The ABAP Underverse

Risky ABAP to Kernel Communication & ABAP-tunneled buffer overflows

Andreas Wiegenstein

Virtual Forge GmbH

andreas.wiegenstein@virtualforge.com

"Hailing frequencies open."

Virtual Forge GmbH

- SAP security product company based in Heidelberg, Germany
- Focus on (ABAP) application security services
  - ABAP Security Scanner
  - ABAP Security Guidelines
  - ABAP Security Trainings
  - SAP Security Consulting

Andreas Wiegenstein

- CTO and founder of Virtual Forge, responsible for R&D
- SAP Security Researcher, active since 2003
- Speaker at SAP TechEd 2004, 2005, 2006, DSAG 2009, …
- Co-Author of "Secure ABAP Programming" (SAP Press)

Belief: „Our SAP system is secure.“

- Roles & Authorizations
- Segregation of Duties
- Secure Configuration & System / Service Hardening
- Encryption
- Secure Network Infrastructure
- Password Policies
- Patch Management
- Identity Management
- Single Sign-on

SUPER-GREEN!

Reality-Check

"...and this is our ABAP security department."

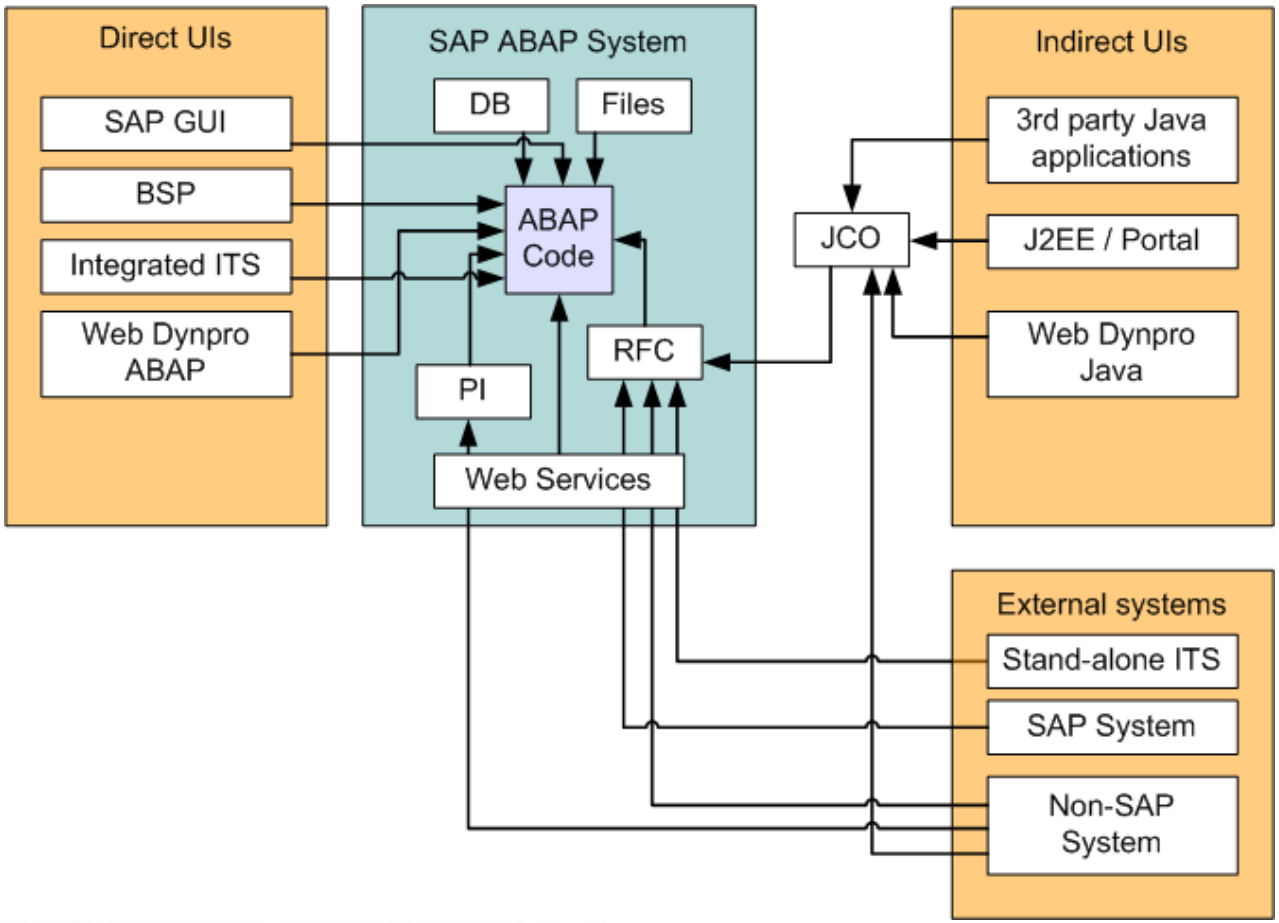
Advanced **B**usiness **A**pplication **P**rogramming

Fundamentals

- Proprietary language, exact specification not (freely) available
- Platform-independent code
- Client separation built-in
- Integrated auditing capabilities
- System-to-System calls via SAP RFC standard
- Built-in transport system and version control
- Various programming paradigms:
  - Programs & Forms, Reports, Function Modules, Dynpros
  - Classes & Methods, Business Server Pages, Web Dynpro ABAP

- Integrated platform-independent SQL Standard: Open SQL
- Built-in authentication, roles and (explicit) authorization model
- Thousands of well-known standard programs and database tables
- 150+ Million Lines of Code in an ECC6.0 System

ABAP's Attack Surface

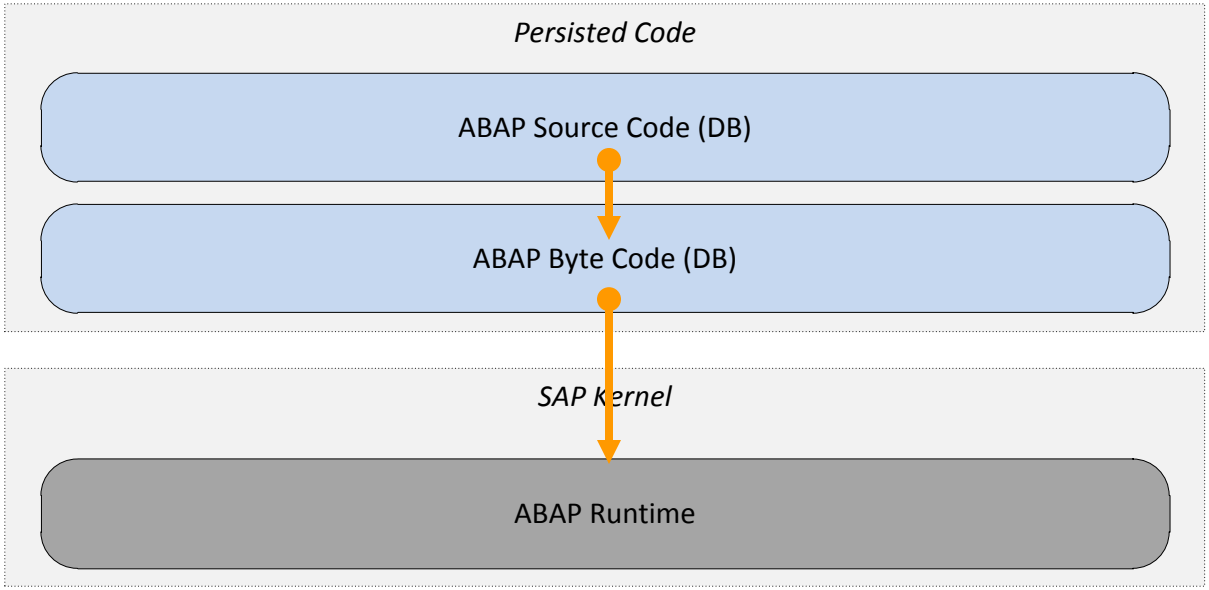


ABAP Security Risks

- Back Doors
  - Intentional bugs, e.g. malicious developer or spy
- Side-effects
  - Unintentional bugs, e.g. due to lack of knowledge
- Insecure alternatives to SAP standard security mechanisms

The ABAP Runtime

(theoretical behavior)



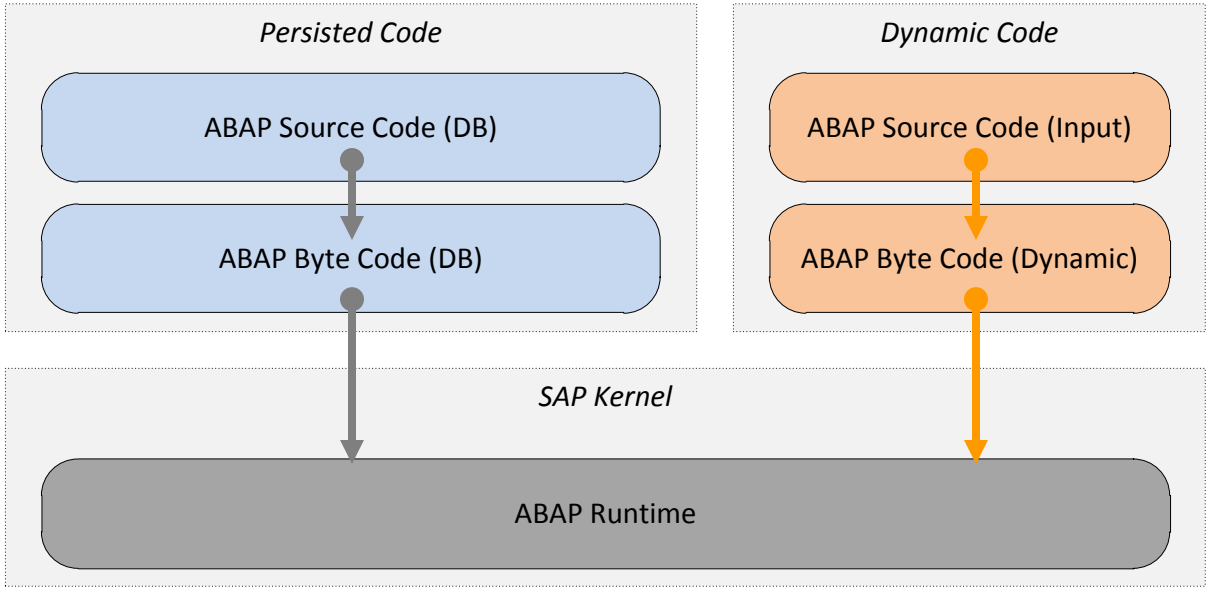
Static ABAP examples

```
REPORT ZSTATIC_ASSIGN.
DATA lv_secret TYPE string.
FIELD-SYMBOLS <fs> TYPE ANY.
PARAMETERS lv_tmp TYPE string DEFAULT 'Bishop'.

ASSIGN lv_tmp TO <fs>.
```

```
REPORT ZSTATIC_DELETE.
DELETE FROM usr02 WHERE bname = sy-uname.
```

The ABAP Runtime
(dynamic ABAP commands)



Dynamic ABAP example #1

```
REPORT ZDYNAMIC_ASSIGN.
DATA lv_secret TYPE string VALUE 'ZFT'.
DATA lv_share TYPE string VALUE 'Bishop'.
FIELD-SYMBOLS <fs> TYPE ANY.
PARAMETERS lv_tmp TYPE string DEFAULT 'lv_share'.

ASSIGN (lv_tmp) TO <fs>.
```

Hack #1: **lv_secret**

Hack #2: **sy-opsys**

GENERIC VARIABLE READER

Dynamic ABAP example #2

```
REPORT ZDYNAMIC_DELETE.
PARAMETERS lv_tmp TYPE string
                  DEFAULT 'bname = sy-uname'.

DELETE FROM usr02 WHERE (lv_tmp).
```

Bad Hack: `1 = 1`

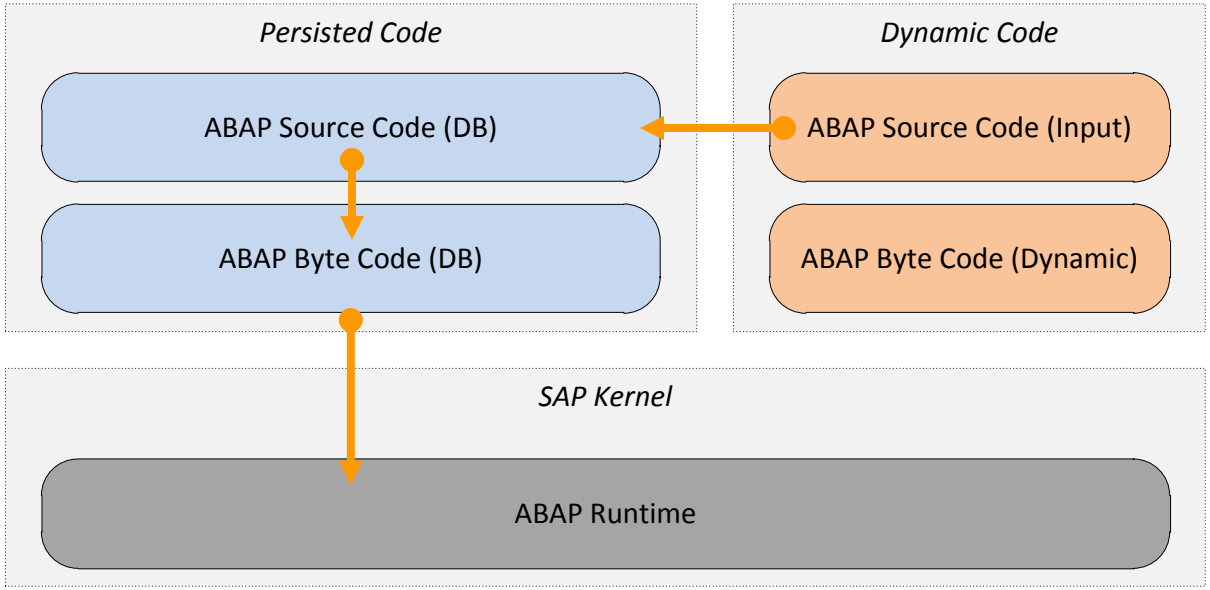
Better Hack: `mandt = sy-mandt`

Good Hack:

OPEN SQL INJECTION

The ABAP Runtime

(persisted code generation)



Persisted code generation

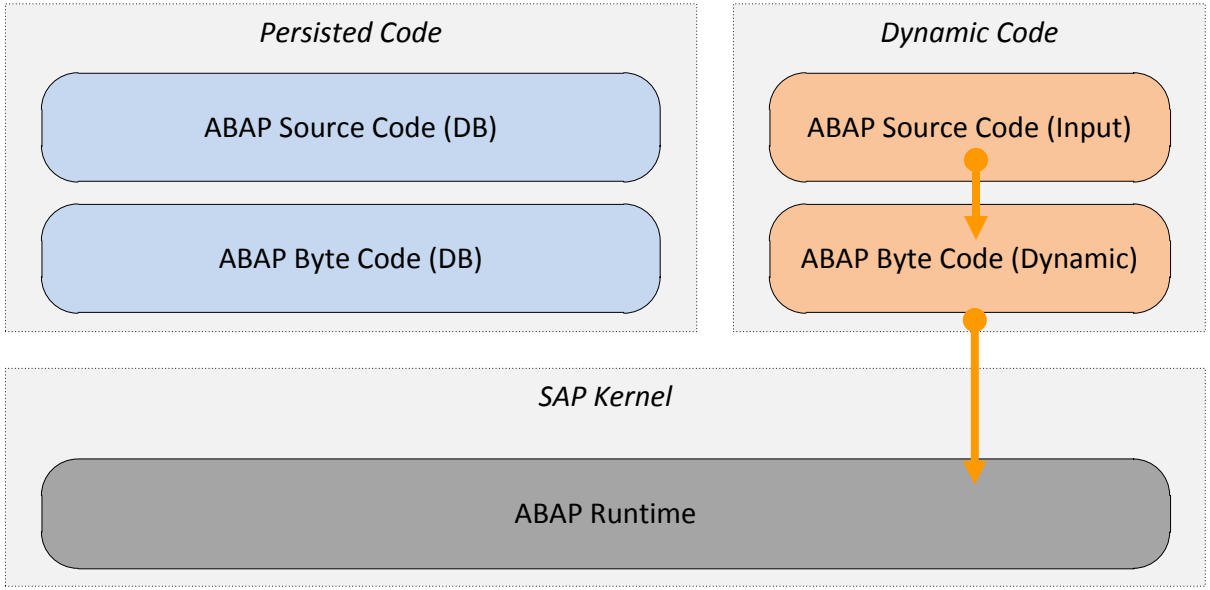
```
REPORT ZPERSISTED_ABAP.
DATA: lt_prog(97) OCCURS 0 WITH HEADER LINE.
PARAMETERS lv_name TYPE string.

lt_prog = 'REPORT ZFT.'.      APPEND lt_prog.
CONCATENATE `DATA lv_tmp(82) TYPE c VALUE '`
  lv_name `'.` INTO lt_prog. APPEND lt_prog.
lt_prog = 'WRITE / lv_tmp.'. APPEND lt_prog.

INSERT REPORT 'ZFT' FROM lt_prog.
GENERATE REPORT 'ZFT'.
SUBMIT ('ZFT').
```

ABAP CMD INJECTION

The ABAP Runtime
(dynamic code generation)



Dynamic code generation

```
REPORT ZDYNAMIC_ABAP.
DATA: prog(72) OCCURS 0 WITH HEADER LINE.
DATA: pool(20) TYPE c.
PARAMETERS lv_name TYPE string.
prog = 'REPORT MASSIVEDYNAMIC.'.      APPEND prog.
prog = 'FORM BELL.'.                  APPEND prog.
prog = 'DATA name TYPE string.'.      APPEND prog.
CONCATENATE `name = '` lv_name `'.  `
  INTO prog.                          APPEND prog.
prog = 'WRITE : / ''Hello '', name.'. APPEND prog.
prog = 'ENDFORM.'.                    APPEND prog.
GENERATE SUBROUTINE POOL prog NAME pool.
PERFORM ('BELL') IN PROGRAM (pool).
```

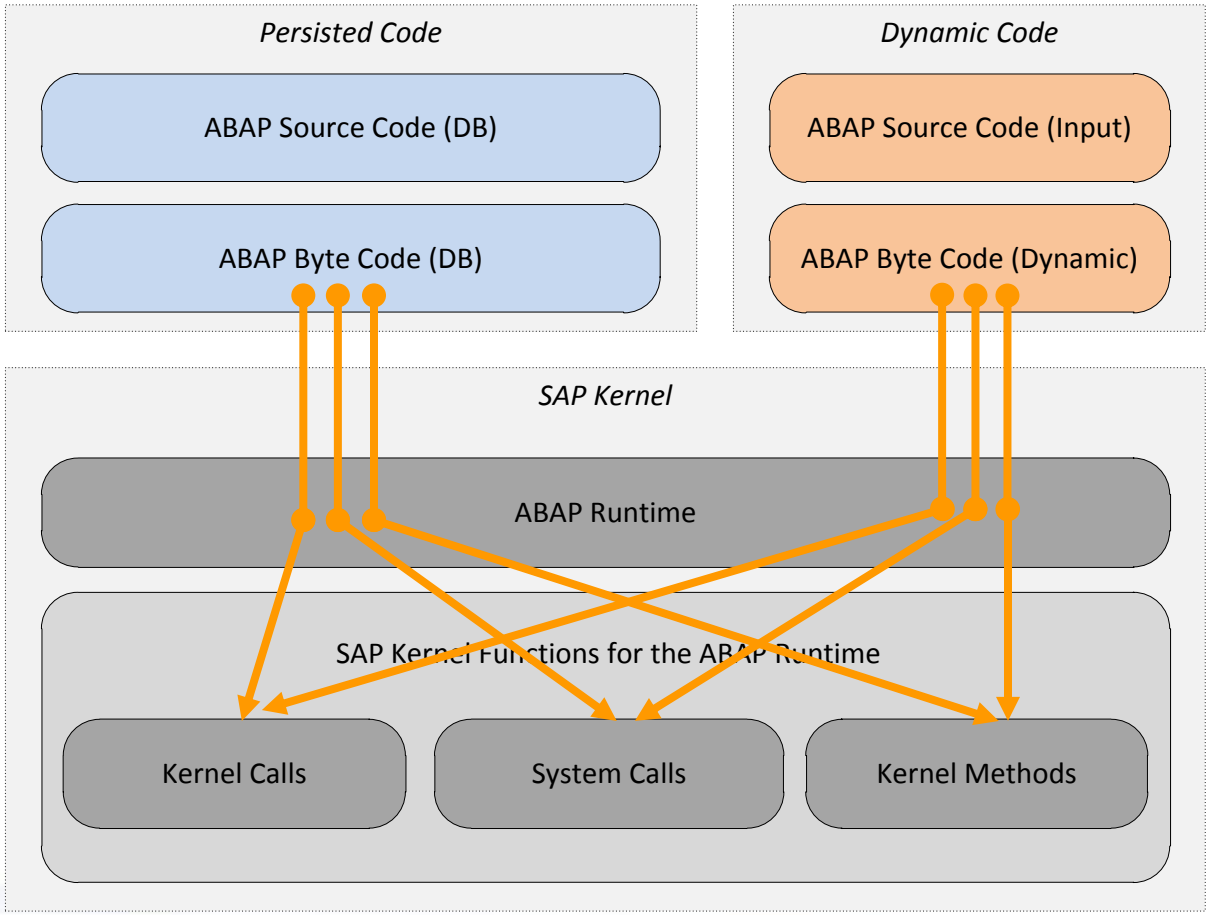
ABAP CMD INJECTION

ABAP code generation risks

- ABAP developer right = "root" access
- Silent way to acquire "SAP ALL" privileges takes 56 characters.
- Noisy way to acquire "SAP ALL" privileges takes 34 characters.
- *Entire SAP system must be considered compromised in case of an ABAP Command Injection vulnerability*

The ABAP Runtime

(Kernel Communication)



Kernel Communication

- High-performant pool of helper functions
- Should only be used by SAP
- Can be explicitly invoked from ABAP
- Mostly undocumented functionality
- Three types of Kernel functions exist:
  - Kernel Calls
  - System Calls
  - Kernel Methods

Kernel Calls

```
REPORT ZKERNEL_CALL.
DATA lv_dbhost LIKE msxxlist-host.
CALL 'C_SAPGPARAM' ID 'NAME' FIELD 'SAPDBHOST'
                   ID 'VALUE' FIELD lv_dbhost.
```

- 370+ different Kernel Calls in ECC 6.0
- In general, no implicit authority check
- Importing / Exporting parameter not obvious

System Calls

```
REPORT ZSYSTEM_CALL.
DATA: lv_nam, lv_val TYPE string.
DATA: m_err          TYPE i.
DATA: m_c_msg        TYPE %_c_pointer.

lv_name = 'chevron_V'.
SYSTEM-CALL ICT DID 11
  PARAMETERS m_c_msg lv_nam lv_val m_last_err.
```

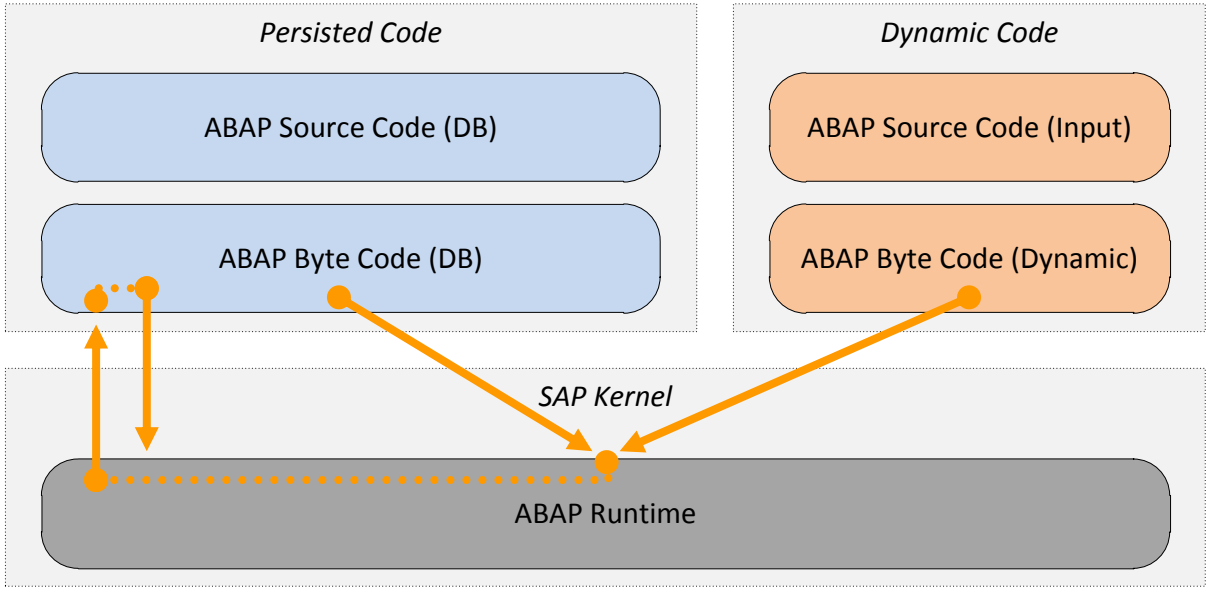
- Polymorph interface

Kernel Methods

```
METHOD convert BY KERNEL MODULE ab_expconvDbuf.
ENDMETHOD.
```

- Can't be declared by custom code
- Are called like normal methods
- Have typed parameters

The ABAP Runtime
(Kernel Hooks)



Kernel Hooks

- Kernel Hooks are ABAP subroutines (FORMs) called from the ABAP C/C++ Runtime.
- Data exchange works via Kernel Calls
- Multiple ABAP commands use Kernel Hooks
  - OPEN DATASET
  - CONVERT TIME-STAMP
  - WRITE AS TIME-STAMP
  - …

Kernel Hooks example (part 1)

```
FORM SYSTEM_HOOK_ZFT.

FIELD-SYMBOLS:  <P1> type any, <P2> type any,
                <P3> type any, <P4> type any,
                <R1> TYPE I, <R2> LIKE SY-SUBRC.

* First, pop the Kernel parameters from the stack
CALL 'AB_GET_C_PARMS' ID 'P1' field <P1>
                      ID 'P2' field <P2>
                      ID 'P3' field <P3>
                      ID 'P4' field <P4>.
```

Kernel Hooks example (part 2)

```
* Next, call a function to process the data

CALL FUNCTION 'ZPROCESS_THIS_STUFF'
  EXPORTING date  = <P1>
            time  = <P2>
            hour  = <P3>
            mint  = <P4>
  IMPORTING res   = <R1>
            subrc = <R2>
  EXCEPTIONS NO_PERMISSION.
```

Kernel Hooks example (part 3)

```
* Finally, push the results onto the Kernel stack
and exit the hook

CALL 'AB_SET_C_PARMS' ID 'P1' FIELD <R1>
                      ID 'P2' FIELD <R2>.

ENDFORM.
```

Kernel Hooks risks

- Malicious code can attach itself to the execution of ABAP commands that use Kernel Hooks

Whoever experiments with this: don't call the ABAP command you're hooked in…

Risky Kernel Calls (8 out of 370+)

Kernel Call	Risk
'SYSTEM'	OS command execution
'XXPASS' & 'XXPASSNET'	Compute password hash
'INTERNET_USER_LOGON'	Brute force credentials
'C_GET_TABLE'	Unauthorized table read access
'C_MOD_TABLE'	Unauthorized table write access
'C_DB_EXECUTE'	SQL Injection; DB modification
'C_DB_FUNCTION'	SQL Injection; DB modification

CALL 'SYSTEM'

```
CALL 'SYSTEM' ID 'COMMAND' FIELD lv_cmd
              ID 'TAB' FIELD lt_result-*sys*.
```

- ID 'COMMAND' defines OS command
- ID 'TAB' receives output (optional)
- *Implicit* auth check in Kernel (S_C_FUNCT)
- Can be disabled by setting profile parameter ‘rdisp/call_system’ to ‘0’
- *Not the only way to execute OS commands!*

CALL 'XXPASS' / 'XXPASSNET'

```
CALL 'XXPASS' ID 'CODE'     FIELD 'akagi'
              ID 'NAME'     FIELD 'castle'
              ID 'CODX'     FIELD lv_hash
              ID 'PASSCODE' FIELD lv_pass
              ID 'VERS'     FIELD lv_vers.
```

- ID 'CODX' returns the hash value
- Terminates session and locks user account
- Defensive reaction depends on name(space) of the calling program (Y*, Z* problematic)

CALL 'INTERNET_USER_LOGON'

```
CALL 'INTERNET_USER_LOGON'
  ID 'UNAME'     FIELD lv_userid
  ID 'PASSW'     FIELD lv_password
  ID 'TICKET'    FIELD lv_ticket
  ID 'PASSFLAG' FIELD lv_pwdstate.
```

- Performs user switch & creates logon ticket
- Can check credentials only (TESTMODE)
- Keeps track of failed login attempts

CALL 'C_GET_TABLE'

```
CALL 'C_GET_TABLE'
   ID 'TABLNAME'   FIELD lv_tab
   ID 'INTTAB'     FIELD lt_tab-*sys*
   ID 'GENKEY'     FIELD lv_key
   ID 'GENKEY_LN' FIELD lv_keylen
   ID 'BYPASS'     FIELD lv_bypass.
```

- Low-level direct access to database tables
- Cross-Client data access

CALL 'C_MOD_TABLE'

```
CALL 'C_MOD_TABLE'
   ID 'TABLNAME' FIELD lv_tab
   ID 'INTTAB'   FIELD lt_tab-*sys*
   ID 'FCODE'    FIELD 'U'
   ID 'DBCNT'    FIELD lv_dbcnt
   ID 'SQLCODE'  FIELD lv_sqlcode.
```

- Low-level direct access to database tables
  - U(pdate), I(nsert) or D(elete) table rows
- Cross-Client data access

CALL 'C_DB_EXECUTE'

```
CALL 'C_DB_EXECUTE' ID 'STATLEN' FIELD lv_len
                    ID 'STATTXT' FIELD lv_stmt
                    ID 'SQLERR'  FIELD lv_sqlerr.
```

- Executes an *arbitrary* SQL command
  - (Except SELECT)
- Cross-Client data access
- Can gain full control over SAP database

CALL 'C_DB_FUNCTION' #1

```
CALL 'C_DB_FUNCTION' ID 'FUNCTION' FIELD 'DB_SQL'
                     ID 'FCODE'    FIELD 'EP'
                     ID 'PROCNAME' FIELD lv_proc
                     ...
```

- Executes an *arbitrary* stored procedure
- Can gain full control over SAP database

CALL 'C_DB_FUNCTION' #2

```
CALL 'C_DB_FUNCTION' ID 'FUNCTION' FIELD 'DB_SQL'
                     ID 'FCODE'    FIELD 'PO'
                     ID 'STMT_STR' FIELD lv_stat
                     ...
```

- Executes an *arbitrary* SQL statement
- Cross-Client data access
- Can gain full control over SAP database

Risky Kernel Calls

DEMO

Risky Kernel Calls Summary

- No (comprehensive) documentation
  - Unknown number of critical Kernel Calls
- Several known calls can bypass SAP security mechanisms
- Several 0 days wait for "responsible disclosure"
- *Kernel Calls must be addressed in ABAP security guidelines, development standards & security assessments: Don't use Kernel Calls*

Buffer overflows in Kernel Calls

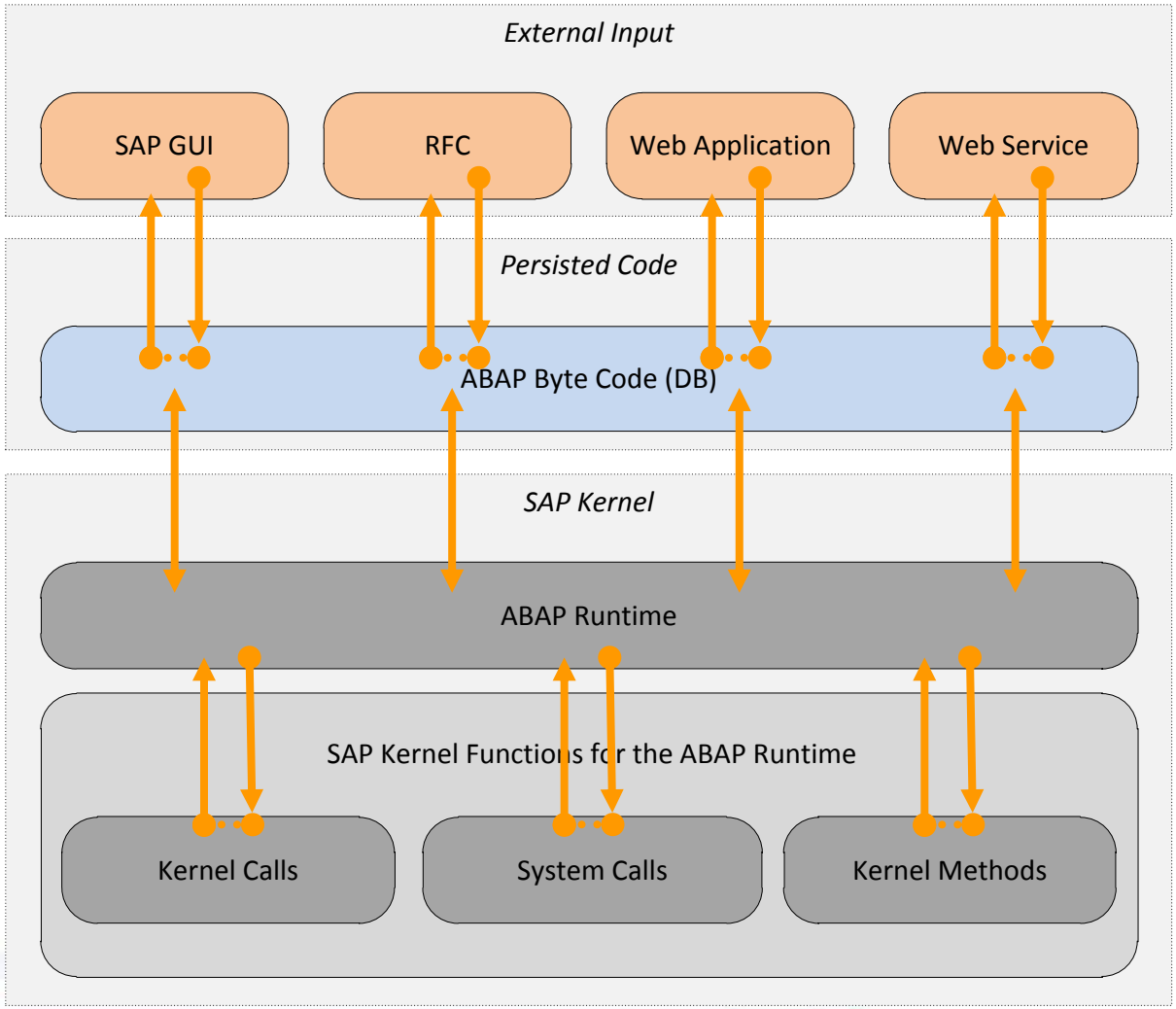
- Kernel Calls can't be accessed remotely
- ABAP has automatic memory management – bad news for "buffer overflow exploiters"

But...

- (Custom) ABAP programs can expose Kernel Calls to external input

➢ Attackers must find a data flow path to a vulnerable parameter of a Kernel Call

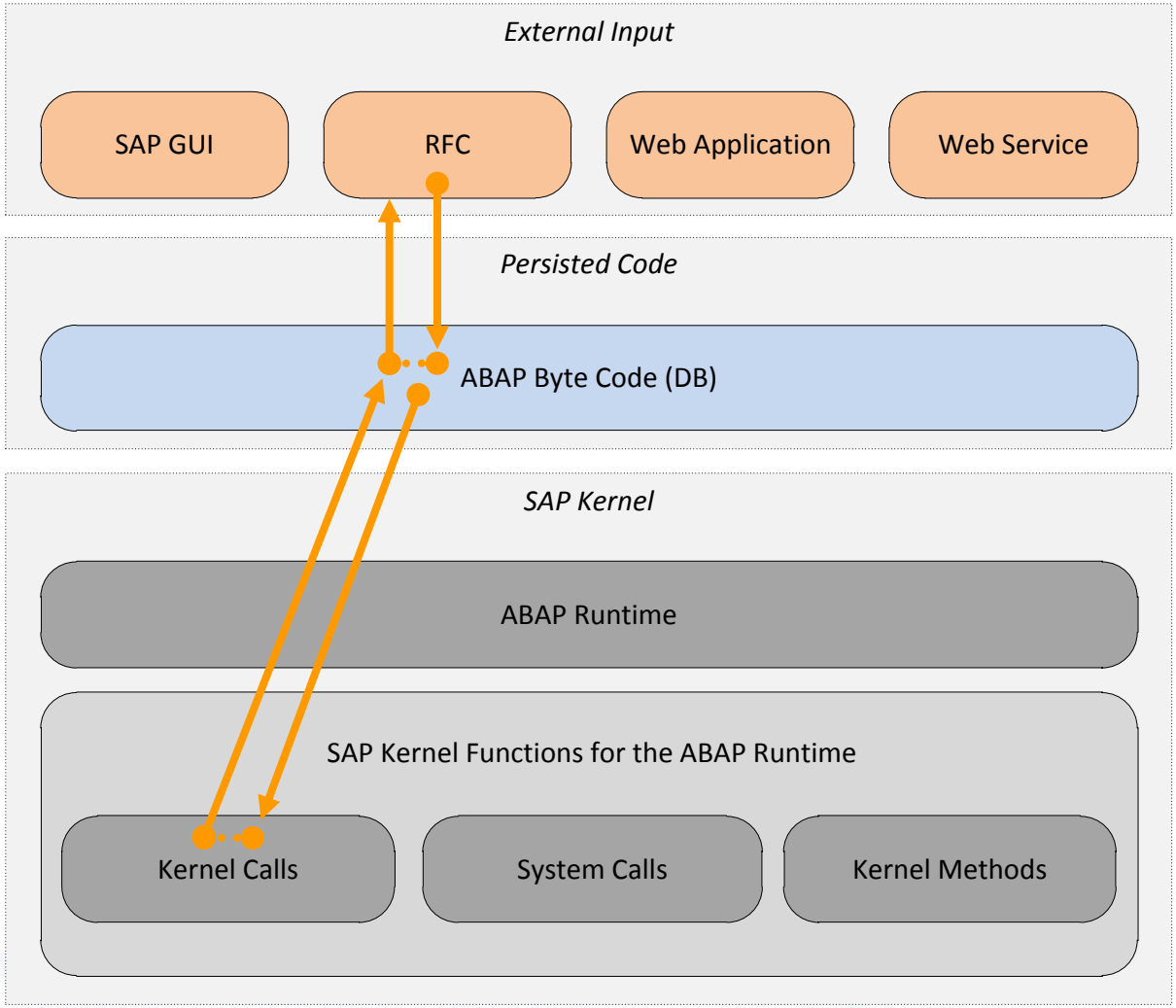
ABAP Runtime shields Kernel Calls



Buffer overflow risks

- An attacker may find a way to pass input to a vulnerable Kernel or System Call
- Since Kernel / Systems Calls can only be executed by ABAP commands, an exploit needs to be tunneled through the ABAP code
- The risk increases, if the "ABAP tunnel" can be invoked remotely, e.g. via RFC or a Web page.

ABAP-tunneled Buffer Overflows



Buffer overflows in Kernel Calls (1)

```
CALL 'C_SAPGDEFPARAM' ID 'NAME'  FIELD lv_buffy
                      ID 'VALUE' FIELD lv_dummy.
```

- Buffer overflow* in **NAME** with > 103 chars
- Exposed by RFC-enabled function module
- But data-type is a TEXT60 only.

Buffer overflows in Kernel Calls (2)

```
CALL 'BUILD_DS_SPEC' ID 'FILENAME' FIELD lv_buffy
                     ID 'PATH'     FIELD lv_buffy
                     ID 'OPSYS'    FIELD lv_buffy
                     ID 'RESULT'   FIELD lv_na.
```

- Buffer overflow* in **NAME** with > 404 chars
- Buffer overflow* in **PATH** with > 204 chars
- Buffer overflow* in **OPSYS** with > 428 chars
- Exposed by RFC-enabled function module
- But data-type is a TEXT80 only.

Buffer overflow in System Calls

```
SYSTEM-CALL ICT DID 29
   PARAMETERS
      lv_buff1 lv_buff2 lv_dummy lv_dummy
      lv_dummy lv_dummy lv_dummy lv_dummy.
```

- Buffer overflow* in parameter 1 & 2 with > 200 characters
- Not exposed

Exposed Buffer overflow in a Kernel Call

```
CALL 'C_SAPGPARAM' ID 'NAME'  FIELD lv_buffy
                   ID 'VALUE' FIELD lv_dummy.
```

- Buffer overflow* in **NAME** with > 108 chars
- Exposed by RFC-enabled function module
- Data-type is a TEXT255 ☺

Buffer Overflow PoC

DEMO

Buffer overflow protection steps

1) Pick a Kernel / System Call and identify its importing parameters
2) Send overlong input and check if the ABAP work-thread dies
3) Check where this Kernel / System Call is used (in the SAP standard) and which data types are used in the data flow path
4) Test if overlong user input can reach the vulnerable Kernel / System Call
5) Block access to the vulnerable ABAP code or use a data type that is too short for an exploit

Summary

There are severe security risks in ABAP, too:

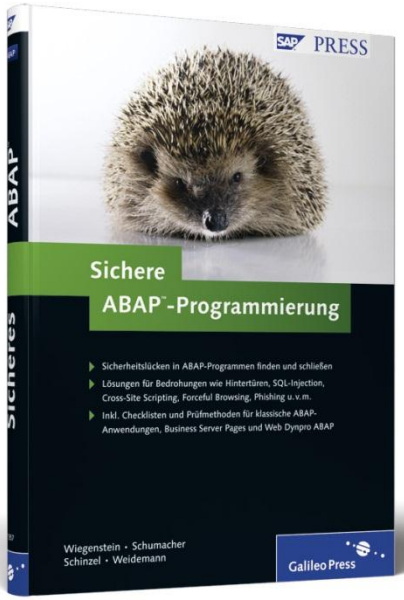
- (Custom) ABAP Code can bypass security features of the SAP standard and thereby violate regulatory compliance.
- Kernel Calls provide lots of (undocumented) functionality that can bypass SAP security mechanisms
- Kernel Calls can also introduce (remote exploitable) buffer overflow risks

Secure ABAP Development

Organizations

BIZEC – Business Security Initiative
http://www.bizec.org

Literature



"Secure ABAP-Programming"
(German only)
SAP Press 2009

SAP Security Notes

- SAP Security Note 1493516 – "Correcting buffer overflow in ABAP system call"
  - Covers the buffer overflow in the SYSTEM-CALL
- SAP Security Note 1487330 – "Potential remote code execution in SAP Kernel"
  - Covers the 5 buffer overflows in the Kernel Calls

Q & A

"... So, after all these slides, you must have lots of questions to ask. Am I right?"

"Yes, of course.

Lots of questions."

"… Well. What are they?"

Disclaimer

SAP, R/3, ABAP, SAP GUI, SAP NetWeaver 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.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only.

The author assumes no responsibility for errors or omissions in this document. The author does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

The author shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of this document.