SAP History Fail: Why XOR is Still Not Secure

Jonathan Stross, Lead Researcher at Pathlock Inc.,
Julian Petersohn, Principal Systems Engineer at Fortinet Inc.

Abstract—SAP, a global provider of enterprise resource planning (ERP) software, supports the operations of over 400,000 organizations worldwide, including critical sectors such as finance, healthcare, and manufacturing. The security of SAP environments is essential due to their role in managing sensitive data and critical processes. This paper examines vulnerabilities in the user input history function of SAP GUI, which serves as the primary interface for interacting with backend systems. We highlight weaknesses, including inadequate encryption methods and the reliance on insecure techniques such as XOR. These vulnerabilities expose sensitive data and may pose a risk to organizational operations. A comprehensive analysis of the architectural flaws and specific vulnerabilities is provided, along with actionable recommendations to mitigate the risks and insights into SAP's subsequent patches.

TABLE 3: Vulnerability Overview

CVE Number	CVSS Vector	Severity
CVE-2025-0056	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N	Medium (6.0)
CVE-2025-0055	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N	Medium (6.0)

Index Terms—SAP Security, XOR Encryption, Vulnerability Analysis, Enterprise Resource Planning, Secure Software Design, CVE-2025-0055, CVE-2025-0056

1 Introduction

SAP is a widely used enterprise resource planning (ERP) solution, central to managing critical business functions with significant implications for security, confidentiality, and availability. Given the sensitive nature of the data and processes handled by SAP systems, safeguarding their security is paramount. Certain functionalities within the SAP ecosystem, such as user access controls and authorization mechanisms, are well researched and understood, with established best practices for mitigating risks. However, other features, such as the user input history in the SAP GUI, have not been extensively documented in public research, which may result in overlooked vulnerabilities related to data exposure

This paper focuses on the security of user input history in SAP GUI for Java and SAP GUI for Windows, specifically examining shortcomings in their encryption methods. By analyzing these architectural vulnerabilities, the paper provides insights into potential risks and offers recommendations for mitigating these weaknesses to enhance the security of SAP environments.

2 Background

2.1 SAP Architecture

SAP deployments are designed using a client-server architecture to support enterprise resource planning (ERP) capabilities. This architecture enables organizations to manage complex business processes, including financial transactions, supply chain logistics, and human resource operations. The SAP system architecture consists of the following key components:

2.1.1 Presentation Layer

The presentation layer constitutes the primary user interface, facilitating end-user interactions with the SAP system. This layer is implemented through the SAP Graphical User Interface (SAP GUI), which is offered in multiple variants tailored to diverse platforms and operational requirements:

- **SAP GUI for Windows**: A dedicated desktop application developed for the Windows operating system.
- **SAP GUI for Java**: A platform-independent client compatible with macOS, Linux, and Windows environments, providing versatility across different system architectures.
- **SAP GUI for HTML**: A browser-based interface accessible via standard web browsers, removing the necessity for installing local client applications.

The presentation layer enables essential tasks such as data input, report generation, and execution of business processes.

2.1.2 Application Layer

The application layer is responsible for executing business logic within the SAP system. It comprises one or more application servers that facilitate the functionality of various SAP modules, such as Financial Accounting (FI), Materials

• *J. Stross and J. Petersohn are cybersecurity researchers specializing in enterprise systems security. Correspondence: jonathan.stross@pathlock.com, julian@petersohn.it*

Management (MM), and Sales and Distribution (SD). The primary responsibilities of the application layer include:

- Processing user requests received from the presentation layer (including authorization).
- Executing business rules, workflows, and associated logic.
- Interacting with the database layer for data retrieval and updates.

The architecture of the application layer supports horizontal scalability, allowing organizations to manage high transaction volumes effectively by distributing the computational workload across multiple application servers. This scalability ensures robust performance and reliability in complex enterprise environments.

2.1.3 Database Layer

The database layer is responsible for the storage, retrieval, and management of business data and configuration settings within the SAP system. SAP systems are compatible with various relational database management systems (RDBMS), including SAP HANA, Oracle, and Microsoft SQL Server. The key features and functions of the database layer include:

- **Data Storage**: Centralized storage of master data, transactional data, and configuration settings, ensuring efficient access and consistency across the system.
- **Availability**: Robust mechanisms such as failover strategies and data replication to maintain uninterrupted access to critical data and ensure high system reliability.

All layers need to be observed and maintained to ensure the secure operation of the SAP system.

2.2 Mathematical Foundation

The XOR (Exclusive OR) operation is a fundamental binary operation used extensively in computer science and cryptography. XOR is defined as:

$$A \oplus B = \begin{cases} 1 & \text{if } A \neq B \\ 0 & \text{if } A = B \end{cases}$$

2.2.1 Properties of XOR

XOR has several key properties that make it useful in cryptographic applications:

1) **Commutativity:** $A \oplus B = B \oplus A$
2) **Associativity:** $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
3) **Identity:** $A \oplus 0 = A$
4) **Inverse:** $A \oplus A = 0$

These properties enable XOR to be used as a building block for simple encryption schemes.

2.2.2 Example: XOR in Cryptography

In a simple cryptographic context, XOR can be used to encrypt and decrypt data:

- Encryption: Ciphertext $=$ Plaintext $\oplus$ Key
- Decryption: Plaintext $=$ Ciphertext $\oplus$ Key

The reversible nature of XOR ensures that applying the same key twice cancels out the effect of encryption, recovering the original plaintext.

2.2.3 Limitations of XOR

Although XOR is computationally efficient, its security depends entirely on the randomness and secrecy of the key. Reusing keys or applying predictable patterns makes XOR-based encryption vulnerable to attacks. For instance, if the same key is used across multiple messages, attackers can exploit the relationship:

$$\text{Ciphertext}_1 \oplus \text{Ciphertext}_2 = \text{Plaintext}_1 \oplus \text{Plaintext}_2$$

This reveals information about the plaintexts without requiring knowledge of the key. Given this weakness, it is important to introduce additional complexity to mitigate vulnerabilities.

2.2.4 Cascading XOR

Cascading XOR refers to the repeated application of the XOR operation across multiple inputs. For example, given inputs $A, B, C, \ldots, N$, cascading XOR can be expressed as:

$$R = A \oplus B \oplus C \oplus \ldots \oplus N$$

This operation retains the properties of commutativity and associativity, which means the order of operations does not affect the final result.

In encryption, cascading XOR becomes more robust when combined with **key-dependent transformations** and **iterative state propagation**, adding additional layers of complexity and security.

2.2.4.1 Principles at Play:

- **Key Scheduling:** The key is used to generate unique transformations for each character. Each iteration of the process ensures that the key is applied uniquely to subsequent characters, effectively behaving as a "seed" for cascading XOR operations. This ensures that each input is uniquely influenced by the key.
- **Cascading State Propagation:** Changes applied to earlier characters propagate to subsequent ones through iterative XOR operations. This cascading effect creates a cumulative transformation, where the state of each character influences the encryption of later characters.
- **Pseudo-Randomness via XOR:** XOR inherently introduces non-linear transformations. When combined with varying inputs (e.g., characters and corresponding key values), it generates outputs with high entropy. This pseudo-randomness is a key feature for robust encryption.

By combining these principles, cascading XOR ensures that each input is influenced not only by its own key-dependent transformation but also by transformations applied to previous inputs. This approach enhances the complexity and entropy of the resulting encrypted data, making it more resistant to attacks.

2.3 Understanding SAP GUI User History Storage

The SAP GUI user history functionality is designed to enhance usability by storing frequently entered inputs, such as usernames, field values, and search queries. This feature aims to improve user efficiency by reducing the need to repeatedly type commonly used entries.

2.3.1 SAP GUI History Functionality

The SAP GUI history functionality is a feature designed to enhance user efficiency by retaining a record of inputs provided in specific fields across various SAP transactions and interfaces. This functionality enables the system to store user-provided data, such as customer names or transaction-specific parameters, in fields used during selection criteria or other input processes. Upon subsequent interactions, users can easily access and select previously entered values from a dropdown list or similar interface element. This feature aims to streamline repetitive tasks, reduce manual entry errors, and improve overall user productivity by leveraging historical input data.

2.3.2 SAP GUI History Data Location

The user history data is stored locally on the client system where the SAP GUI is installed. SAP GUI for Windows and SAP GUI for Java leveraging different mechanism and locations to store the history entries. Specifically:

- **SAP GUI for Windows** does save the history data by default in the following directory:

  `%APPDATA%\LocalLow\SAPGUI\Cache\History`

- **SAP GUI for Java** installed on Linux, the history data is stored at:

  `$HOME/.SAPGUI/Cache/History`

- **SAP GUI for Java** installed on MacOS, history data is stored at:

  `$HOME/Library/Preferences/SAP/`

  `Cache/History`

- **SAP GUI for Java** installed on Windows, history data is stored at:

  `%APPDATA%\LocalLow\SAPGUI\Cache\History`

2.3.3 Contents of the SAPGUI user input history

The SAP GUI user input history records nearly all data entered into clear-text input fields within the SAPGUI interface. This includes fields such as ALV grids, input line feeds, and dropdown lists.
This can include **sensitive or personally identifiable information (PII)** such as addresses, user IDs, or other input data. However, it excludes certain types of fields:

- **Password Fields:** Inputs marked as password fields, where characters are masked (e.g., represented by asterisks).
- **BLOB Fields:** Text edits which contain unstructured data mainly stored as Binary Large Objects (BLOBs).

Security Conciderations

Security must be implemented across all layers of the SAP architecture to ensure the protection of sensitive information and the integrity of the system. A critical aspect is the use and correct implementation of secure encryption algorithms to protect data against unauthorized access and breaches. Addressing vulnerabilities in each layer, from the presentation to the database layer, ensures a comprehensive defense against potential threats.
While the input history functionality improves usability by allowing users to quickly access previously entered values, it introduces risks, **as this information is stored locally on the client**.
This paper delves into the User Input History, a component of the Presentation Layer, **that highlights critical vulnerabilities** within this feature.

3 Structure of the History file

The structure and method of storing User Input history vary depending on the SAP GUI version and edition (for Windows or Java). The research in this paper was conducted on SAP GUI for Windows version 800 Patch Level 0 and SAP GUI for Java version 780 Rev 8.

3.1 SAP GUI for Java

To store the User Input History, the SAP GUI for Java generates files with uniquely identifiable names in the designated directory for each operating system. These files contain Java serialized objects, which include plain text representations of user inputs entered via the SAP GUI. Due to the lack of any encryption within Java serialized objects by default, the data is stored unencrypted.

3.1.1 Extracting content of User History

The SAP GUI for Java stores user input history in files with uniquely identifiable names, located in designated directories for each operating system. However, the security measures implemented for these files appear to be minimal. Analyzing the contents requires only basic tools and no advanced technical expertise. To examine the stored data, basic operating system commands can be used to display the the Java serialized objects without requiring any decryption.

```
00000000: aced 0005 7372 0032 636f 6d2e 7361 702e  ....sr.2com.sap.
00000010: 706c 6174 696e 2e62 6173 652e 6869 7374  platin.base.hist
00000020: 6f72 792e 4869 7374 6f72 794d 616e 6167  ory.HistoryManag
00000030: 6572 2448 6973 746f 7279 21d7 b96f 5771  er$History!..oWq
00000040: f866 0300 034c 000c 6d48 6973 746f 7279  .f...L..mHistory
00000050: 4461 7461 7400 124c 6a61 7661 2f75 7469  Datat..Ljava/uti
00000060: 6c2f 5665 6374 6f72 3b4c 000c 6d54 656d  l/Vector;L..mTem
00000070: 7048 6973 746f 7279 7100 7e00 014c 0006  pHistoryq.~..L..
00000080: 7468 6973 2430 7400 2c4c 636f 6d2f 7361  this$0t.,Lcom/sa
00000090: 702f 706c 6174 696e 2f62 6173 652f 6869  p/platin/base/hi
000000a0: 7374 6f72 792f 4869 7374 6f72 794d 616e  story/HistoryMan
000000b0: 6167 6572 3b78 7073 7200 106a 6176 612e  ager;xpsr..java.
000000c0: 7574 696c 2e56 6563 746f 72d9 977d 5b80  util.Vector..}[.
000000d0: 3baf 0103 0003 4900 1163 6170 6163 6974  ;.....I..capacit
000000e0: 7949 6e63 7265 6d65 6e74 4900 0c65 6c65  yIncrementI..ele
000000f0: 6d65 6e74 436f 756e 745b 000b 656c 656d  mentCount[..elem
00000100: 656e 7444 6174 6174 0013 5b4c 6a61 7661  entDatat..[Ljava
00000110: 2f6c 616e 672f 4f62 6a65 6374 3b78 7000  /lang/Object;xp.
00000120: 0000 0000 0000 0175 7200 135b 4c6a 6176  .......ur..[Ljav
00000130: 612e 6c61 6e67 2e4f 626a 6563 743b 90ce  a.lang.Object;..
00000140: 589f 1073 296c 0200 0078 7000 0000 0a74  X..s)l...xp....t
00000150: 0005 5553 5230 3270 7070 7070 7070 7070  ..USR02pppppppp
00000160: 7878                                     xx
```

Fig. 1: Example Content of Example History file

3.1.2 Findings

The examination reveals plain text representations of user inputs stored in the files. The lack of encryption in these files means that their contents are accessible to anyone with file access privileges.

3.1.3 Implications

The lack of encryption increases the risk of unauthorized access to input history data of the user input history. the unencrypted state of the data suggests that applying even basic encryption could mitigate unauthorized access. to safeguard sensitive information from unauthorized access. Enhancing these security measures would align with best practices for protecting user data.

3.2 SAPGUI for Windows

The SAP user history database, referred to here as `SAPHistory<WINUSER>.db`, is a SQLite3 database file utilized by the SAP GUI to store user input history. This database facilitates the functionality of retaining previously entered data for certain input fields. A closer examination of its structure reveals the following key fields:

- **HistFieldname:** Represents the clear-text name of the dynamic program (dynpro) input field. This field allows the SAP GUI to associate stored history data with specific input elements in the interface.
- **HistTime:** A timestamp indicating when the input was recorded. This enables chronological tracking of input history entries.
- **HistEntry:** Contains an encoded or encrypted string representing the input value. The exact encoding or encryption method used here is critical to the security of the stored data, as it protects potentially sensitive information from direct exposure.

In the following sections, we will explore the encoding and encryption methods implemented within the `HistEntry` field. The data used for this analysis is sourced from a controlled lab environment, enabling detailed evaluation and experimentation.

3.2.1 Decoding the Ciphers

The SAP GUI for Windows employs an iterative XOR-based mechanism to encode user input history. For example, given the input string `USR02` (User Master Data table), the resulting encoded `HistEntry` value is `55b0966d76`. This process utilizes a sequence of keys, applied iteratively, to encode each character of the input string.

3.2.2 Encoding Mechanism

The encoding process involves the following components:

- **P:** The plaintext input string, where each character is converted to its hexadecimal ASCII representation.
- **C:** The encoded output in hexadecimal, calculated using the XOR operation.
- **K:** The key sequence, where each character in `P` is encoded using a unique key. This key is then applied iteratively to all subsequent characters until the next character in `P` is encoded with the following key.

3.2.3 The Forensic Analysis Process

The encoding mechanism can be analyzed using a known plaintext attack. By comparing the plaintext string with its corresponding encoded value, the sequence of keys can be deduced. The following equations describe the encoding process:

1) For each character in the input string:

$$C[i] = P[i] \oplus K[i], \quad i \in \{1, 2, \ldots, n\},$$

where n is the length of the plaintext string.

2) For cascading key application:

$$P[i+n] = P[i+n] \oplus K[i],$$
$$n \in \{1, \ldots, |P| - i\}$$

3.2.4 Nested Iterative Encoding

The encoding process can be visualized as two nested loops:

- The outer loop iterates over each character in the input string.
- The inner loop applies the current key to all subsequent characters in the string, encoding them in a cascading fashion.

This iterative process highlights the positional dependence of the encoding mechanism and underscores the potential vulnerabilities in the design. By analyzing the cascading effect and deducing the keys, the encoded data can be effectively reversed, revealing the original input string.

TABLE 4: Step-by-Step Encryption Process

Step	Operation	Plaintext State	Output
1	$C[0] = 0x55 \oplus 0x00 = 0x55$ Cascade: $P[1..4] \oplus 0x00$ (no change)	$P = [55, 53, 52, 30, 32]$	$C = [55]$
2	$C[1] = 0x53 \oplus 0xE3 = \texttt{B0}$ Cascade: $P[2] \oplus 0xE3 = 0xB1$, $P[3] \oplus 0xE3 = 0xD3$, $P[4] \oplus 0xE3 = 0xD1$	$P = [55, \texttt{B0}, \texttt{B1}, \texttt{D3}, \texttt{D1}]$	$C = [55, \texttt{B0}]$
3	$C[2] = 0xB1 \oplus 0x27 = 96$ Cascade: $P[3] \oplus 0x27 = 0xF4$, $P[4] \oplus 0x27 = 0xF6$	$P = [55, \texttt{B0}, 96, \texttt{F4}, \texttt{F6}]$	$C = [55, \texttt{B0}, 96]$
4	$C[3] = 0xF4 \oplus 0x99 = \texttt{6D}$ Cascade: $P[4] \oplus 0x99 = 0x6F$	$P = [55, \texttt{B0}, 96, \texttt{6D}, \texttt{6F}]$	$C = [55, \texttt{B0}, 96, \texttt{6D}]$
5	$C[4] = 0x6F \oplus 0x19 = 76$	Final P state	$C = [55, \texttt{B0}, 96, \texttt{6D}, 76]$

3.2.5 Encoding Example

In the Example of Table 1, we reuse the string `USR02`, representing the user master data table, with the resulting encoded value `55b0966d76`.

3.2.6 Observations

The cascading application of the key introduces a dependency between each character and all preceding characters. This ensures that the result for any character depends on the cumulative effect of all previous characters and the initial key. This key is being reused, so all entries in `HistEntry` have the key values for the 5 characters in common.

3.2.7 Findings

The technical analysis of the SAP GUI for Windows encoding mechanism reveals the following:

- The encoding process is based on XOR operations with a sequence of keys applied in a cascading fashion—each key byte modifies the current character and all subsequent characters.
- This design introduces a dependency chain, where each character is affected by the cumulative effect of prior characters and keys.
- Encoded values are stored as hexadecimal strings in the `HistEntry` field, where each byte of plaintext is represented by two characters, exposing input length and enabling inference attacks.
- Entry field names, such as `DATABROWSE-TABLENAME`, remain in cleartext, allowing attackers to guess likely input values based on well-known SAP terminology.
- The use of a limited and predictable SAP character set (letters, digits, umlauts, and symbols) allows narrowing down the possible key space—especially for the first key byte—by eliminating invalid character decodings.
- The mechanism is reversible: with access to one known plaintext and its corresponding encoded output, the full key sequence (e.g., `0x00`, `0xE3`, `0x27`, `0x99`, `0x19`) can be derived and reused to decode other strings of the same format.

3.2.8 Implications

The observed encoding design has several critical implications for the security of SAP GUI input history:

1) **Low Resistance to Cryptanalysis:** The simplicity and predictability of the XOR operation—combined with static key reuse—make the mechanism highly susceptible to known plaintext attacks.
2) **Reusability of Recovered Keys:** Once a key sequence is derived for one known input, it can be used to decode any other encoded entry of the same length. This creates a systemic weakness in environments with repetitive user input.
3) **False Sense of Security:** The cascading logic adds procedural complexity but lacks properties associated with modern cryptographic robustness. It can be reversed with minimal effort and no brute force.
4) **Non-Compliance with Security Standards:** The lack of strong, standardized encryption and proper key handling means the mechanism does not meet modern requirements for protecting sensitive or regulated data.

3.3 Exploiting the XOR-Based Encoding Mechanism

The XOR-based encoding mechanism exhibits fundamental cryptographic weaknesses, particularly its vulnerability to known-plaintext attacks. Due to the deterministic and reversible nature of XOR operations, the full key sequence can be recovered from a single plaintext-ciphertext pair. Once obtained, this allows the decryption of other ciphertexts with equivalent structure or length. encoded with the same mechanism.

This characteristic exposes a critical flaw: the absence of entropy, key separation, and forward secrecy. As a result, the scheme is easily reversible and inadequate for securing sensitive data.

4 Impact of Exploitation

Although password fields are excluded from the SAP GUI input history, the vulnerability still represents a potential security and compliance risk due to the sensitive nature of stored input data.

4.1 Exposure of Sensitive Information

Even without passwords, input history may include:

- **Usernames and user IDs**, which can aid in targeted phishing or brute-force attacks.
- **Personnel or customer numbers**, often linked to internal HR or CRM records.

- **Social Security Numbers (SSNs), national IDs, or tax identifiers**, especially in fields used in public sector or payroll modules.
- **Bank account or credit card numbers**, if used in financial transaction inputs.
- **Internal SAP table names and business objects**, which can reveal operational logic or give attackers insight into backend structures.

4.2 Compliance and Legal Implications

Storing such data on the client device using a weak or reversible encryption method — such as the XOR-based scheme analyzed in this paper — may still constitute a violation of regulatory requirements, especially when it does not meet cryptographic standards for protecting sensitive information.

Although the data is not stored in plain text, the simplicity of the encoding mechanism means it can be trivially reversed using known or guessed inputs. This level of protection may be insufficient under laws and industry standards such as:

- **GDPR** – Encryption must be "state of the art" and appropriate to the risk. Weak encoding that can be reversed without brute-force computation likely fails this threshold.
- **PCI DSS** – Requires strong cryptographic protection for cardholder data. XOR-based schemes without key management or entropy controls do not qualify.
- **HIPAA** – Calls for technical safeguards to ensure confidentiality. Weak reversible encoding of identifiers or other PII could result in non-compliance.

Encryption methods that lack key management and strong cryptographic properties may not meet regulatory compliance requirements, as outlined in standards such as GDPR or PCI DSS.

4.3 Operational and Reputational Risks

The ability for an attacker with local access or malware to extract cleartext history entries from the file system introduces additional risks:

- **Credential-based attacks**, using known usernames or IDs for impersonation.
- **Business process exposure**, by revealing common transactions and internal structures.
- **Loss of trust or audit failures**, particularly during security assessments or compliance reviews.

While passwords are not stored, the retained input history still provides an attacker with valuable contextual and structured information that can be leveraged to escalate privileges or gather reconnaissance information.

5 Recommendations and Mitigation's

The vulnerabilities are summarized in Table 3, detailing their CVSS vectors, severity scores, and potential impacts. Although the assigned CVSS score for these vulnerabilities is 6.0 (Medium), this score primarily reflects technical exploitability and impact on system confidentiality, integrity, and availability. However, the CVSS framework does not account for several critical factors, including:

- **Regulatory non-compliance**, such as potential violations of GDPR, PCI DSS, or HIPAA.
- **Business impact**, including reputational damage, loss of customer trust, or legal liability.
- **Data classification sensitivity**, where exposed fields may contain personally identifiable or business-critical information.

In environments that process sensitive or regulated data, even vulnerabilities with a Medium CVSS score may present significant security risks. Organizations should assess the specific context in which SAP GUI is deployed and account for these factors when prioritizing remediation.

5.1 Immediate Workaround for SAP GUI for Java

The immediate workaround to mitigate the risks associated with the storage of user input history is to disable the input history functionality. This can be accomplished through the Preferences dialog in SAP GUI for Java. For detailed instructions, refer to the SAP Online Documentation [5].

Additionally, it is recommended to delete all existing history files stored locally in the input history folder to ensure that previously entered data is no longer accessible. This folder is typically located in the following directories based on the operating system mentioned in 2.3.2.

Disabling the input history ensures that no further data is stored locally, and deleting the existing files removes any previously saved user inputs, mitigating potential security risks.

5.2 Immediate Workaround for SAP GUI for Windows

To disable the input history functionality in SAP GUI for Windows, configure the DWORD registry value `DisableHistory` with the value of `1`. This setting can be applied under either `HKEY_LOCAL_MACHINE` (for every user on the PC) or `HKEY_CURRENT_USER` (only for the current user) in the following registry location:

```
{HKLM/HKCU}\software\sap\SAPGUI Front
    \SAP Frontend Server\LocalData
```

In addition to disabling the history feature, it is recommended to delete all existing history files to ensure that previously stored input data is removed. Specifically, all `.db` files in the history directory should be deleted. The exact location is described in chapter 2.3.2.

For detailed guidance on disabling the history functionality, consult the official SAP GUI documentation [1].

As a long term solution, SAP allows to exclude certain fields from being logged in the User Input History. More details can be found in the SAP Support Portal [6]

5.3 Patches and Updates

SAP has released security notes with the respective patches and recommendations to address these vulnerabilities.

5.3.1 SAPGUI for Java Patch

As long-term solution, SAP has implemented a encryption function to keep history entries secure. The encryption functionality is enabled by default with SAP GUI for Java 7.80 Patch Level 9 or higher or SAP GUI for Java 8.10. Detailed information can be found in SAP Note 3502459 [2].

5.3.2 SAPGUI for Windows Patch

As a long-term solution, SAP has revised and enhanced the encryption function for history entries using the latest cryptographic technology provided by the Windows Operating System. In case this is not available, a fallback to the previous encryption will occur. In case this is also not possible, the SAP GUI will disable the history functionality completely to avoid any unencrypted data. To disable or enable this behavior, configure the DWORD registry value for `HistoryIgnoreDisabledEncryption` with the value of `0` or `1` at the following location:

```
HKey_Local_Machine\software\sap\SAPGUI Front
    \SAP Frontend Server\LocalData
```

This capability has been introduced with SAP GUI for Windows 8.00 Patch level 9 and later. Detailed information can be found in SAP Note 3472837 [3].

5.3.3 SAP GUI for HTML (SAP WebGUI)

In correlation with SAP GUI for Windows and SAP GUI for Java, SAP has identified a similar behavior in the SAP GUI for HTML. SAP GUI for HTML stores history data within the Browser that is used by the user in an unencrypted format. Detailed information can be found in SAP Note 3503138 [4]. As a workaround, SAP recommends the following:

- Disable the input history by setting the parameter `~sapdisableinputhistory` to the value of `1` (SAP Note 2156174 [8])
- Reduce the user entered data retention time by setting the parameter `~data_aging_default` to the value of the amount of days to keep (SAP Note 2660665 [9])
- And remove the ability to change the retention time by a client by setting the parameter `~data_aging_readonly` to the value of `1` (SAP Note 2660665 [9])

6 Conclusion

This research highlights significant vulnerabilities within the SAP GUI, underscoring the need for robust encryption mechanisms and adherence to secure software design principles. The findings emphasize that sensitive data stored on client-side systems is inherently at risk of exposure. Given the relative ease with which input history files can be accessed and exploited, these vulnerabilities meet criteria that indicate a need for timely remediation.

Organizations leveraging SAP systems must adopt proactive measures to safeguard their sensitive data and operations. Applying updates and patches provided by SAP is a critical step in mitigating known vulnerabilities. Regular updates not only address existing issues but also protect systems from emerging threats. However, technical measures alone are insufficient; adopting a layered security strategy that includes constant monitoring, incident response readiness, and regular security audits is essential to maintaining a resilient environment.

The findings suggest that storing sensitive data on client-side systems introduces security risks that can be mitigated by server-side handling with encryption. The results indicate that legacy software implementations may retain vulnerabilities that benefit from systematic review. Addressing these vulnerabilities is not only about technical remediation but also about fostering a security-first mindset across organizations. Collaboration between researchers, developers, and cybersecurity teams is essential to implementing best practices, addressing weaknesses, and maintaining secure enterprise resource planning (ERP) environments.

Acknowledgments

The authors express their gratitude to the SAP Security Response team, and the responsible SAP development teams, for their prompt efforts in addressing the identified vulnerabilities and working on the security fixes. The team's clear and effective communication throughout the process was instrumental in facilitating a timely resolution. Both authors would also like to thank the OWASP Core Business Application Security Project [7] for providing a platform and community to discuss and exchange on SAP Security Research and Business Application Security in general. Both authors are grateful to Prof. Dr. Julia Kallrath from Hochschule Darmstadt (HDA) for the helpful insights and valuable suggestions, which contributed to improving the quality of this work.

References

[1] SAP, "Input History: disable via registry key" [Online]. Available: https://me.sap.com/notes/925639. [Accessed: Jan. 02, 2025].
[2] SAP, "SAP GUI for Java patch" [Online]. Available: https://me.sap.com/notes/3502459. [Accessed: June 07, 2025].
[3] SAP, "SAP GUI (Windows) patch" [Online]. Available: https://me.sap.com/notes/3472837. [Accessed: Jan. 13, 2025].
[4] SAP, "SAP GUI for HTML (SAP WebGUI) patch" [Online]. Available: https://me.sap.com/notes/3503138. [Accessed: June 07, 2025].
[5] SAP, "SAPGUI for Java Prefrences" [Online]. Available: https://help.sap.com/docs/sap_gui_for_java/e665f2b67dbd4328ab6bd9e029b84581/46a979cfcef54b5ba7ece258856fdca4.html. [Accessed: Dec. 23, 2024].
[6] SAP, "History: disable history for specific fields in registry" [Online]. Available: https://me.sap.com/notes/924376. [Accessed: Jan. 02, 2025].
[7] OWASP, "OWASP Core Business Application Security Project" [Online]. Available: https://owasp.org/www-project-core-business-application-security/. [Accessed: Jan. 02, 2025]
[8] SAP, "SAP GUI for HTML - Disabling local history and typeahead" [Online]. Available: https://me.sap.com/notes/2156174. [Accessed: June 07, 2025].
[9] SAP, "SAP GUI for HTML - Data aging" [Online]. Available: https://me.sap.com/notes/2660665. [Accessed: June 07, 2025].