A Security Guide for ABAP SAP Developers

kiuwan
An Idera, Inc. Company

INDEX

ABAP
SAP

Overview

ABAP (Advanced Business Application Programming), the high-level programming language created by software company SAP SE, is used for developing applications for the SAP R/3 platform. While certainly not as ubiquitous in B2B and enterprise applications as, say, Java and Python, being designed specifically for the SAP environment means that its market share is directly tied to that of the German software company.

Some sources say that ABAP/SAP is used by over 31,000 companies in their database, accounting for **0.79% of market share**. The largest segments for ABAP/SAP use are Information Technology and Services and Computer Software, followed by, perhaps less predictably but also understandably, Food and Beverage, Oil and Energy, and Retail. SAP clients using ABAP are primarily located in the United States, followed by India and Germany. Other sources list companies such as Walmart and Ace Hardware as **active users of ABAP/SAP**.

This information paints a picture of a highly corporate, high-level programming language dedicated to SAP applications, which means that unsecured, recklessly developed ABAP code is rife with potential attack vectors to systems that hold valuable, confidential, and highly sensitive data. And data breaches are costly. The **average cost of a data breach** in the U.S. is $9.44 million, and the worldwide average is $4.35 million.

The top security vulnerabilities highlighted in this guide, while not comprehensive, should effectively inform organizations regarding security approaches for developing in the ABAP/SAP language. Understanding the issues underlying these common vulnerabilities will help you understand what measures can be put into place to protect your SAP applications, what preemptive steps can be taken to avoid the issues in the first place, and what principles and concepts may have some implications beyond the issues outlined here:

- SAP's Security Recommendations
- Top 5 Security Risks:
    - ABAP Risk– Injection Flaws
    - ABAP Risk– Cross-Site Scripting
    - ABAP Risk– Broken Authentication and Session Management
    - ABAP Risk– Insufficient Authorization and Access Control
    - ABAP Risk– Security Misconfiguration and Misuse of Mechanisms
- Reliable Security Capabilities

SAP's Security *Recommendations*

In 2017, SAP published "Security Recommendations: A Practical Guide for Securing SAP Solutions." The report goes through most of the important security-relevant topics impacted by the company's recommendations, categorized into five key areas:

- **Security compliance** - Covers security governance, audits, cloud security, and emergency concept
- **Secure operation** - Spans users and authorizations, authentication and single sign-on, support security, and security review and monitoring
- **Secure setup** - Covers secure configuration, communication security, and data security
- **Secure code** - Spans security maintenance of code and customer code security
- **Secure infrastructure** - Involves network security, operating system and database security, and front-end security

All of these key areas and their subcategories are undoubtedly important points of focus when developing in ABAP/SAP, and it is highly recommended that your CI/CD pipeline takes into account the 16 specific topics covered by these five key areas.

The same report also outlines known issues you will commonly encounter when trying to implement SAP's recommendations:

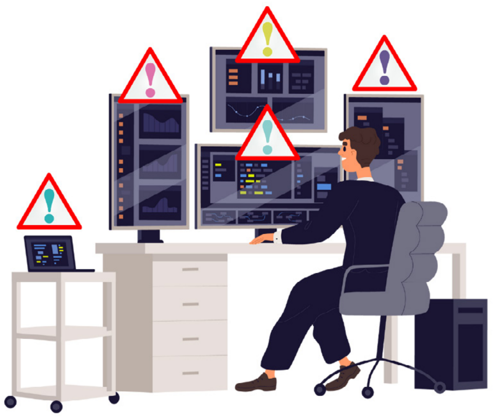
- Poorly established patch processes
- Unencrypted communications between SAP solutions
- Inadequately secured interfaces
- Lack of established data backup processes and emergency processes
- Incorrect or missing security system configurations

The 16 topics and the five known issues are an excellent start to securing your ABAP/SAP code and can drastically reduce the chances of you ever needing to fix problems that arise from the top security risks discussed below.

Additionally, you would also want to pay attention to the topics discussed by SAP's security experts in the community blog. Among them are personal recommendations on ABAP/SAP development security, some inspiration from the software development market at large and how key industry learnings apply to ABAP/SAP developers, and using SAP's own tools to complement your CI/CD pipeline measures.

Top 5 Security Risks When Developing in ABAP/SAP

The top security risks discussed below are injection flaws, cross-site scripting, broken authentication and session management, insufficient authorization and access control, and security misconfigurations and standard mechanism misuse.



Notice that these security risks and vulnerabilities are very similar to those you might find in other programming languages, arguably highlighting a common theme that human error — or at least persistent human negligence — is the ultimate software development security risk. Still, there are distinct factors unique only to ABAP/SAP that you need to be aware of, even if you're familiar with the top security risks in general.

String literals create arrays of static storage during compile time. They contain both a specific character sequence and a null character for termination. Most string literals get referred to by a pointer to characters or an array. Best practices call for only assigning string literals to pointers or arrays made up of const char or const wchart_t.

Developers modifying any part of a string literal results in undefined behavior. For that reason, programmers should avoid assigning string literals to a pointer to a type that is not a const or casting string literals to types that are not a const. Any values returned from the following functions should be treated as a string literal, if their first argument consists of a string literal.

1. Injection Flaws

Injection flaws are a type of security weakness that can occur when untrusted data is used to unexpectedly alter the execution of an ABAP program. Injection flaws can allow an attacker to execute unauthorized ABAP code, access or modify sensitive data, or interfere with the normal operation of an SAP system.

ABAP code is susceptible to SQL injection, remote function call (RFC) injection, and table buffer overflow vulnerabilities.

SQL Injection

SQL injection is a type of code injection attack that can be used to compromise the security of an SAP system. This attack occurs when user-supplied input, such as via an HTTP request or web form, is used to construct and execute SQL statements without proper encoding or validation. This can allow an attacker to execute unauthorized SQL code, access sensitive data, or interfere with the normal operation of an SAP system.

For example, consider the following ABAP code:

```
SELECT * FROM users WHERE name = '[input_name]'
AND password = '[input_password]';


```

If user input is not properly validated, it may be possible for an atacker to enter malicious input that causes the above code to execute unintended actions. For instance, if an attacker entered the following values for [input_name] and [input_password]:

' OR 1=1—' (without quotation marks)

The resulting SQL query would be:

```
SELECT * FROM users WHERE name = '' OR 1=1--''
AND password = '';


```

This query would return all rows from the users table because it evaluates to true regardless of what value is supplied for [input_password]. An attacker could then use this information to gain unauthorized access to sensitive data or systems.

RFC Injection

RFCs are a type of SAP protocol that allows communication between different SAP systems or between SAP and non-SAP systems. An attacker can exploit an RFC injection vulnerability to execute unauthorized ABAP code, access or modify sensitive data, or interfere with the normal operation of an SAP system. This attack occurs when user-supplied input, such as via an HTTP request or web form, is used to construct and execute RFC calls without proper encoding or validation. For example, let's say there is an ABAP program that makes an RFC call to retrieve customer information from a remote system.

An attacker could modify the input data for the RFC call in order to inject malicious ABAP code into the program. When the program executes this malicious code, it could allow the attacker to gain unauthorized access to sensitive data or systems, or interfere with the normal operation of the SAP system.

Table Buffer Overflow

A table buffer overflow vulnerability can occur when too much data is inserted into an ABAP program's internal table. This can cause the ABAP program to crash or allow an attacker to execute unauthorized code. Table buffer overflow vulnerabilities are typically caused by coding errors, such as improper input validation.

For example, if an ABAP program does not properly validate the length of user-supplied input before inserting it into a table, a malicious user could supply excessive data that would cause the program to crash or allow the execution of unauthorized code.

It's important to check if input values will fit into the allocated space for fields in tables.

2. Cross-Site Scripting (XSS)

A malicious user can inject code into an ABAP web application that will be executed by other users who visit the site. When malicious code is executed by victims who visit the web page, it can result in the theft of sensitive information such as cookies or session tokens. XSS attacks can also be used to infect victims with malware or to hijack their sessions, allowing the attacker to take over their account and access sensitive data.

Script Injection

One example of malicious XSS code is script injection. This involves injecting a script tag into the web page that will be executed by the victim's browser. The script can perform any action that the attacker desires, such as stealing cookies or session tokens, redirecting the user to another page, or infecting the user with malware. Cookiejacking and clickjacking are common script injections. The former steals browser cookies to access sensitive info or hijack sessions, while the latter places an invisible frame over a button or link on a web page and intercepts clicks intended for that button or link. The attacker can then use those clicks to perform any desired action, such as redirecting the user to another page or downloading malware onto their machine.

Event Handler Hijacking

Event handler hijacking is another common form of XSS. This involves putting malicious code in an event handler, such as onclick or onmouseover. When the user activates the event (by clicking or mousing over), the malicious code is executed.

Attack Vectors and Best Practices

There are two main ways that XSS attackers can insert their malicious code into ABAP/SAP applications:

1. **Through user input fields:** Attacker-controlled input fields such as search bars, comments sections, and contact forms are often used to inject malicious JavaScript code. When this code is executed by users who visit the page, it can allow the attacker to steal sensitive information or infect the victim's machine with malware.
2. **By exploiting vulnerabilities in application logic:** Some ABAP/SAP applications contain vulnerabilities that can be exploited by attackers to inject malicious code into web pages. For example, an attacker may exploit a security flaw in an application's file upload feature to upload a malicious JavaScript file that is then executed by users who visit the page containing the uploaded file.

There are a few best practices that ABAP/SAP developers can follow to help prevent XSS attacks:

→ ***Validate and sanitize all input data before processing it.*** This includes using proper input validation checks on web forms, browser cookies, and URL parameters.

→ ***Encode all output data before displaying it on a web page.*** This prevents malicious code from being executed by the victim's browser.

→ ***Use a secure development methodology*** such as the SAP Secure Software Development Lifecycle (SSDL).

The good news is that these best practices also help mitigate risks arising from other vulnerabilities, especially similarly patterned ones such as SQL injections.

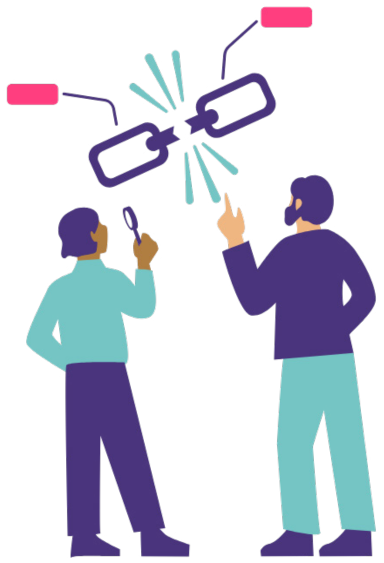
3. Broken Authentication and Session Management

Poorly implemented authentication and session management mechanisms in ABAP applications can enable attackers to gain access to resources or data they should not have access to. When ABAP/SAP development uses weak or easily guessed authentication credentials, it increases the risk of unauthorized access. Additionally, if session management is not properly implemented, attackers may be able to hijack user sessions and gain access to sensitive data.

Weak Authentication Credentials

Attackers can exploit weak authentication credentials in ABAP/SAP code by guessing or brute-forcing passwords, using default passwords, or exploiting vulnerabilities in how the authentication process is implemented.

Attackers can try to guess or brute force passwords in order to gain access to ABAP/SAP systems, often by using common password lists or default passwords, or by trying variations of an organization's name or other easily guessed strings. Additionally, many ABAP/SAP systems come with default accounts that have well-known passwords. Attackers can exploit these accounts to gain access to the system if they are not disabled or changed.



This is one of the most well-known and easily rectified issues not only in software development, but also in cybersecurity in general. Still, you would be surprised how many even in the software industry neglect the most basic of fundamentals. Usually, developers use placeholder credentials initially, meaning to replace them with proper ones down the line. But often, "down the line" never comes, and the placeholders are left holding that place — until disaster strikes.

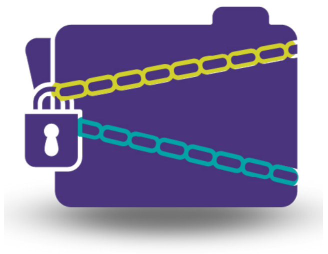
Malicious agents can also exploit vulnerabilities in how authentication is implemented in order to gain access without needing valid credentials. For example, an attacker may be able to bypass the login page altogether and directly send requests that would normally require authentication, usually if the login page did not function correctly or was hosted on a server that was configured to allow unauthenticated access.

Poor Session Management

Session management vulnerabilities can be exploited in several ways, including:

- **Guessing or brute forcing session IDs** - If session IDs are weak or predictable, attackers can try to guess or brute force them to gain access to user sessions.
- **Using cookies without proper security safeguards (e.g., not setting the secure flag)** - Cookies are often used to store session information. However, if they are not properly secured (e.g., by setting the secure flag), they can be intercepted and exploited by attackers.
- **Session ID poisoning (i.e., injecting malicious content into a user's session)** - This attack involves injecting malicious content into a user's session, which can then be executed when the user accesses it. This can lead to data leakage or system compromise.
- **Cross-site request forgery attacks** - Similar to but not the same as XSS, cross-site forgery attacks exploit vulnerabilities in web applications that allow an attacker to inject illegitimate requests that are executed by the target user without their knowledge or consent. This can be used, for example, to hijack a user's session or perform unauthorized actions on behalf of the victim.

4. Insufficient Authorization and Access Control



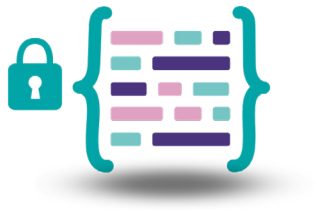
Lack of proper authorization checks within ABAP code can allow unauthorized users access to sensitive functionality or data. There are two primary ways for attackers to take advantage of poor authorization checks on SAP applications: gaining valid credentials, or exploiting flaws in ABAP code that allow unauthorized access to data or functionality. Ultimately, the results are the same: data leakage, corruption, and disruptions in service.

Putting Effective Authorization Checks in Place

Setting up proper authorization checks is a significant barrier that prevents SAP applications from being exposed to security risks. **Some principles to adopt** include:

- Always be certain to program the proper authorization checks and complete their execution.
- When calling a transaction, always check the transaction start authorization.
- In RFC-capable function models, always check business authorizations.
- Always secure critical PAI (Process After Input) events triggered by directly inputting function codes — this is extra sneaky as the corresponding interface element may be deactivated during the PBO (PROCESS BEFORE OUTPUT) event.
- Always double-check whether you're using the right authorization object.
- Do not use a proprietary authorization check.
- Always handle the return values of checks.

Due Diligence Is the Key Preventative Measure

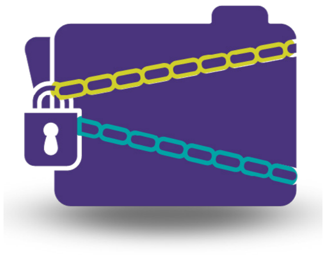


In terms of weak authentication credentials, poor session management, ineffective authorization checks, and insufficient access control measures, there's certainly much to be said regarding due diligence.

Your organization needs to standardize an approach to CI/CD security for ABAP/SAP applications. Naturally, compliance with any external regulatory bodies is included, as are any company-wide guidelines. Your security approach also needs to be compliant and compatible with the requirements and recommendations of SAP itself. Finally, it must be cascaded throughout your organization, with cogent buy-in from all C-suite leaders, middle managers, and dev team leaders, to the point that it's embedded into your development culture as much as it's required in your CI/CD processes.

The human element in application security is not only significant, but it's also critical.

5. Security Misconfigurations and Misuse of Mechanisms



Incorrectly configured security settings in the SAP R/3 system can leave it open to attack. To mitigate the risk of security misconfigurations, developers need to follow best practices for configuring SAP systems and application components. These best practices include keeping system passwords confidential, using strong encryption for data storage and communication, properly securing web services interactions, and limiting access privileges to only those users who absolutely require it.

A lot of security misconfigurations open up your SAP applications to the risks enumerated above. An administrator leaving the password in its default state, for example, is poor access control. Couple that with ineffective authorization checks, and that clears the way for an intentional malicious agent to compromise your application and data security. Another fundamental but often neglected security configuration is not properly securing web services, which leaves otherwise confidential information readily accessible.

Additionally, SAP offers mechanisms to protect against attacks: client separation, logging, and authorization management. However, misconfiguring or misusing these mechanisms also inadvertently abuses the trust that companies put into them — you might have a false sense of security, thinking that they're available when in fact they're misconfigured or used incorrectly.

Common security issues arise when developers bypass client separation, most commonly to access data in other clients or fix an issue that spans multiple clients (performance testing and debugging). This should only be done when completely necessary, and code that bypasses client separation should be reviewed by a senior developer or security expert before it is deployed to production.

There are also known issues that can be caused by database logging being deactivated. This is generally done when performance needs to be improved or if space in the log file is limited. Deactivating logging can make it difficult to troubleshoot problems that occur after the changes are made, so it should only be done when necessary. In a similar vein, vulnerabilities arise when custom programs alter business data in SAP applications without creating change documents. ABAP provides standard update functions that allow programs to change business data while also creating appropriate change documents, but if a program directly accesses database tables on its own, no change documents are created automatically.

These examples above are dangerous because they invalidate what would otherwise be comprehensive security measures. A dev team would likely assume their measures are in place, properly configured, and used correctly.

Reliable Security Capabilities for Your *ABAP/SAP Applications*

If you're looking to ramp up the security around your ABAP/SAP applications, this guide is only a starting point. Adopting best practices and staying current on security vulnerabilities related to ABAP/SAP are important, but partnering with a code security solution like Kiuwan can take your application development security to the next level.

Kiuwan's static application security testing (SAST) tool — Kiuwan Code Security — is compliant with popular standards like OWASP and Common Weakness Enumeration (CWE). Plus, our software composition analysis tool — Kiuwan Insights Open Source — can help reduce risk from third-party components, address vulnerabilities, and ensure license compliance throughout your software development life cycle.

ABAP/SAP applications are just one of the many languages we secure at Kiuwan. In addition to covering ABAP, we provide code security for over 30 coding languages in all. Want to see how it works? Schedule a demo with us today.

YOU KNOW ***CODE,*** *WE KNOW* ***SECURITY!***

GET IN TOUCH:

Headquarters
2950 N Loop Freeway W, Ste 700
Houston, TX 77092, USA

United States **+1 732 895 9870**
Asia-Pacific, Europe, Middle East and Africa **+44 1628 684407**
contact@kiuwan.com
Partnerships: **partners@kiuwan.com**