liefert messbare Mehrwerte:

- **Höhere Prüfdichte:** Es können deutlich mehr Systeme und von dort Programme in kürzerer Zeit geprüft werden, was die Gesamtsicherheit der Systemlandschaft erhöht (da Customizing / System- & Datenänderungen im Entwicklungssystem erfolgen und von dort nach oben

durchtransportiert werden, sind auch die Systemstufen unterhalb der Produktivebene rechnungslegungs- und nachweisrelevant, das DEV-System speist die höheren Systeme).

- **Reduzierung von „False Positives“:** Durch das Verständnis semantischer Zusammenhänge liefert die KI präzisere Treffer als rein regelbasierte Skripte.
- **Frühwarnsystem für Fachbereichs-IKS:** Die KI, sofern sie auch analog im zuständigen Fachbereich eingesetzt wird, kann Schwachstellen im internen Kontrollsystem (IKS) aufdecken, wenn sie erkennt, dass Reports systematisch Kontrollen umgehen.
- **Revisions sichere Dokumentation:** Die von der KI generierten Erläuterungen können in den Revisionsbericht einfließen, Feststellungen sind für Nicht-Entwickler transparent.

Die finale Entscheidung über die Kritikalität bleibt beim Revisor. Für die SAP-Revision ohne spezialisierte SCA- und / oder DCA-Tools ist die KI der entscheidende Hebel, um von einer stichprobenartigen Prüfung zu einer tiefgreifenden, risikoorientierten Analyse der gesamten Systemlandschaft zu gelangen. Sie ist der Sparringspartner, der die Nadel im Heuhaufen nicht nur findet, sondern auch erklärt, warum sie dort nicht liegen darf.

Mögliches Vorgehen einer unterstützten Analyse

Das nachfolgende Programmdesign zeigt den Aufbau einer kategorisierten Code-Analyse nach bestimmten Statements im Quellcode als SCA. Zu erkennen sind hier zunächst links die einzelnen Queries mit ihren Bedeutungen, die ausgeführt werden können. Rechts davon stehen die Systeme, die zur Prüfung anstehen, hier sind die Codezeilen enthalten, auf die die Analysen wirken. Es handelt sich also um jeweils eigene Quellcode-Tabellen mit Millionen Code-Zeilen der verschiedenen Systeme, die jeweils analysiert werden. Dies können alle Systeme mit Anpassungen sein, die hier nacheinander abgearbeitet werden. Im unteren Bereich stehen die Ergebnisse als Grid / Tabelle, wenn man mit Bezug zu einem System einen Query gestartet hat.

The screenshot shows the 'Check ABAP (static)' application. The 'Queries' section on the left lists various statements and their meanings. The 'Data Tables' section in the middle shows a list of tables, with 'PL__abap2' selected. The 'Results' section at the bottom displays a table with columns ID, NAME, Row-Nr, and CODE.

Statement	Meaning
Editor-Call	Das Öffnen des internen ABAP-Editors! gem a
FIND REGEX	FIND REGEX, Suche nach Mustern wie DATA
FOR INPUT IN TEXT M(FOR INPUT IN TEXT MODE als Direkteingab	
FROM DATABASE	IMPORT I FROM DATABASE als Laden von I
FROM REPOSRC	SELECT * FROM REPOSRC als direkter Date
Generate Report	Das Generieren einen Reports aus bestehend
Generate Subroutinepool	Das Generieren von Pool-Funktionen (dynamic
Insert Report	Das Öffnen des internen ABAP-Editors! gem au

ID	NAME	Row-Nr	CODE
2229251	ZI...TABINH_DOWNL	169	editor-call for gt_lines title gv_title
2513310	ZS..._I01	54	EDITOR-CALL FOR REPORT gs_node-progname DISPLAY-MODE .
2513537	ZS..._I01	281	I* EDITOR-CALL FOR content[] TITLE u-title.;
2513683	ZS..._I01_V1	75	EDITOR-CALL FOR REPORT gs_node-progname DISPLAY-MODE .
2513899	ZS..._I01_V1	291	I* EDITOR-CALL FOR content[] TITLE u-title.;

Abb. 1: Trefferliste einer Analyse.

Nimmt man den markierten Eintrag der Trefferliste, hier ZSC... (Row-Nr. 281), so kann man sich über den Button Details den Quellcode als Ausschnitt ansehen, und zwar +- 10 Zeilen ober- und unterhalb des gesuchten Schlüsselwortes, hier **EDITOR-CALL**. Unter Code fragment +- kann man auch Werte eingeben zwischen 10 und 50, sodass mindestens 20 und höchstens 100 Zeilen des Quellcodes eingesehen werden können. Bereits hier kann man abschätzen, ob es sich um validen problematischen Code handelt, oder dieser gar nicht genutzt wird, weil er z.B. funktional „kaltgestellt“ wurde.

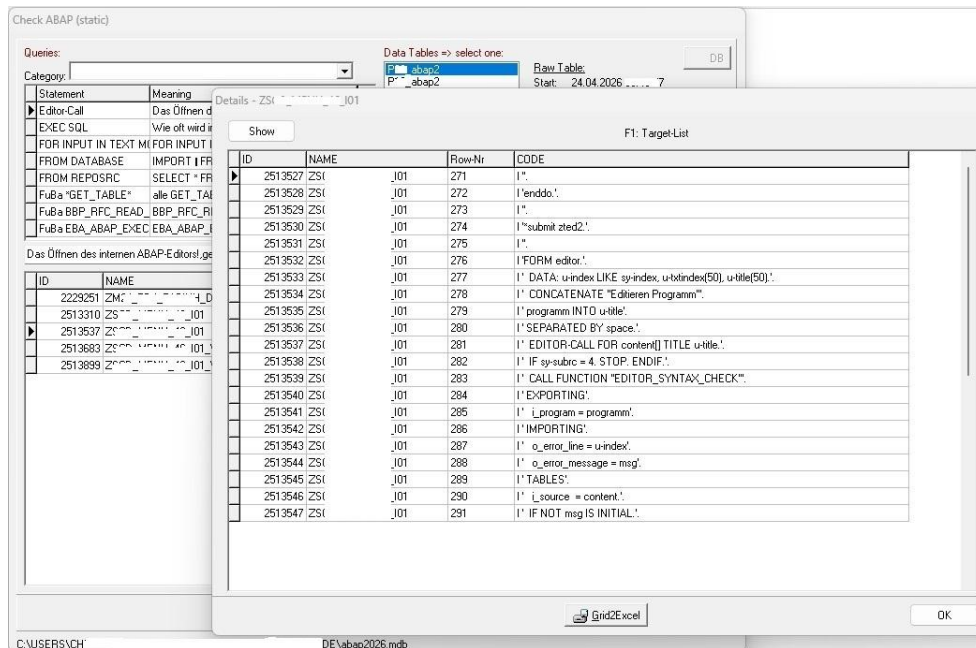


Abb. 2: Details zum gewählten Programm, Ausschnitt aus dem Code.

Erkennt man nun, dass es sich um kritischen Code handelt, kann man über F1 die Target-Liste ansehen und das Programm mit Systemzuweisung und dem Hinweis auf den Kritikalitätsauslöser der Liste hinzufügen.

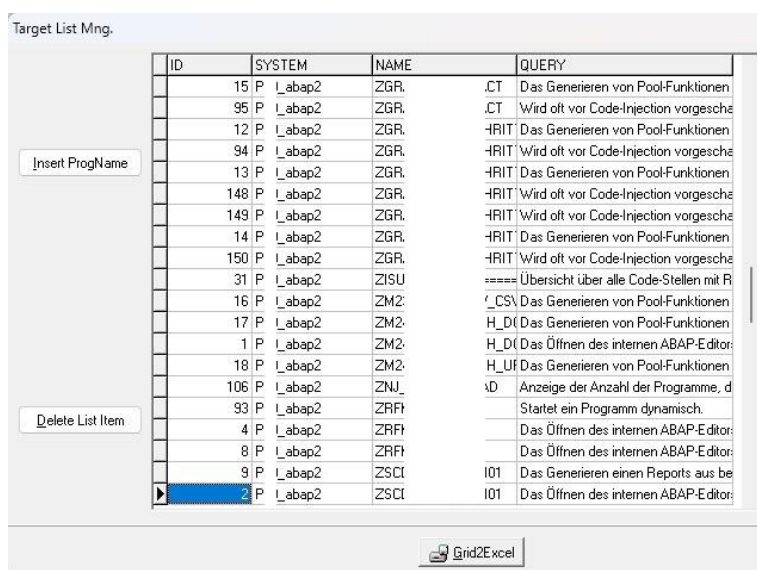


Abb. 3: Target-Liste.

Diese Target-Liste kann nach Vervollständigung an Excel übergeben und dort am Ende weiterverarbeitet werden, d.h. hier wird nach einer nochmaligen Ergebnisüberprüfung gekennzeichnet, ob es sich um ein notwendiges Programm handelt, oder dieses zu löschen oder im Zugriff stark einzuschränken ist.

Zusätzlich können auch weitere Auswertungen ausgeführt werden, die zwei Kriterien mit AND oder OR in derselben Zeile kombinieren.

ABFRAGE	FIELD1	SQLTEXT	FILTER1	AND/OR	FIELD2	FILTER2	SC	INAKTIV
OO Editor Call / Insert Report => *CL_ABAP_SOURCE* OR *CL_OO_FACTORY*	CODE LIKE	SELECT * FROM #	*CL_ABAP_SOURCE*	OR	CODE LIKE	*CL_OO_FACTORY*	Y	☐
OO Editor Call / Insert Report => *CL_ABAP_COMPILER* OR *RS_INSERT_REPORT*	CODE LIKE	SELECT * FROM #	*CL_ABAP_COMPILER*	OR	CODE LIKE	*RS_INSERT_REPORT*	Y	☐
OO Editor Call / Insert Report => *CL_GUI_TEXTEDIT* OR *CL_WB_SOURCE*	CODE LIKE	SELECT * FROM #	*CL_GUI_TEXTEDIT*	OR	CODE LIKE	*CL_WB_SOURCE*	Y	☐
OO Editor Call / Insert Report => *EDITOR_PROGRAM* OR *abapxt*	CODE LIKE	SELECT * FROM #	*EDITOR_PROGRAM*	OR	CODE LIKE	*abapxt*	Y	☐
OO Editor Call / Insert Report => *go_editor* OR *set_text_as_stream*	CODE LIKE	SELECT * FROM #	*go_editor*	OR	CODE LIKE	*set_text_as_stream*	Y	☐
OO Editor Call / Insert Report => *_source(* OR *edit_string(*	CODE LIKE	SELECT * FROM #	*_source(*	OR	CODE LIKE	*edit_string(*	Y	☐
Insert Report ohne bisherige Standard-Befehle I (*RS_PROGRAM* oder *RS_CORR_INSERT*)	CODE LIKE	SELECT * FROM #	*RS_PROGRAM*	OR	CODE LIKE	*RS_CORR_INSERT*	Y	☐
Insert Report ohne bisherige Standard-Befehle II (*set_trdir* oder *set_source*)	CODE LIKE	SELECT * FROM #	*set_trdir*	OR	CODE LIKE	*set_source*	Y	☐

Abb. 4: Bsp. Felder-Konstellation für AND / OR.

Sofern in der Liste der vorigen Standardabfragen als auch diesen kombinierten Fragen keine gewünschte Fragestellung an die Datenbank gestellt werden kann, kann auch mit Angabe eines Teilstrings eine Suche gestartet werden, zuerst unter F5 und dort unter F1 (Search [Part-String]). Resultate werden angezeigt, die ebenso zur Target-Liste hinzugefügt werden können.

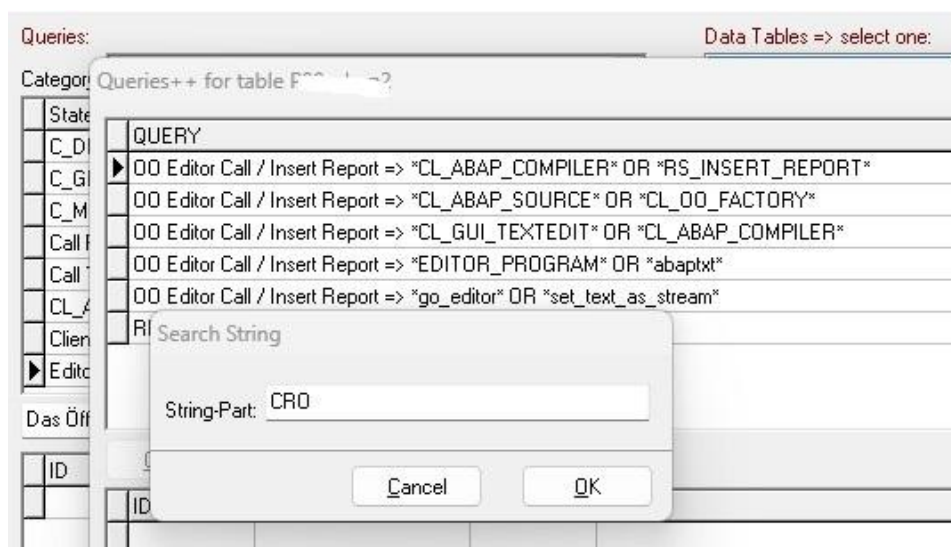


Abb. 5: Beliebige Part-String-Suche.

Beliebig viele Datenbanken

Abbildung 2 zeigt unten links einen Pfad zu einer Datenbank, in der die Quellcodezeilen enthalten sind. Um nicht Entwicklungs-, Test- / Integrations- und Produktionsdatenbanken miteinander zu vermischen, können mit Ziel einer solchen Trennung beliebig viele Datenbanken angelegt und gleichzeitig ausgewertet werden. Die Target-Listen sind dann ebenfalls voneinander getrennt. Vorteilhaft ist dies dahingehend, dass das Datenbanklimit jeder Access-Datenbank hierdurch aufgehoben ist (mdb, nicht accdb). Ausgewählt wird über den DB-Button oben rechts.

Doch hier endet das Programm nicht.

KI-Integration

Ist man sich nicht sicher, ob das Code-Fragment tatsächlich problematischen Code beinhaltet oder dieser wie befürchtet wirkt, wählt man aus der ersten Darstellung des erkannten Problemcodes heraus F10 (Xtr), es öffnet sich der gesamte Quellcode des ausgewählten Programms. Sofern in der Vergangenheit bereits zu diesem Programm eine Analyse erstellt wurde, kann diese über den Button Load hereingeholt werden, ist dies aber nicht der Fall oder es wird eine neue Analyse angestrebt, kann über den Button Analyze eine KI-basierte Analyse durchgeführt werden.

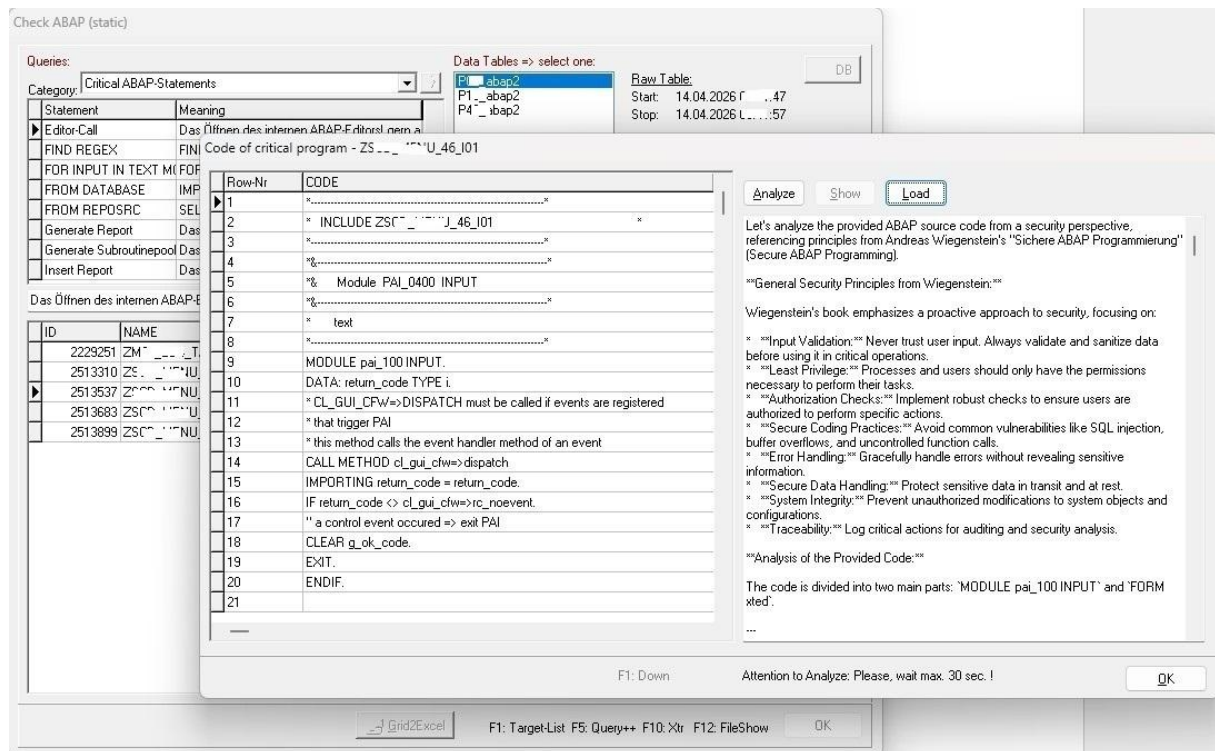


Abb. 6: Gesamter Quellcode des gewählten Programms, folgend KI-Bewertung.

Nach Anstarten der KI-Anfrage über den Button Analyze wird ein Connector gestartet, der per API eine Verbindung zur KI aufbaut und die Frage mit den Quellcode-Daten des gewählten Programms an die KI sendet. Die KI bearbeitet die Anfrage und sendet eine Antwort zurück an den Connector. Dieser stellt die KI-generierte Antwort dem aufrufenden Programm zur Verfügung und schließt sich. Das aufrufende Programm hat darauf gewartet, dass sich der Connector schließt und übernimmt mit dem Schließen des Connectors die Antwort und stellt sie danach im Fenster bereit.

Die erste Frage, die gestellt wird, lautet: „Analyze the source code generally from security aspects, especially according to those of Andreas Wiegstein, who wrote the book +Sichere ABAP Programmierung+. Represent the problematic parts with code snippets. =>“, hinzugefügt wird zur Frage der gewählte Quellcode. Die Frage kann selbstverständlich erweitert / geändert werden.

Über F1 (Down) kann der Fensterinhalt unter c:\temp mit <System>-<Programmname>.txt abgespeichert werden, die KI-generierten Ergebnisse werden abgelegt und später untersucht. Hilfreich ist, dass die KI-Resultate iterativ Hinweise auf weitere Befehle oder Konstrukte beinhalten, die Kritik unterworfen sind, die folgend die Liste der statischen Abfragen erweitern.

Beliebig viele Fragen an die KI

Sinnvoll ist dabei aber auch, ob mit dem KI-System weitere Kommunikation zum Fall ablaufen kann. Der Prüfer kann beliebig viele Individualfragen stellen und die Kommunikation mit der KI wird automatisch hergestellt.

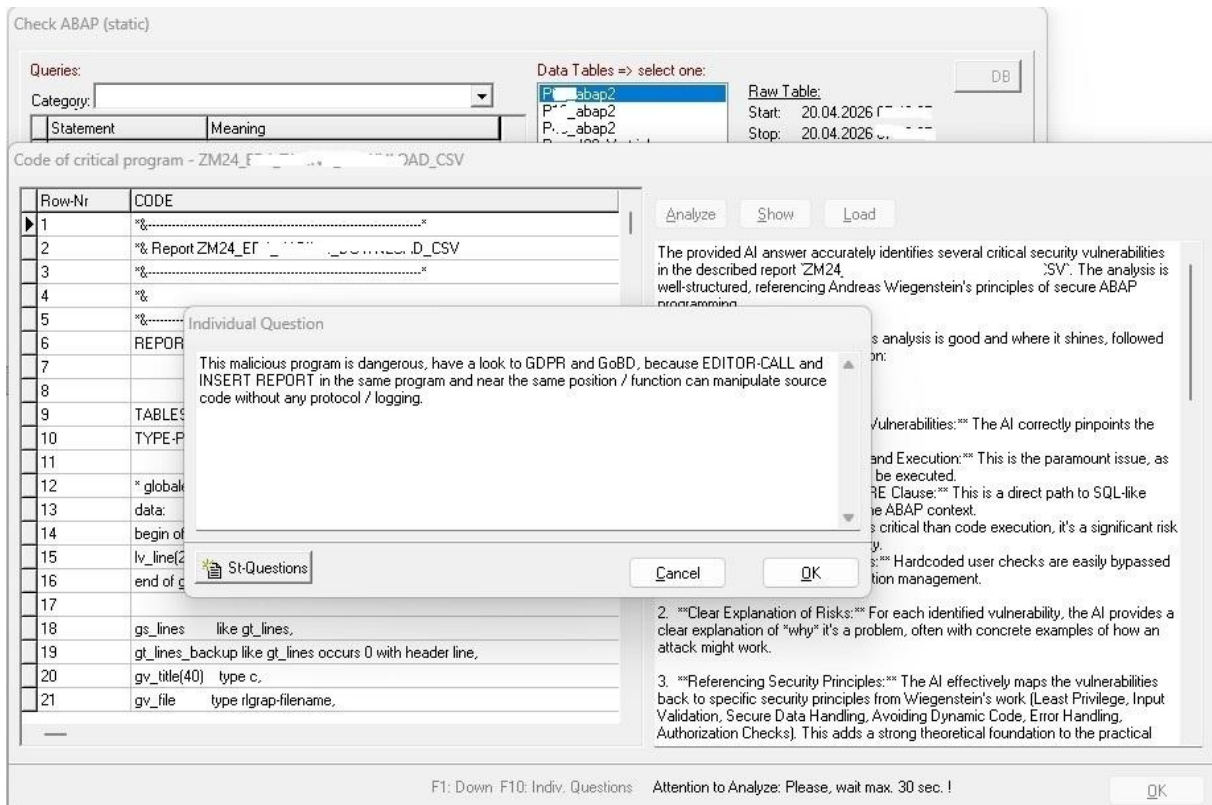


Abb. 7: Nach der ersten Standardfrage Möglichkeit individueller Kommunikation mit der KI.

Mit OK wird die Frage an die KI gesendet, ist diese Frage das erste Mal aufgetreten, kann sie auch als Standardfrage abgespeichert und bei jeder weiteren programmbezogenen Analyse wieder herausgeholt werden, das spart Tipparbeit.

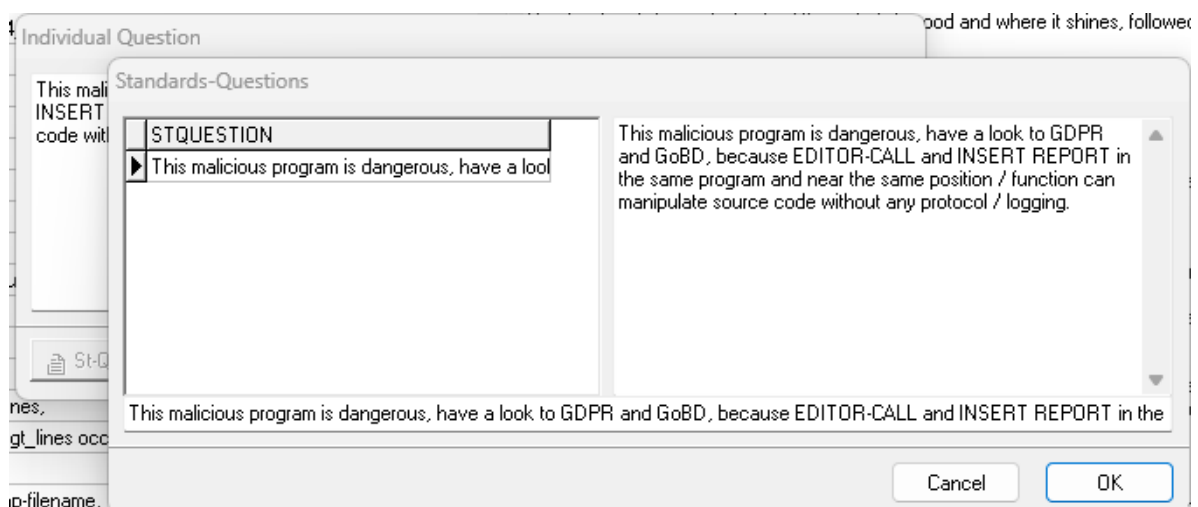


Abb. 8: Standardfragen, die immer wieder vorkommen, können hereingeholt und ausgeführt werden.

Die Ergebnisse zu jeder einzelnen Analyse je Programm werden als einfache Textdateien abgelegt, können aber auch so oft mit der KI generiert werden, wie dies gewünscht ist, alte Versionen werden nicht überschrieben, sondern historisiert, um Ansätze zur Erweiterung des Analyse-Portfolios nicht zu verlieren und alle Aspekte der bisherigen Analysen einbeziehen zu können. Bei Sicherung der Dateien in einem spezifischen Verzeichnis kann dies ausgewählt und als Standardleseverzeichnis markiert werden.

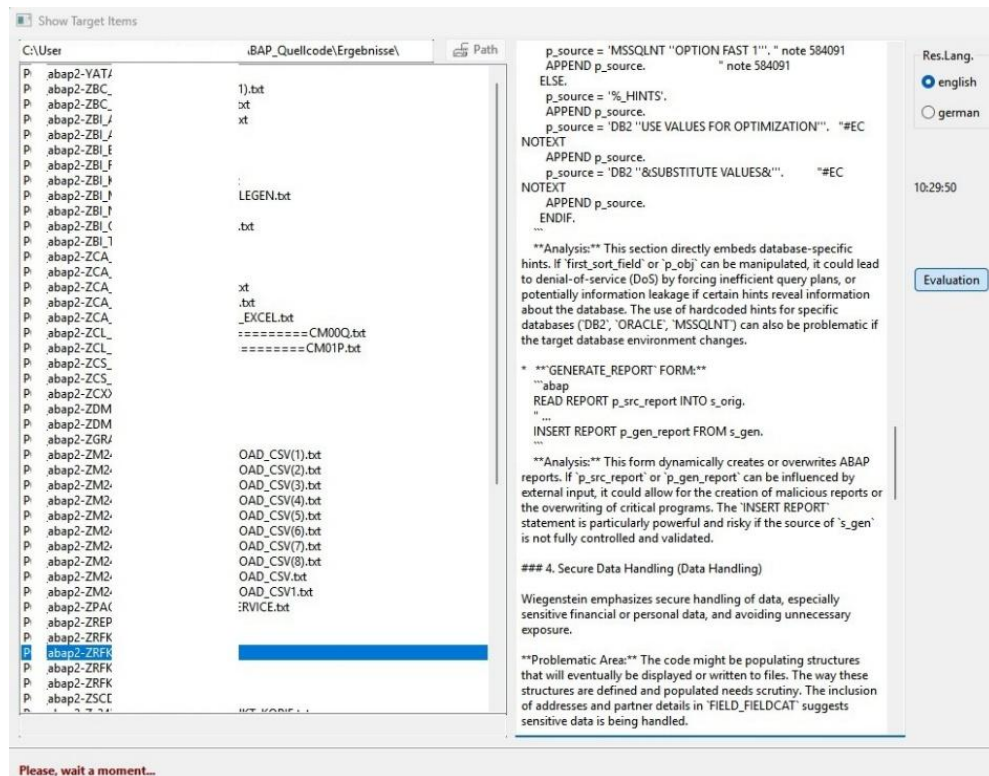


Abb. 9: Einsicht in die historisierten KI-Abfrageergebnisse (Ausschnitt).

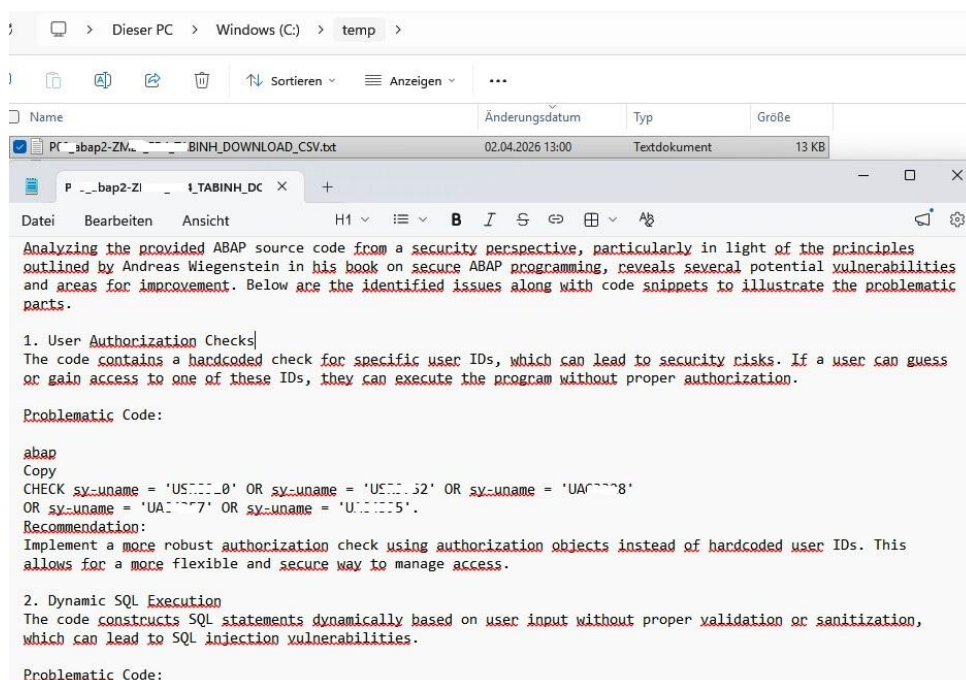


Abb. 10: Bereitgestellte Textdatei aus der KI im Datei-Explorer zum ausgewählten Programm und System.

Datenschutz

Datenschutz ist in diesem Rahmen wesentlich. Eine Bereitstellung von Quellcode, der durch den Anfragenden aufgrund des Umfangs nicht vollumfänglich vorqualifiziert werden kann – weder fachlich-inhaltlich noch aus Sicht des Datenschutzes – an eine KI ist mit hohem Risiko des Datenabflusses personenbezogener Daten verbunden, denn innerhalb des Quellcodes finden sich ggf. an mehreren Stellen Verweise auf Usernamen, UserIDs, Email-Adressen usw. Zumindest kann bei einer Bunch-Bereitstellung kaum davon ausgegangen werden, dass ein Revisor zunächst die DSGVO-Konformität für jeden einzelnen Fall bzw. jede einzelne Zeile prüfen kann und die Gewähr dafür übernimmt. Daher ist es zwingend, dass eine DSGVO-konforme Umsetzung der bilateralen Datenbereitstellung erfolgt. Ist diese aber im Zugriff, auch wenn dieser ggf. in der Geschwindigkeit im Verhältnis zu den frei zugänglichen Anbietern aus den USA oder China leicht gedrosselt ist, steht einer DSGVO-konformen Dauernutzung nichts im Wege.

KI	Kommunikationsattribute der spezifischen Connectoren	DSGVO-konform
Groq	Adresse=https://api.groq.com/openai/v1/chat/completions Modell=llama-3.3-70b-versatile APIKey=gsk_geWfQKH...	Nein
Gemini	Adresse=https://generativelanguage.googleapis.com/v1/models Modell=gemini-2.5-flash-lite APIKey=AlzaSyBEeDv....	Nein
DeepSeek	Adresse=https://api.deepseek.com/chat/completions Modell=deepseek-reasoner APIKey=sk-5126cb...	Nein
Mistral	Adresse=https://api.mistral.ai/v1/chat/completions Modell=mistral-large-latest APIKey=ZRml1Z...	Ja
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	Ja

Eine Protokollierung der KI-Aufrufe ist ebenfalls gegeben. Diese zeigt die aufgerufene Kommunikationsseite der KI, das genutzte Modell und den API-Key. Außerdem wird festgehalten, ob die Antwort übermittelt wurde. Im Fehlerfall wird ein Fehlercode ausgegeben.

```
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.
22.04.2026  06:43:10 - Site: https://11.11.11.11/cognitive-0000000.cognitiveservices.azure.com/
22.04.2026  06:43:10 - Modell: gpt-5.1-0000000
22.04.2026  06:43:10 - APIKey: 6elMS7zKL.....AAACOG0Tcm
22.04.2026  06:43:10 - APIVers: 2024-12-01-preview
22.04.2026  06:43:40 - Answer stored.
```

Abb. 11: Log-Schreibung erfolgreich, Ergebnis mit anonymisierten Testdaten bei DeepSeek und Azure.

Auch bei der Zusammenfassung der Ergebnisse jedes einzelnen, als kritisch erkannten Programms zu einem Gesamtergebnis kann der KI-Connector helfen. Bereitgestellt werden alle abgespeicherten Dateien zu den jeweiligen Programmen mit den KI-generierten, umfänglichen Texten in Englisch. Um Negativeffekte wie z.B. Halluzinationen auszuschließen, **sollten zu jedem dieser Programme mehrere KI-Bewertungen vorliegen. Jede dieser Dateien wird mit Referenz auf den Programmnamen durch die KI zusammengefasst** und diese kurzen Beurteilungen werden in der Tabelle rechts gezeigt.

[illegible]

Abb. 12: Zusammenfassung der Teilergebnisse zu einer Kurzberichtstabelle.

Die Zusammenfassung soll wie die Frageumgebung als klassifizierte Listaufbereitung auf Englisch erfolgen, kann aber auch auf Deutsch herausgegeben werden.

CPU	_ABAP_Quelle\Ergebnisse\	Path
P	i2-ZRF	d
P	i2-ZRF	d
P	i2-ZRF	d
P	i2-ZSC	bt
P	i2-Z_3	/PRODUKT_KOPIE.txt
P	i2-Z_C	
P	i2-Z_R	COPY.txt
P	i2-ZBC	PIO_EMAIL.txt
P	i2-ZBC	FORMS.txt
P	i2-ZBI	
P	i2-ZBI	
P	i2-ZBI	i_KANN.txt
P	i2-ZCI	=====
P	i2-ZCI	4002.txt
P	i2-ZCI	vTRACT= MIOIS.txt
P	i2-ZCI	JTH=====
P	i2-ZCI	VE_ISSUE!
P	i2-ZCI	OR=====
P	i2-ZCI	OR=====
P	i2-ZCI	=====
P	i2-ZCI	MIP_CHAI
P	i2-ZCI	=====
P	i2-ZC	.txt
P	i2-ZC	=====
P	i2-YBF	=====
P	i2-ZOS	=====
P	i2-ZB	i\$ADRLSM02.txt
Ra	ab-ZC	t
Ra	ab-ZC	t
Ra	ab-ZFI	:
Ra	ab-ZH	: NLOAD_CSV.txt
Ra	ab-ZV	: nload_csv
Ra	ab-ZL	xt
Ra	ab-ZL	xt
Ra	ZDTP	
Ra	ZMBJ	

Below is a security-focused review of the program, with emphasis on aspects that Andreas Wiegstein discusses in "Sichere ABAP-Programmierung": dynamic code, authorizations, direct DB access, missing checks, etc. I'll quote the relevant code snippets and explain why they are problematic and how they could be hardened.

```

---
## 1. Dynamic code creation and execution (self-modifying code)
### Problem

The program defines arbitrary ABAP source in an internal table, writes it back to the repository and generates it:

""abap
parameters programm like rs38m-programm obligatory default 'Z'.

data: content(255) occurs 0 with header line, msg(400), anover.

start-of-selection.

    read report programm into content.
    perform editor.
    clear msg.
    insert report programm from content.

...

* Report generieren
generate report programm line msg+10(10) message msg+50.
  
```

Security issues:

- "Dynamic source manipulation":
 - read report ... and insert report ... allow the user to read and overwrite ABAP code of the report named in 'programm'. Together with 'generate report', this becomes ""self-modifying code"" which is extremely critical. It allows:
 - Introduction of backdoors,
 - Bypassing of transport control,
 - Manipulation of system behavior outside of standard ABAP Workbench controls.

Abb. 13: Mit der KI generierte Gesamtliste.

Die finale Ergebnisbereitstellung mit allen Programmen als Gesamtzusammenstellung erfolgt in Tabellenform. Die kurzen Zusammenfassungen können beliebig oft erstellt werden und der **Prüfer kann die für ihn besten Ergebnisformulierungen** für die Berichtserstellung **nutzen**.

Ein Ergebnis kann wie folgt aussehen:

File	KI-Antwort
PC_1_abap2-ZGR...	##### Der ABAP-Report 'ZGR' weist ein hohes Sicherheitsrisiko durch umfangreiche dynamische Codegenerierung ('GENERATE SUBROUTINE POOL') auf; unzureichend validierte Eingaben (z.B. Tabellen-/Feldnamen) können zu Code-Injection führen, da nur rudimentäre Prüfungen ('FORM c_it', einfache String-Ersetzungen) vorhanden sind. - Der Zugriff auf zahlreiche, teils sehr kritische Tabellen (u.a. EANL, ERCH, DFKK, USR02, CDHDR/CDPOS, TADIR, DBTABLOG etc.) erfolgt ohne erkennbare, systematische Berechtigungsprüfungen; die Steuerung über die Tabellen 'ZGRA' ... 'TRICT' und 'ZGR/ ...' ist sicherheitskritisch und muss streng gepflegt und selbst geschützt werden. - Die Autorisierung wird lediglich indirekt über Tabellenexistenz-Checks ('DD02L') und Restriktions-/Allow-Tabellen beeinflusst, nicht jedoch konsequent über SAP-Standardmechanismen wie 'AUTHORITY-CHECK OBJECT', was ungewollte Datenzugriffe ermöglicht und dem Prinzip 'Nur notwendige Rechte' widerspricht. - Die Validierung der Benutzereingaben ('FORM v_e') ist nur teilweise vorhanden: Pfad- und Dateiprüfungen sind implementiert, aber nicht umfassend; die einfache Erkennung von 'insert/delete/update' o.Ä. in Strings ist kein hinreichender Schutz gegen Injection-Angriffe, insbesondere im Zusammenspiel mit dynamischem Code. - Die Handhabung des Verschlüsselungsschlüssels ('p_enckey') ist sicherheitsrelevant: Er wird als Parameter übergeben und in dynamisch generiertem Code verwendet, ohne abgesichertes Schlüsselmanagement (z.B. Secure Storage) und robuste, korrekt implementierte Verschlüsselungslogik besteht ein erhöhtes Risiko für Datenexposition. - Der Report kann potenziell sensible System- und Geschäftsdaten über Fehlermeldungen, Logausgaben und Exportinhalte preisgeben; es fehlen klar erkennbare Mechanismen, um Informationen in Meldungen und Logs systematisch zu minimieren und zugleich intern revisionssicher und ausreichend granular zu protokollieren. - Die Logging-Funktionalität ('it_log', 'FORM p_l') dokumentiert zwar Parameter und Verarbeitungsschritte, jedoch ist unklar, ob Umfang, Detailtiefe und Aufbewahrungsdauer für Sicherheits- und Forensikzwecke ausreichen; zudem könnte die Weitergabe von Systemfehlern (z.B. über 'filemsg') Angriffsfläche bieten. - Die Webservice-Prüfung ('FORM c_ws') ist lediglich als Platzhalter vorhanden und führt keine tatsächliche Validierung (z.B. Erreichbarkeit, Zertifikate, TLS-Konfiguration) des E*-Servers durch, was für den Online-Modus eine zusätzliche Sicherheitslücke darstellt.++++
PC_1_abap2-ZM2...	##### Die wichtigsten Sicherheitsprobleme betreffen **dynamisches Open SQL mit frei eingegebenen WHERE-Klauseln und Tabellennamen**, die direkt aus Benutzereingaben (Select-Options und EDITOR.CALL) in den generierten SQL-Text übernommen werden; dies entspricht einem klassischen SQL-Injection-Szenario und erlaubt das Umgehen von Filtern sowie potenziell extrem teure Abfragen. - Die Lösung nutzt **dynamische Programmgenerierung ('GENERATE SUBROUTINE POOL') mit Benutzerinput** für Tabellennamen und WHERE-Klauseln; damit entsteht ein hochriskantes Muster von dynamischer Code-Erzeugung, das aus Sicht 'Sichere ABAP-Programmierung' heute als grundsätzlich zu vermeiden gilt. - Für den Zugriff auf beliebige Tabellen werden **keine rollenbasierten Berechtigungsprüfungen** durchgeführt; stattdessen gibt es nur eine Liste hart codierter Benutzer ('CHECK syname = ...'), wodurch das Programm für berechtigte oder kompromittierte Nutzer faktisch als allgemeines Daten-Exfiltrationstool fungiert. - Über ein frei wählbares Selektionsfeld für den Tabellennamen werden **beliebige DDIC-Tabellen per 'SELECT *' ausgelesen und exportiert**, inklusive potenziell sehr sensibler HR-, FI- oder Sicherheitstabellen, was ein gravierendes Datenschutz- und Compliance-Risiko darstellt. - Die Anwendung ermöglicht **Schreibzugriffe auf das Applikationsserver-Dateisystem** über einen frei konfigurierbaren Pfad (ohne Pfadvalidierung oder Verzeichnis-Whitelisting) sowie Downloads auf den Frontend-Client, jeweils ohne zusätzliche logische Einschränkungen oder Protokollierung. - Insgesamt fehlen **Logging und Nachvollziehbarkeit** für diese massenhaften Datenexporte (kein Protokoll von User, Tabelle, WHERE-Klausel, Pfad, Datensatzanzahl), zudem bestehen Robustheitsrisiken wie mögliche DoS-Szenarien durch 'SELECT *' auf sehr großen Tabellen ohne WHERE-Bedingung. PARTIALLY POOR AND CRITICAL =>++++
PC_1_abap2-ZRF...	##### Die sicherheitskritischste Schwachstelle liegt in der fehlenden Validierung und Bereinigung externer Eingaben (z.B. 'paramsel_type1', 'p_name', 'p_num', 'p_nm_opc', 'p_tp_opc', Felder aus 'ifieldinfo'/'ifranges'), die direkt für dynamische Code-Generierung verwendet werden und so prinzipiell Code-Injection bzw. fehlerhafte, manipulierte Reports ermöglichen. - Die umfangreiche dynamische Code-Generierung (z.B. in 'GENERATE_DATA_BLOCK', 'GENERATE_FORM_BLOCK', 'GENERATE_REPORT' mit 'INSERT REPORT') ist ein generelles Hochrisiko: Feldnamen, Strukturnamen und Order-By-Felder werden ohne Whitelisting in ABAP-Quellcode eingefügt; bei manipulierbaren Steuerdaten kann dies zu gefährlichen oder unvorhersehbaren Programmen führen. - Es sind keine expliziten Berechtigungsprüfungen ('AUTHORITY-CHECK') im Code erkennbar; bei Zugriff auf Finanz- und Geschäftspartnerdaten kann dies zur unzulässigen Einsicht sensibler Informationen führen, falls die Aufruferseite dies nicht zuverlässig abdeckt. - Die Behandlung sensibler Daten (z.B. Geschäftspartner-Name, Adresse, 'CLERK_ID', diverse Namens- und Adressfelder im 'FIELD_FIELDCAT') erhöht das Expositionsrisiko, da diese Daten ohne

Abb. 14: Mögliches Ergebnis (Auszug).

Weitere Analysen

Die Ergebnisse der Target-Liste(n) und die Kurzergebnisdarstellung beinhalten final die System- und Programmnamen der als zunächst kritisch erkannten Programme und Funktionsbausteine im Kundennamensraum. Die Programmnamen können mit den SAP-Tabellen TADIR (Katalog der Repository-Objekte) und TRDIR (Systemtabelle) und mit der Transaktion SE38 (Code und Eigenschaften) betrachtet werden. Die Felder der Tabelle TADIR sind OBJECT und OBJ_NAME sowie DEVCLASS (dieses referenziert auf das Berechtigungsobjekt S_PROGRAM mit den Feldern P_GROUP und P_ACTION in den Berechtigungsstammsätzen in Kombination mit S_TCODE und Aufruftransaktionen wie SA38, OODR u.a.). Die Felder der Tabelle TRDIR sind CNAM, CDAT, UNAM, UDAT, IDATE und SUBC (Typ). Für die Funktionsbausteine gilt die Transaktion SE37 und die Tabelle TFDIR (Funktionsbausteine) mit den Feldern FUNCNAME, PNAME, PNAME_MAIN und STEXT. Eine weitere Sicht ist das Durchführen von Transporten von Quellcode durch die Systeme (Überprüfung mit Transaktion SUIM mit den Berechtigungsobjekten S_TCODE, S_PROGRAM, S_TRANSPRT und der Transaktion STMS mit den Tabellen TPALOG, E071 und USER_ADDR).

DEVCLASS ist insoweit für die Prüfung relevant, dass wenn der Zugriff für den Programmaufruf der Y- / Z-Programme so weit gestreut ist und wenig Klassifizierung und Begrenzung bewirkt, dass beinahe

keine Steuerungswirkung eintritt, eine zu große Anzahl an Personen Programme des Kundennamensraumes aufrufen dürfen. Dies sollte unterbunden werden. Auch weisen viele Programme eine Aufrufsteuerung abhängig von der aufrufenden UserID auf, sy-username. Ist dies der gewählte und häufige Weg, um Programmaufrufe zuzulassen oder zu unterbinden, bedeutet dies, dass diese auch gelten, wenn die Personen den Bereich verlassen und andere Aufgaben übernehmen oder sogar das Unternehmen verlassen, wenn dann neue Kollegen die Aufgaben übernehmen sollen und kein Zugriff mehr besteht. Dies ist eines der häufigsten Problemfelder in diesem Kontext. Und zuletzt können auch Programme im System gesucht werden, die für beliebige Tabellen Änderungen in Feldern ohne Nachvollzug durchführen können. Sind solche vorhanden, kann dies die Systemstabilität gefährden. Der Anpassungsbedarf ist erheblich.

Fazit

Die Integration von KI bei Datengenerierungen und -anreicherungen in Revisionsprüfungen ist nicht nur möglich, sondern angeraten. Die KI-generierten Texte können separat strukturiert und ausgewertet werden, um bisherige Ergebnisse zu bestätigen, die Erkenntniswelt zu verbessern und die Basis der Abfragen zu erweitern. Außerdem kann die gesamte Systemlandschaft unkompliziert und ohne wesentlichen Mehraufwand in die Prüfung integriert werden. Quellcode-Audits sind aufgrund der Kreativität von Entwicklern, aber besonders des immensen externen Berater- und Implementierer-Einsatzes im eigenen Haus und regelmäßig notwendigen Prozesserweiterungen in SAP immanent wichtig, somit sollten die Bemühungen, „malignen“ Code zu identifizieren und zu eliminieren, spätestens alle drei Jahre wiederholt werden. Es handelt sich also um ein *dankbares* Thema hinsichtlich des Aufbaus eines „**Continuous Auditing**“. Dies gilt im Übrigen für jede Form von Code-Review. Professionelle Tools sind teuer und nicht immer im Haus gewünscht. Das eigene Unternehmen offeriert KI-Modelle mit einer DSGVO-konformen Nutzung und erheblich höheren Nutzungslimits. Diese sollte die Revision dauerhaft nutzen und eigene Kapazitäten stärken sowie Skills ausbauen, um eigene Lösungen zu entwickeln. Und dieses Programm ist beliebig erweiterbar. Letztlich besteht das übergeordnete Ziel darin, hinreichend zu verifizieren, dass die regulatorischen Anforderungen, wie sie in den Entwicklerrichtlinien festgelegt sind, eingehalten werden (z.B. getrennte Rollenwahrnehmungen ANFORDERUNG / MODULBETREUUNG / DEVELOP / TEST / FACHFREIGABE und TRANSPORT, Befehlsausschlussliste, RFC-Rechte, usw.) und dass der Umgang mit hochprivilegierten Zugriffsrechten in allen Systemstufen angemessen erfolgt. Dadurch wird zudem der unausgesprochene Vorwurf entkräftet, dass Entwickler ihre Sonderrechte ggf. missbräuchlich nutzen (können).

[Aus stilistischen Gründen bzw. zur Vereinfachung der Lektüre wird gelegentlich die männliche Schreibweise verwendet. Die weibliche Form ist dann jeweils ebenso gemeint und zu bedenken.]

Literatur

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): Projekt SCA; <http://www.wildensee.de/sca.zip> .

Anlage

Technische Elemente (Lazarus; uses ... fphttpclient, opensslsockets, fpjson, jsonparser)

Der Connector wird im Verwaltungsprogramm wie folgt aufgerufen:

```
[...]
if (anzds1 > 0) then // Anzahl von Datensätzen des Quellcodes des ausgewählten Programms
begin
// hier Aufruf des Connectors, wenn er im Pfad des Hauptprogramms vorhanden ist
...
if (FileExists(Uppercase(PChar(ExtractFilePath(ParamStr(0))+ 'ABAPSTA1.EXE')))) then
begin
...
ShellExecAndWait(ExtractFilePath(ParamStr(0))+ 'ABAPSTA1.EXE ', '', SW_SHOWMINIMIZED);
end;
...
end;
[...]
```

// ShellExecAndWait bewirkt, dass das Hauptprogramm erst weiterarbeitet,
// wenn der Connector nach der Kommunikation wieder beendet ist.

Connector Azure:

```
function TForm1.SendToAI(const Frage: string): string;
var
  HTTPClient: TFPHttpClient;
  ResponseStream: TStringStream;
  RequestBodyStream: TMemoryStream;
  FullURL, JsonBody, SafeFrage: string;
  UTF8Bytes: RawByteString;
begin
  Result := '';

  // URL-Zusammenbau ohne Risiko von Doppelslashes
  FullURL := Trim(Self.Site1) + '/openai/deployments/' + Trim(Self.Model1) +
    '/chat/completions?api-version=' + Self.APIVers;

  // Dein funktionierender JSON-Body
  SafeFrage := Frage.Replace('\', '\\').Replace('"', '\"').Replace(#13, '\r').Replace(#10, '\n');
  JsonBody := '{"messages":[{"role":"user","content":"' + SafeFrage + '"}],"max_completion_tokens":16000}';
  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;
```

Autor

Dipl.-Betriebswirt Christoph Wildensee, DBA, MFTA, CISM, CRISC, CDPSE, ist seit vielen Jahren in der Internen Revision der enercity AG, Hannover, tätig. Zwischen 2008 und 2012 war er in Personalunion Datenschutzbeauftragter des Unternehmens und der zugehörigen Netzgesellschaft.