Here you have a figure with the distribution of reported security vulnerabilities, as described in the SAP Security Notes, affecting SAP's own ABAP code (the proprietary code implementing many SAP functions).

These results are aligned with what we said before, almost ¼ of the vulnerabilities are related to access control (missing, incorrect or badly implemented authorization). The second in line is cross-side scripting (XSS), relevant but not of much impact in this context as we will see later; path traversal is third, very easy to introduce but hardly known by the ABAP developer community; and then other classics like SQL injection, cross-side request forgery (CSRF) and code injection.

I think it is safe to extrapolate these results to custom ABAP code out there, so now you know what you are up against. To understand why these, and not others are the usual suspects we have to consider when coding ABAP applications, let’s take a closer look at the SAP attack surface.

Source for upper graphic: ERPScan Research, "Analysis of 3000 vulnerabilities in SAP" (2014)
In modern SAP implementations you can find other services like Fiori (richer user experience), Afaria (mobile device management), Business Objects, Mobile Platform, Hana (architecture beyond R/3), Business Warehouse or Lumira (data visualization). The flexibility (hence complexity) of a SAP system implies that there are many weak points that pose a threat to possible attacks. All these points put together are considered the attack surface.

For starters, you have the user entry points, GUI screens, web pages and mobile apps used to interact with the system. If you are serious about application security you should know that user input should always be treated as an “untrusted” source and your applications have to make sure the input can only be trusted depending on what the final destination of that input is going to be. Again it is the developer’s responsibility to check all user input.

Then you have protocols and APIs. The protocol for remote execution of ABAP programs and functions, RFC; web protocols HTTP/S, the protocols to communicate with databases, xDBC and SQL/MDX in Hana, if not used correctly can pose a serious threat to security. SAP exposes a good number of APIs and not all are always secure (ABAP programmers can even make direct calls to the SAP kernel) the aforementioned OpenSQL for example could be highly insecure if not used properly.

Another concern that adds up to the attack surface is the fact that SAP stores application data, configuration and system info and even the ABAP code itself in the same database. This opens the door (via SQL injection) to attacks that can install harmful code, change the security configuration to gain special privileges and, considering that SAP processes usually run with high OS privileges, attackers could potentially take full control of your system. No joke.
Missing authorization checks

You can argue if it was a good decision or not, but it is what it is. SAP authorization model is explicit, so the responsibility of checking authorization to access resources is in the developer's hands. They have to do AUTHORITY-CHECK on authorization objects associated to roles before using those resources. There, language provides functions for commonly used resources/objects:

- AUTHORITY_CHECK_TCODE
- AUTHORITY_CHECK_DATASET
- CALL TRANSACTION ... WITH AUTHORITY_CHECK...

What resources should you consider sensitive? The obvious answer is all, but if you have to choose, go for: transactions, RFC functions or programs, system commands, files, SAP system tables and user tables with sensitive data (you should know which ones these are).

Even if there is an explicit check it could be weak or even useless if the sy-subrc return code is not treated appropriately or if DUMMY fields or '*' are used instead of explicit values for the protected resource.

Want to see what this looks like?

- Call to sensitive transaction without authorization check

```plaintext
1 " VIOLATION: No explicit authorization check
2 CALL TRANSACTION 'S538' USING BODATA MODE 'N' MESSAGES INTO NESSTAB.
3 " Starts ABAP editor, where attacker may inject or alter code in SAP system
```