Developer’s Guide to smbd

Ralph Böhme, Samba Team, SerNet
2020-05-27
Outline

Fileserver Overview

Tour de smbd: Libraries and Subsystems

Dive into the smbd sources: from main() to SMB2_CREATE processing

Reference: Sourcecode-Tree Overview

Links

Q&A
Filesver Overview
SMB fileserver

- `smbd` process, parent and childs
- Forking architecture: forks a child process per connection
  - parent listens for SMB connections on port 445
  - `accept()` TCP connections and calls `fork()`
  - from then on child handles the connection

`smbd` child

- SMB frontend: network IO, SMB packet processing
- Windows NTFS semantics to POSIX translator
  (which contributes much of the code in `smbd`)
- Local filesystem access via VFS abstraction
- `smbd` is not just a fileserver:
  - database for user accounts and groups in a Windows fashion
    (SAM, Security Account Manager)
  - can be queried over the network with RPC
Why `fork()`?

One smbd process per connection

- **security**: let the OS do all the permission checking by switching uid (and gid, gids)
- **reliability**: if one process crashes others are unaffected
- **scalibilty**: a lot of sync processing which gets nicely parallelized by having one process per connection
smbd: processes, sessions, tcons

(Without multi-channel)
winbindd

- nsswitch module
  - nsswitch is the plugin mechanism in UNIX to use different sources for defining users and groups
  - allows users and groups from Windows domains appear as OS users/groups
- Domain authentication proxy
  - smbd forwards authentication requests to winbindd
- Identifier mapping:
  - Windows SIDs to/from POSIX ids (idmapping)
  - SIDs to/from names
- Caching
  - only talk to DCs is something is not in the cache
Authentication Aftermath: the Big Picture

After a user has been authenticated we need to fill in a valid UNIX account.

What do we need?

- A name for the UNIX account
- At least `struct passwd` home-directory
- uids and gids to impersonate the user at the UNIX level

What do we get?

- Names: the domain and username from the authentication process
- SIDs: User SID and auxilliary group SIDs from the PAC

What do we do?

- See next slide ...
Authentication Aftermath: Building a Useraccount

1. `getpwnam()`
   Get user account

2. `winbind_request_response()`
   WINBIND_GETPWNAM

3. RPC: `lsa_lookupnames()`
   Name -> SID

4. `wb_sids2xids_send()`
   User SIDs to uid

5. + 9. `idmap_rid`
   Algorithmic mapping

6. + 10. `ldap_search()`
   LDAP

7. `sids_to_unixids()`
   Map gids for UNIX token

8. `wb_sids2xids_send()`
   Group SIDs to gids

9. `idmap_ad`

10. `idmap_rid`

11. `nsswitch`

12. `smbd`
Tour de smbd: Libraries and Subsystems
Main important libraries

- event handling and async processing abstraction: `tevent`
- hierarchical memory allocator: `talloc`
- key/value store: `tdb`, typically used via the `dbwrap` API
- Async socket programming: `tsocket`
- sources of all four in `lib/

**Tutorials:**

- [https://tevent.samba.org/tevent_tutorial.html](https://tevent.samba.org/tevent_tutorial.html)
- `lib/tsocket/tsocket_guide.txt`
Important subsystems

VFS: abstract filesystem interface

- Abstract interface for POSIX filesystem functions
- Plus bunch of other higher level functions
- Loadable and configurable modules implement (parts of) interface
- Much like FUSE for the kernel
- VFS module `vfs_default` maps to POSIX calls
- VFS module that looks into SQL database is possible
- Tutorial: Writing a Samba VFS Module

source3/modules/
Important subsystems, cont.

IPC: messaging based on UNIX datagram sockets

- Mainly for communication between processes
- Support sending to self

/* Sending messages */
NTSTATUS messaging_send(struct messaging_context *msg_ctx, ...);
void messaging_send_all(struct messaging_context *msg_ctx, ...);

/* Listening for messages */
struct tevent_req *messaging_filtered_read_send(...);
struct tevent_req *messaging_read_send(...);
dbwrap: abstract database interface

- Samba uses TDB databases to store various internal bits
- TDB is not clustered, so for clustering ctdb was invented
- API to abstract away locking details and non-clustered vs clustered usecase
- voilà: dbwrap, an API with backends
  (TDB, ctdb, in-memory Red-Black-Tree, ...
Important subsystems, cont.

Authentication components (madness lies here...)  

- **gensec**: Generic Security Framework  
  - provides authentication, authorisation and data integrity  
  - implements SASL, GSSAPI, SPNEGO and NTLMSSP protocols

  `auth/`

- Low level crypto routines for NTLM, DES, netlogon and schannel

  `libcli/auth/`

- smbd authentication support

  `source3/auth/`

- gensec Kerberos backend for smbd

  `source3/liprpc/crypto/`
Important subsystems, cont.

Others...

- RPC server and RPC services
  source3/rpc_server/

- pidl: Perl IDL Compiler
  pidl/

- loadparm: processing smb.conf
  lib/param/ and source3/param/

- passdb/groupdb: local SAM with users and groups
  source3/{passdb|groupdb}/
Dive into the smbd sources: from `main()` to `SMB2_CREATE` processing
Set the ball rolling
source3/smbd/server.c

main()
  `-> open_sockets_smbd()
  |   `-> smbd_open_one_socket()
  |       `-> fd = socket() + listen()
  |                `-> tevent_add_fd(fd, smbd_accept_connection)
  `-> smbd_parent_loop()

To be continued in smbd_accept_connection()...
Accept client TCP connection

source3/smbd/server.c

smbd_accept_connection()
  `-> accept() + fork()
  `-> smbd_process() (in source3/smbd/process.c)
    `-> client = smbXsrv_client_create()
    `-> sconn = talloc_zero(client, struct smbd_server_connection)
    `-> xconn = smbd_add_connection(client, fd, ...)
        `-> fde = tevent_add_fd(fd, smbd_server_connection_handler, ...)
    `-> tevent_loop_wait()

To be continued in smbd_server_connection_handler()...
Core C state objects in smbd

- **Global State**
  - "sconn"
    - struct smbd_server_connection
  - "ev_ctx"
    - struct tevent_context
  - "msg_ctx"
    - struct messaging_context

- **List for Bookkeeping**
  - File Handles
    - "fsp"
      - struct files_struct
  - Tree Connects
    - "conn"
      - struct connection_struct

- **TCP Connections**
  - "xconn"
    - struct smbXsrv_connection

- **SMB Sessions**
  - "sess"
    - struct smbXsrv_session

- **Tree Connects**
  - "tcon"
    - struct smbXsrv_tcon

"ev_ctx", "msg_ctx", "xconn", "sess", "tcon", "conn" and "fsp" are typical variable names.

SMB1

SMB2

"compat"

struct member

SerNet
Process first SMB2 packet (SMB2_NEGROT)
First source3/smbd/process.c, then source3/smbd/smb2_server.c

smbd_server_connection_handler(..., private_data)
  `-> xconn = talloc_get_type_abort(private_data, ...)
  `-> smbd_server_connection_read_handler(xconn)
    `-> receive_smb_talloc()
    `-> process_smb(xconn)
      `-> smbd_smb2_process_negprot(xconn) (in source3/smbd/smb2_server.c)
        `-> smbd_initialize_smb2(xconn, ...)
        |  `-> TALLOC_FREE(xconn->transport.fde)
        |  `-> xconn->transport.fde = tevent_add_fd(
        |  |    fd, smbd_smb2_connection_handler, ...)
        `-> smbd_smb2_request_dispatch(req)

To be continued in smbd_smb2_connection_handler()...
Processing of all subsequent SMB2 packets
source3/smbd/smb2_server.c

smbd_smb2_connection_handler(..., private_data)
  `-> xconn = talloc_get_type_abort(private_data, ...)
  `-> smbd_smb2_io_handler(xconn)
     `-> req = "prepare new SMB2 request object"
     `-> smbd_smb2_request_dispatch(req)
Processing SMB2_CREATE

`smbd_smb2_request_dispatch()` **an** SMB2_CREATE

- calls `smbd_smb2_request_process_create()`
- this is the SMB2 frontend function and parses the SMB2 packet
- calls `smbd_smb2_create_send()`

`smbd_smb2_create_send()`

- process the SMB2_CREATE-context blobs
- process SMB2 features like **Durable Handles** and **Create Replay**
- pathname processing: `filename_convert()`
  - deals with Windows case-insensitive semantics
  - ... and with wildcards
- finally: call `SMB_VFS_CREATE_FILE()`
This is where all the fun is

- Windows win32 API `CreateFileW()` function:
- `CreateFileW()` returns a file-handle like POSIX `open()`

**Additional semantics compared to POSIX `open()`**

- Sharing modes: **READ**, **WRITE** and **DELETE**
- Caching: SMB1 **oplocks** / SMB2 **leases**
- NTFS behavioural differences:
  - Delete-on-close
  - Byterange locks
  - Timestamps (writetime)
- NTFS metadata: DOS Attributes, named streams
Inside the VFS

SMB_VFS_CREATE_FILE()

- calls into the loaded VFS module stack
- every module that implements the VFS function is called in order
- SMB_VFS_CREATE_FILE() is typically only implemented by vfs_default
- vfs_default is the default VFS module, often the only one
- vfs_default calls create_file_default() in source3/smbd/open.c
Inside `create_file_default()`

- Quota support (fake-files)
- SMB2 POSIX WIP support for POSIX mode
- Calls `create_file_unixpath()`

`create_file_unixpath()`

- Calls `lease_match()`, protects against a subtle leases implementation problem
- Calls `open_streams_for_delete()` to prevent deleting a file if any of its streams is open that doesn’t allow deletion
- `SEC_FLAG_SYSTEM_SECURITY` access check. Related to SACLs (auditing support) which is not really implemented.
- Names streams support: if open is for a named stream, recurse into `create_file_unixpath()` to open the "base"-file
Inside `create_file_unixpath()` cont.

The big divide:

- for directories call `open_directory()`
- for files call `open_file_ntcreate()`
- afterwards some postprocessing:
  - set initial EAs: `set_ea()`
  - allocate disk space if requested: `vfs.allocate_file_space()`
  - ACL: set requested ACL or inherit parent dir ACL
Inside `open_file_ntcreate()`

`open_file_ntcreate()`

- Support for async CREATE: call `get_deferred_open_message_state()`
- Call the low level open routine: `open_file()`
  - `open_file()` ultimately calls into the VFS with `SMB_VFS_OPEN()` `SMB_VFS_OPENAT()`
- Deal with errors from opens that suggest a retry:
  `setup_poll_open()` or `schedule_async_open()`

**Implement NTFS stuff (still `open_file_ntcreate()`)**

- Call `lock = get_share_mode_lock()` to fetch record from `locking.tdb`
- `locking.tdb`: our central store for filehandle information
- Delete-on-close semantics: `has_delete_on_close(lock, ...)`
- Implement sharing mode and oplock/lease:
  `handle_share_mode_lease(lock, ...), set_share_mode(lock, ...)`
- DOS Attributes: `dos_mode()` and `file_set_dosmode()`
- more...
locking.tdb in a nutshell

- locking.tdb is our central open-file database
- Of course we use dbwrap API to access it
- One record per file with open filehandles:
  - record has fixed size header with struct share_mode_data
  - record data then contains struct share_mode_entry[] array with opens per file
- How does it look like?
locking.tdb struct share_mode_data

typedef [public] struct {
    hyper sequence_number;
    share_mode_flags flags;
    [string, charset(UTF8)] char *servicepath;
    [string, charset(UTF8)] char *base_name;
    [string, charset(UTF8)] char *stream_name;
    uint32 num_delete_tokens;
    [size_is(num_delete_tokens)] delete_token delete_tokens[];
    NTTIME old_write_time;
    NTTIME changed_write_time;
    [skip] boolean8 fresh;
    [skip] boolean8 modified;
    [skip] boolean8 have_share_modes;
    [ignore] file_id id; /* In memory key used to lookup cache. */
} share_mode_data;
locking.tdb record format, cont.

locking.tdb struct share_mode_entry

typedef [public] struct {
    server_id pid;
    hyper op_mid;
    uint16 op_type;
    GUID client_guid;
    smb2_lease_key lease_key;
    uint32 access_mask;
    uint32 share_access;
    uint32 private_options;
    timeval time;
    udlong share_file_id;
    uint32 uid;
    uint16 flags;
    uint32 name_hash;
    [skip] boolean8 stale;
}
Reference: Sourcecode-Tree Overview
source3/ subdirectories, part 1

auth/ ............ Authentication stack
client/ .......... Home of smbclient
groupdb/ ......... Local SAM (passdb and groupdb)
include/ ........ Well, includes :)
lib/ ............ Kitchensink for various utility functions
libads/ ........ Kerberos, LDAP and SASL client wrappers
libgpo/ ........ GPO client extensions
libnet/ .......... librarar for automated join, remote SAM syncing
librpc/ .......... RPC server helpers
librpc/crypto/ ... gensec Kerberos backend
librpc/idl/ ...... IDLs for various SMB related structures
libsmb/ .......... SMB client library
locking/ ......... Deal with storing Windows NTFS state in locking.tdb
modules/ ......... The home of the VFS modules
source3/ subdirectories, part 2

param/ .......... source3 loadparm backend, smb.conf processing
passdb/ .......... Local SAM (passdb and groupdb)
pidl/ .......... Perl IDL Compiler
printing/ ........ Printserver implementation
profile/ ........ Fileserver profiling
registry/ ........ Implementation of a Windows registry
               (optional smb.conf backend)
rpc_client/ ..... RPC client interfaces
rpcclient/ ...... RPC client CLI tool
rpc_server/ ..... RPC server and services implementations
selftest/ ........ source3/ CI test driver (file tests.py)
smbd/ .......... Home of the core fileserver
torture/ ........ Home of smbtorture3, part of CI
utils/ ........... Various CLI utilities
winbindd/ ........ Well, winbindd
lib/crypto/ ....... Cryptography
lib/dbwrap/ ....... dbwrap: abstract DB interface
lib/messaging/ ... IPC: UNIX datagram messaging
lib/param/ ........ loadparm: smb.conf processing
lib/pthreadpool/ . threadpool implementation
lib/replace/ ..... Platform compatibility layer
lib/tsocket/ ..... Async socket programming
lib/cli/auth/ ..... Low level auth support: NTLM, DES, netlogin, schannel
lib/rpc/ .......... RPC core and IDLs
Links
Links

- https://tevent.samba.org/tevent_tutorial.html
- lib/tsocket/tsocket_guide.txt
- Writing a Samba VFS Module
Q&A
Thank you! Questions?

Ralph Böhme <slow@samba.org>

Samba Team, SerNet