

HOW TO PREVENT THE THREE MOST COMMON SAP HACKS

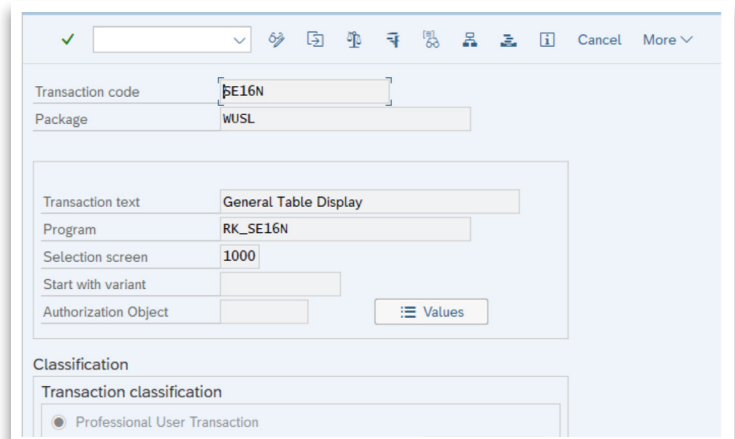
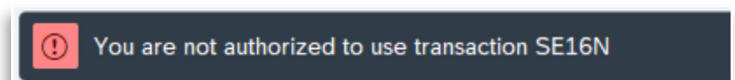
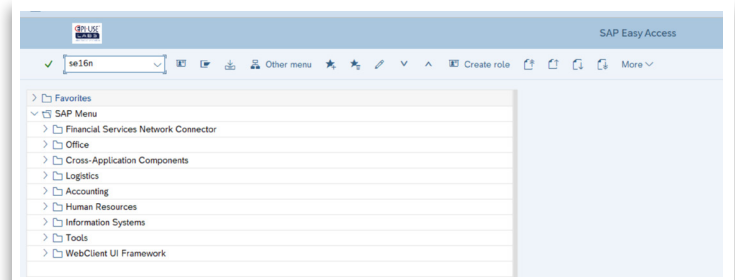
This information is not intended to enable someone to access sensitive information, or carry out activities for which they are not permitted by the organization. Any use of these techniques is entirely the responsibility of the person doing them. Instead, the intention is to show those responsible for system and authorization administration some potential risks in their system and how to mitigate them.

Hack 1: Using a program or function module call rather than a transaction code

When a user enters a transaction, either via the command field at the top left or from their menu, the SAP kernel triggers an authority check against the object S_TCODE for the transaction code entered. This authorization object is often used to see what someone can do in a system, or to make sure they can't run something they're not supposed to. Here we are logged on with a technical user with very little functional authorizations. And when they attempt to run the transaction code SE16N for the data browser they receive an error.

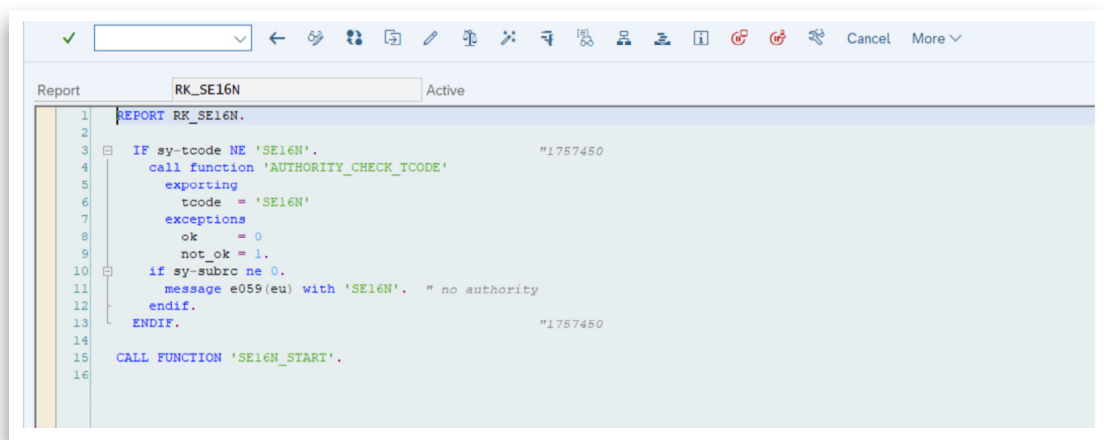
But if we look at the transaction code in SE93 we can see the program calls (see screenshot to the right).

And I don't necessarily need access to SE93 in this particular system; I could view this on another system. In this case, there isn't an additional Authorisation Object linked to the transaction code, but what interests our hacker is the program name. If the program is an executable one

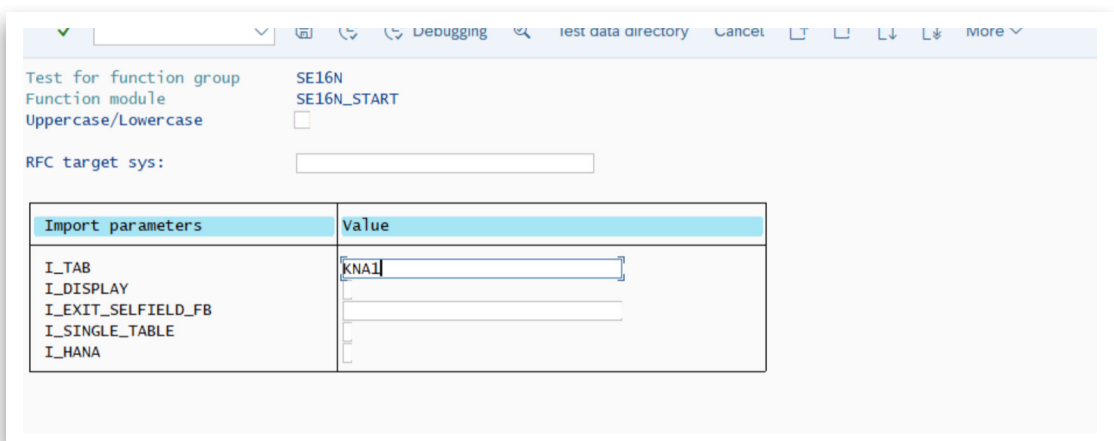




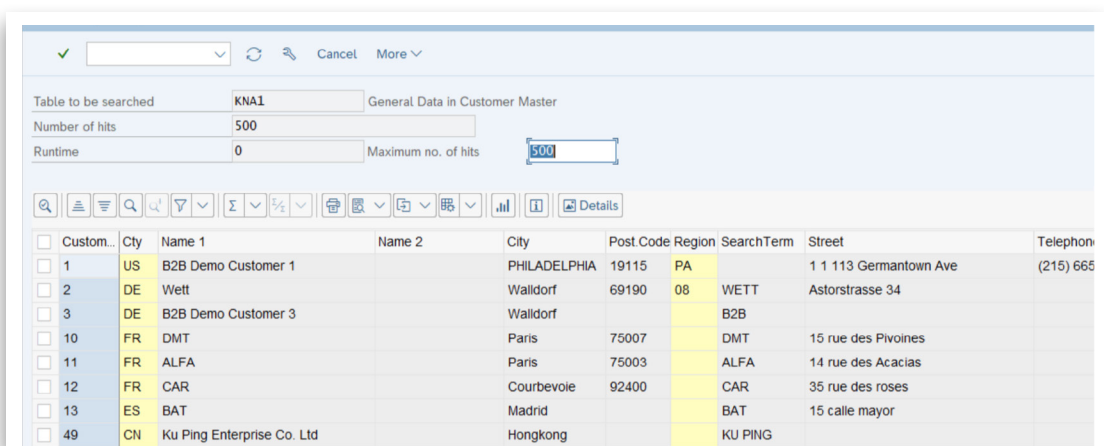
A look at the code though tells us that the program has been protected:



The same S_TCODE check that would happen with the transaction is called at the start of the program code BUT directly below it I see another way in. The program calls a Function Module, and my technical user has access to test function modules in SE37:

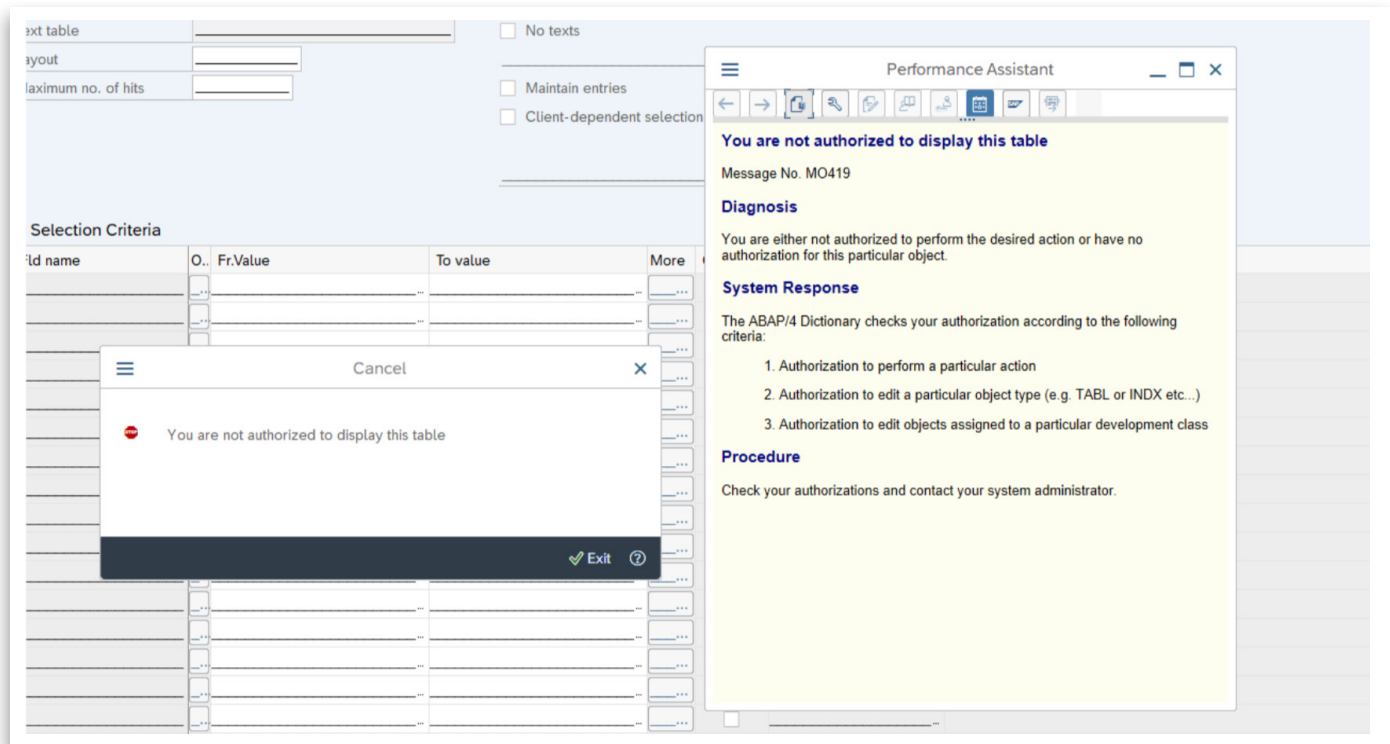


which gives my hacker the chance to enter a table name and then execute to see the data:



Mitigations

The first way to protect against this is the effective use of the S_TABU_DIS object. Here I was able to see a table with customer details but if I tried to view employee data, I was not able to:



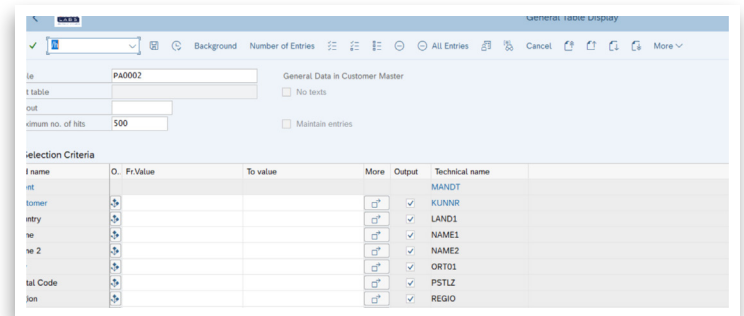
But that remediation is specific to the use of a program or function module to gain access to SE16N.

The more important point is to limit developer access in production systems but also ANY system with real data. This technique could be used to action something the user is not supposed to do in a production system, for example, release a PO; but if sensitive data is in test systems it could also be used there to gain access to personal data. An unauthorised user downloading a table of sensitive data and selling it to the highest bidder is unforgivable in the world today. While more people probably access the production system, you may find people from more organizations accessing your test and development systems (third parties providing development support or testing services), some of whom may be accessing the systems from outside of the region, which also means sensitive data accessed there could be classed a data transfer.

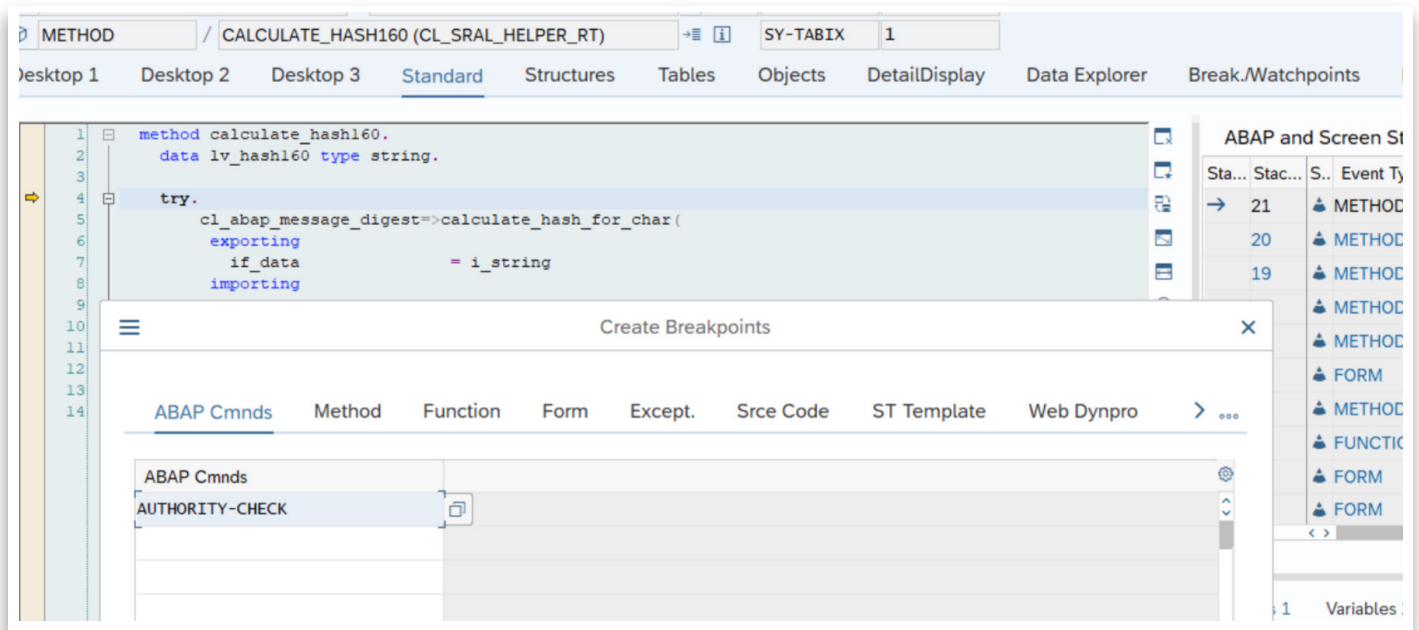
On the few occasions where someone may have to have developer access in production to resolve a critical issue, make sure you then remove the access as soon as they have investigated the issue. Or alternatively look at an 'Elevated Rights Management' solution like the one provided by **Soterion**, or the 'Firefighter' functionality of GRC. This would allow the person to have the required access, but with increased logging of the activities they carry out.

Hack 2: Stepping around an authority check or changing the result of it

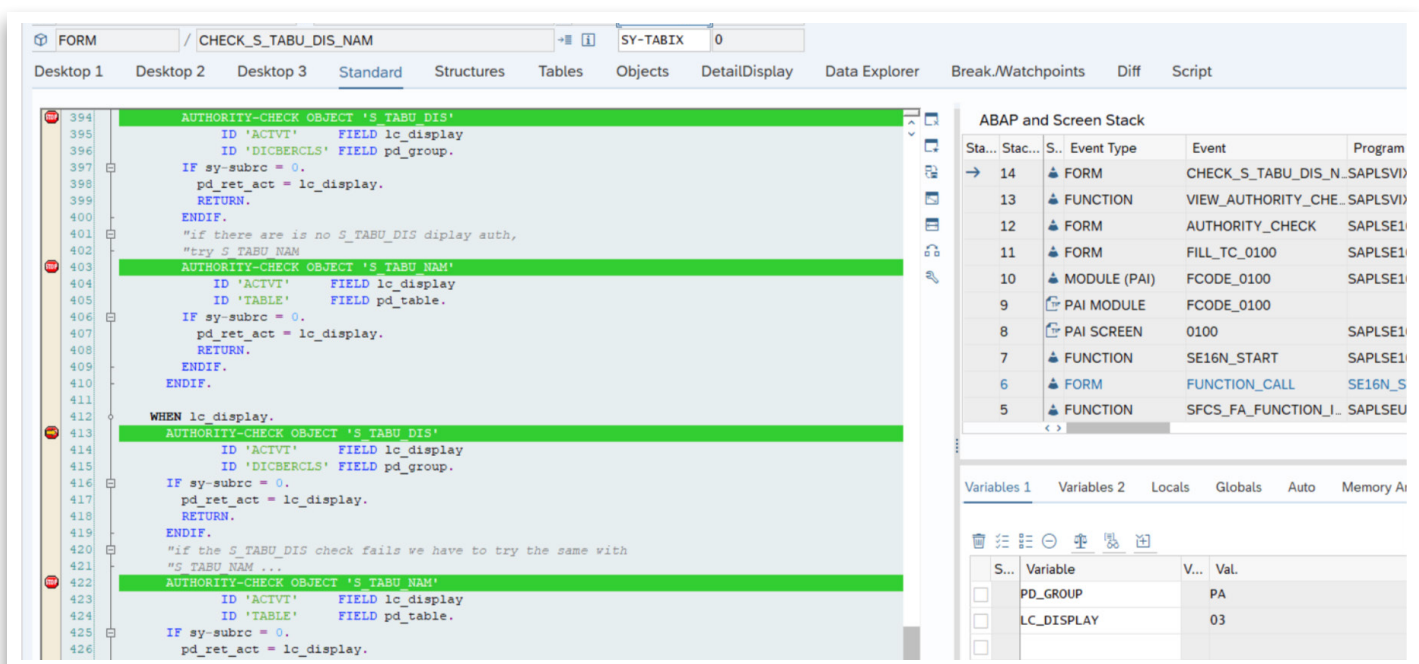
For the second hack, let's continue from where we left off with our unauthorized user. We could access table KNA1 via the SE16N function module, but we could not see the employee data we were really after. As well as being able to test function modules, we have some debug capability.



As soon as we enter the debugger we can put a breakpoint at a statement and enter the AUTHORITY-CHECK statement:



This now stops the code at each authorization check:



The check about to happen is for the table group PA which we don't have authorization for. Our S_TABU_DIS access is limited to group VA which is why we could see KNA1.

And we can now use the 'Go to statement' option to step over the authority check for group PA so it is not called. Or, we can let the authority check happen and then simply change the return code afterwards.

The screenshot shows two parts of the SAP development environment. On the left is the ABAP editor with lines 409 to 428. The code includes an 'AUTHORITY-CHECK OBJECT' for 'S_TABU_DIS' and a conditional check for 'SY-SUBRC = 0'. On the right is the 'Variable Monitor' window, showing the current state of variables: PD_GROUP is 'PA', LC_DISPLAY is '03', and SY-SUBRC is '4'.

S...	Variable	V...	Val.	C...	Hexadecimal
	PD_GROUP		PA		50004100
	LC_DISPLAY		03		30003300
	SY-SUBRC		4		04000000

This screenshot shows the same ABAP code as before, but the variable monitor on the right has been updated. The PD_GROUP variable now contains 'PA', LC_DISPLAY is '03', and SY-SUBRC is '0', indicating that the authority check was bypassed or successfully completed.

S...	Variable	V...	Val.	C...	Hexadecimal Value	Technical
	PD_GROUP		PA		50004100200020002000	C(14)
	LC_DISPLAY		03		30003300	C(2)
	SY-SUBRC		0		00000000	I(4)

And now the code continues as if we were authorized:

The screenshot displays the SAP HR Master Record search results for table PA0002. The search criteria include 'Table to be searched: PA0002', 'Number of hits: 500', and 'Runtime: 0'. The results table shows a list of 101 records, including personal data, dates, and system fields.

PersNo	Sty	ObjID	LI	End Date	Start Date	RNo	Changed on	Changed by	H	Tx	Rf	Co	SC	Re	Reserved Field/Unused Field	Reserved Field/Unused Field	Reserved Field/Unused Field	Reserved Field/Unused Field	Reserve
1				04.10.2006	10.10.1960		25.01.1996	WECKESSER											
1				31.12.9999	05.10.2006		08.10.2010	WEISSANJA											
10				31.12.9999	22.05.1967		07.05.2003	WEISSANJA											
69				31.12.9999	01.01.1956		17.09.2003	HOLDERM											
70				31.12.9999	01.01.1921		17.09.2003	HOLDERM											
71				31.12.9999	09.09.1967		30.09.2003	HOLDERM											
72				31.12.9999	08.09.1965		30.09.2003	HOLDERM											
73				31.12.9999	09.09.1956		30.09.2003	HOLDERM											
100				31.12.9999	02.07.1971		18.11.2013	I300511											
101				31.12.9999	09.12.1969		03.10.2013	I300511											

And we have access to names, dates of births, government ID numbers etc.

Mitigations

If someone carries out this activity, you would see the failed authorizations if you were running an authorization trace from ST01, or if the SM20 audit logging is enabled you would see it there. BUT the best place to monitor for this sort of activity is the system log SM21:

07.07.2020	11:17:22	ed1_ED1_00	DIA	009	800	NOAUTHS	▲	A23	Goto ABAP Debugger: Source:(413)->(416) ByteCode:AUTH(
07.07.2020	11:17:22	ed1_ED1_00	DIA	009	800	NOAUTHS	○	A14	> in program LSVIXU16 , line 0416, event CHECK_S_TABU_DIS_N
07.07.2020	11:19:43	ed1_ED1_00	DIA	009	800	NOAUTHS	●	A19	Field contents changed: SY-SUBRC -> 0
07.07.2020	11:19:43	ed1_ED1_00	DIA	009	800	NOAUTHS	○	A14	> in program LSVIXU16 , line 0416, event CHECK_S_TABU_DIS_N

The first entry at 11:17 is my user stepping over the authorization check, and you can see from the event that this is an authorization check I’ve avoided.

The second entry at 11:19 is where we let the check happen, but then changed the return code. In a development system there may be very valid reasons why someone testing code will change a return code to go into a different branch of code. But again, look at the event in the line underneath; that tells us this related to an authorization check.

Automated monitoring of the system logs can also track for these messages. This is a great example of where connecting **SAP to Splunk** can be of benefit.

But that is of course finding out after the fact. The main mitigation points here are the same as Hack 1; don’t give developer access in production except in exceptional cases. And even then, be careful with S_DEVELOP as this can allow the stepping from one line of code to another or changing variables. Better still; take a look at an elevated rights management solution.

And again, don’t keep real sensitive data in non-production systems otherwise the same mitigations apply there.

Hack 3: Accessing sensitive data by writing code

The final example is something which has driven some of the landscape management recommendations I’ve given to clients over the years, and is solely focused on development and sandbox systems. When someone has the ability to write code, you understand that with just a little knowledge of the SAP data model, they can access sensitive data from any table and present it on the screen or in a downloadable form. They are responsible for the authorization checks their code invokes. But most organizations have very little data in their development client(s), so where’s the risk? Answer: in another client on the same system.

In this example, I am on client 800 which is our development client. But we have used client 900 to test with real data. Now as the hacker, I am particularly interested in a specific employee for which I know the number but have no access to view data in any systems. But I overheard that they were using client 900 for payroll testing and no-one except the payroll team had access there.

In client 800:

Display HR Master Data

Cancel

More

Personnel no.

10004006

Core Employee Info.

Empl. contract data

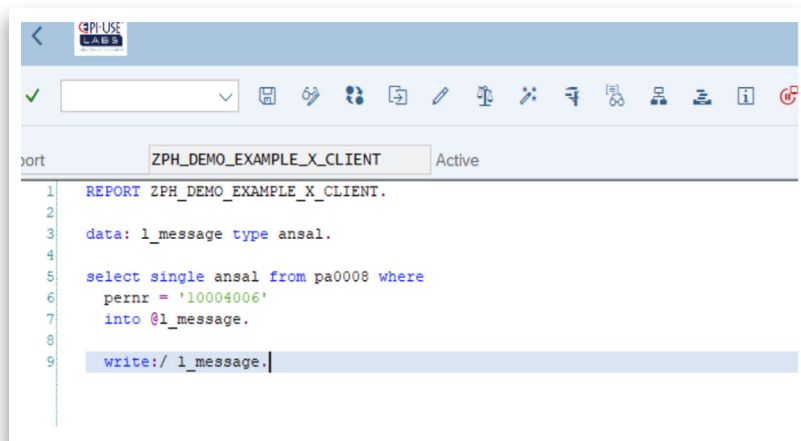
Gross/net payroll

Net payroll

> ...

No data for this employee.

So I decide to write some code:



```
1 REPORT ZPH_DEMO_EXAMPLE_X_CLIENT.
2
3 data: l_message type ansal.
4
5 select single ansal from pa0008 where
6   pernr = '10004006'
7   into @l_message.
8
9 write:/ l_message.
```

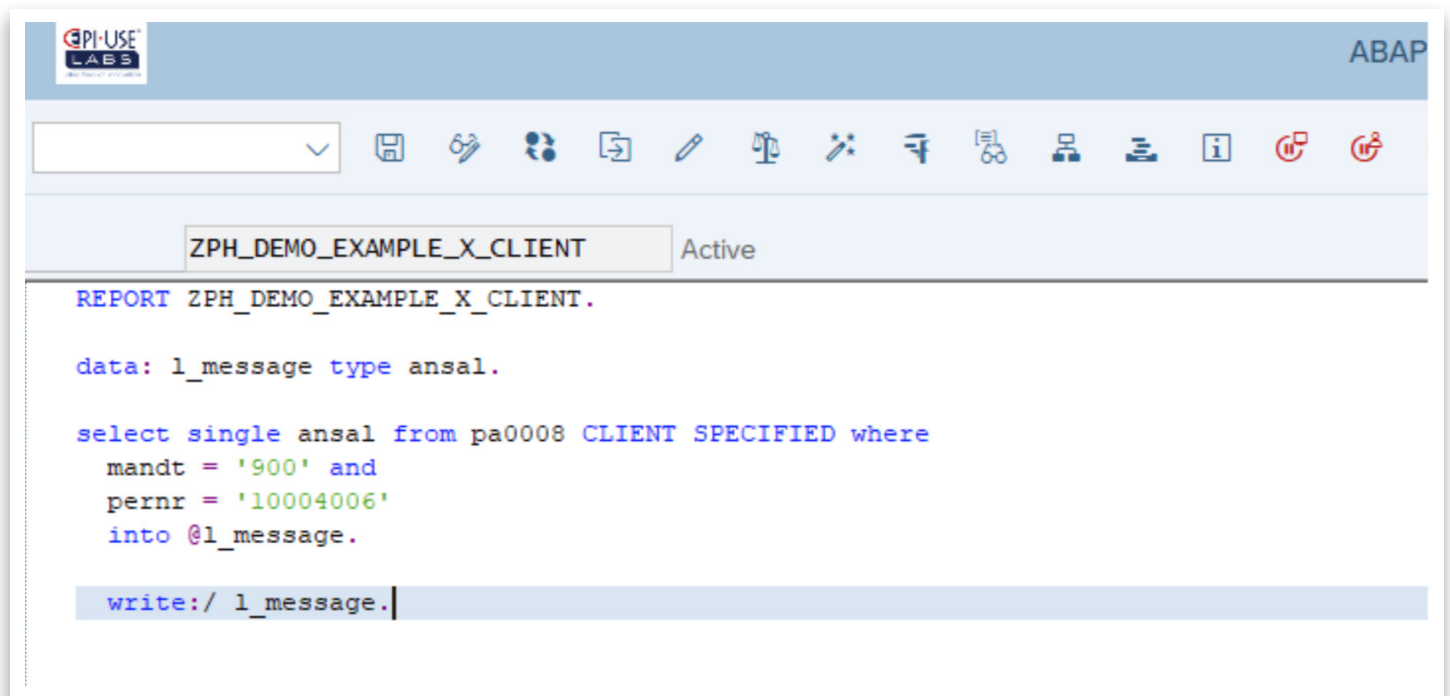
When the program has a select statement like this, the ABAP interpreter that converts this into an SQL statement for the database actually adds in the field 'MANDT' (or in some cases 'CLIENT' or 'MANDANT') which is the field on the client dependent table which tells you which client the data is for. So this statement above would automatically be converted to SQL similar to:

```
select single ansal from pa0008 where
mandt = sy-mandt and
pernr = '10004006'
```

SY-MANDT being the environment variable that is the client the user is logged on to when the code runs. This happens to ensure the client segregation of SAP systems, but there is an addition you can use: CLIENT SPECIFIED, which deactivates the automatic client handling.

This is true for deletions, inserts etc as well.

So by changing the code to:



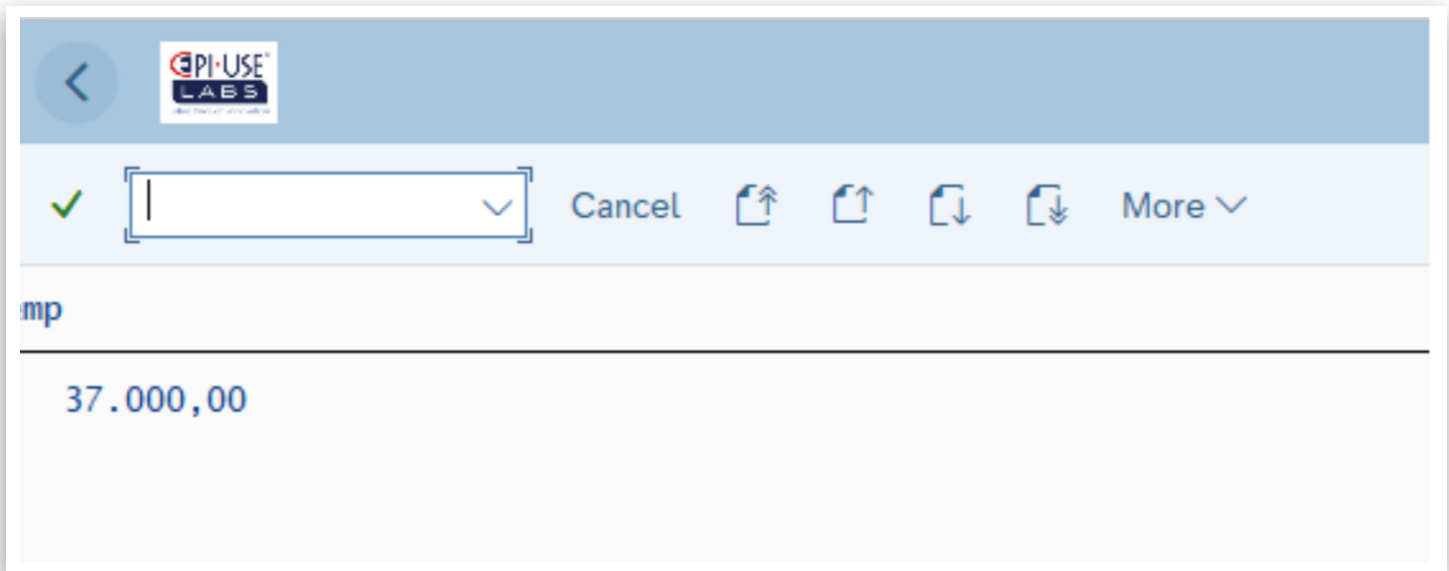
```
REPORT ZPH_DEMO_EXAMPLE_X_CLIENT.

data: l_message type ansal.

select single ansal from pa0008 CLIENT SPECIFIED where
  mandt = '900' and
  pernr = '10004006'
  into @l_message.

write:/ l_message.
```

Which then means the OpenSQL created will not try to add SY-MANDT to the where clause itself, but expects the code to provide the value. And the result....



As a user with no access to client 900, I have just gained access to the annual salary (ansal) from infotype 8 of an employee that only exists there.

Mitigations

I have worked with some organizations that only provide a developer key for a fixed period, and then it has to be renewed; but I would say it makes sense to consider who is given developer keys, and ensure it is limited to those people only.

The most important point here is that sensitive data in any client of a system that allows repository changes is a very dangerous thing to allow. If someone from Payroll does need to test in a development client, then remove the data straight afterwards.

EPI-USE Labs' Object Sync™, which you might have used to move the data there, has a deletion utility too; and there is a SAP standard deletion program for employee data. The Object Sync deletion utility can even be scheduled periodically to delete all employee data in a client. Otherwise, ensure the data is masked, potentially even salary values, keeping in mind that does negate the data for accurate Payroll testing. Ideally though, just find a client on a QA or other system which does not allow repository changes, and use that for any real data testing that needs to be carried out.

And still mask the data that can be masked without affecting the tests.

Paul Hammersley
paul@labs.epiuse.com



epiuselabs.com



Facebook



LinkedIn



Our software, your advantage

