北京大学 PEKING UNIVERSITY

瀚高 HIGH GO

TDE 项目开发实践

开源开发实践-第八周

David & Cary

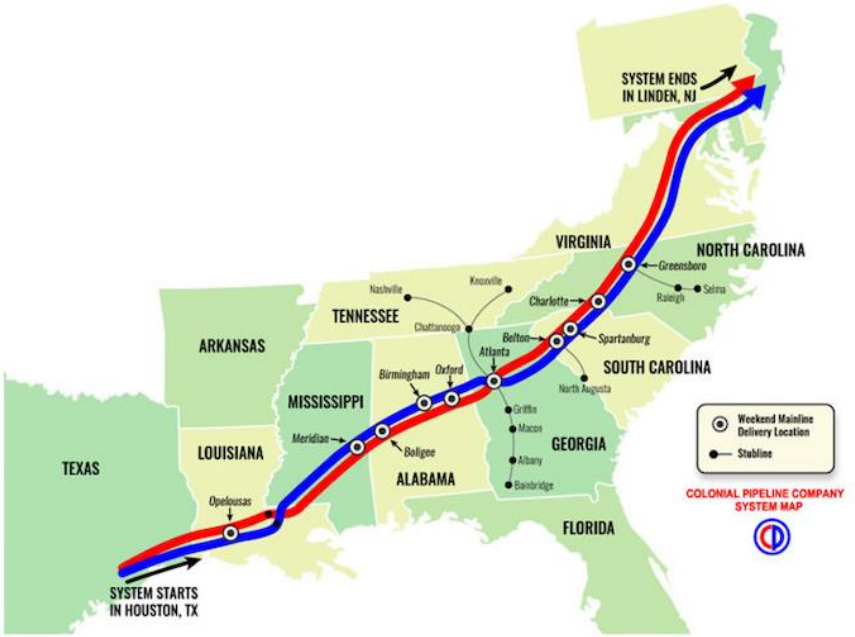
AGENDA
目 录

- Breaking News
- Write Ahead Log (WAL)
- The Impacts of TDE
- Introduction to TDE Project
- Group Assignment

Breaking News

What is happing in the world?

- On **May 7, 2021**, Colonial Pipeline suffered a ransomware cyberattack that impacted computerized equipment managing the pipeline.
- In response, Colonial Pipeline Company halted all of the pipeline's operations to contain the attack.
- It was the largest cyberattack on an oil infrastructure target in the history of the United States.
- President Joe Biden declared a state of emergency on May 9.
- The FBI and various media sources identified the criminal hacking group DarkSide as the responsible party.
- The same group is believed to have **stolen 100 gigabytes of data** from company servers the day before the malware attack.



https://en.wikipedia.org/wiki/Colonial_Pipeline_cyberattack

Breaking News

What is the impact?

Write Ahead Log

Write Ahead Log

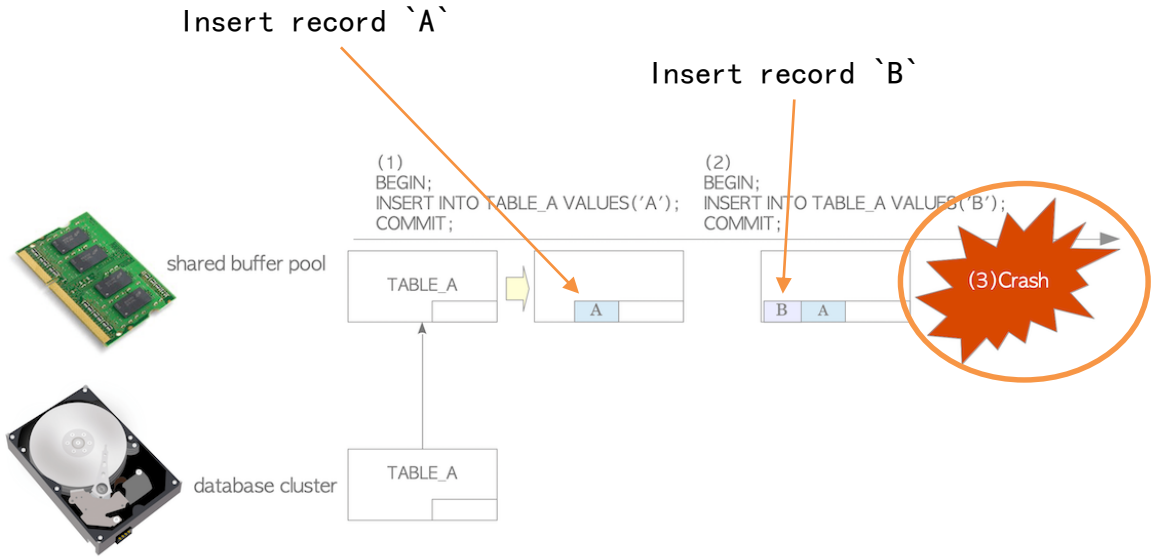
What is WAL?

- Write Ahead Log (WAL) is an essential part of database. It is a history log of all changes and actions in a database system so as to ensure that no data has been lost due to failures, such as **server crash, power failure** etc.
- As the WAL contains sufficient information about each transaction executed already, the database server should be able to recover the database cluster by replaying changes and actions in the WAL in case of the server crash.
- It also made possible the implementation of the Point-in-Time Recovery (PITR) and Streaming Replication (SR).
- *As Postgres being more than 30 years old and developed by people around the world, there was a naming inconsistency issue.*
- Since Postgres 10, the core team has made the difficult change of removing references to "xlog" and "clog," and instead name them "wal" and "pg_xact" consistently.

Write Ahead Log

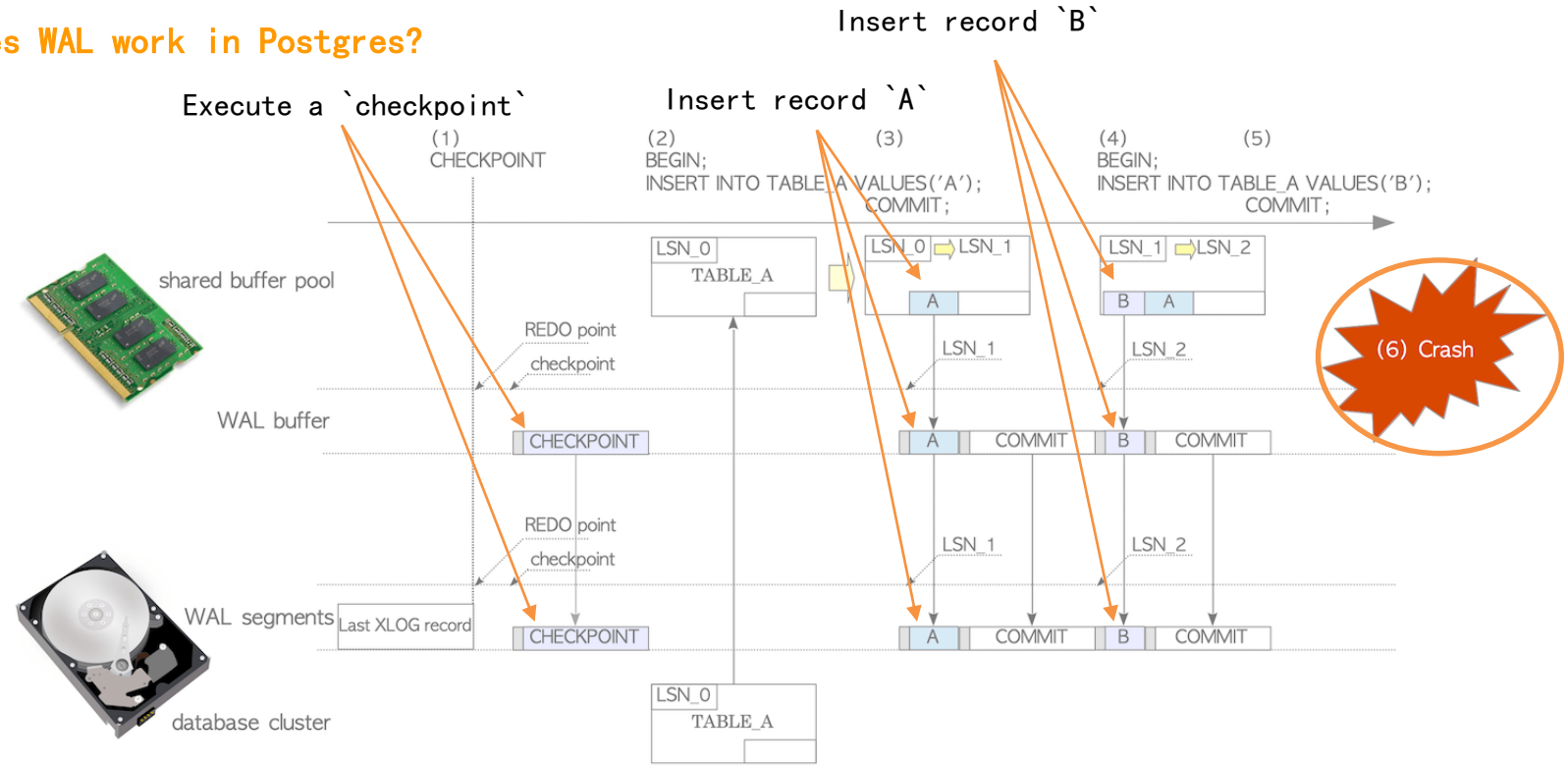
What may happen without WAL?

- One solution is to do the synchronous write whenever there is a page write operation.
- PostgreSQL did synchronous writes to the disk by issuing a sync system call in order to ensure durability.
- The modification commands such as INSERT and UPDATE were very poor-performance.
- Can we make it better?



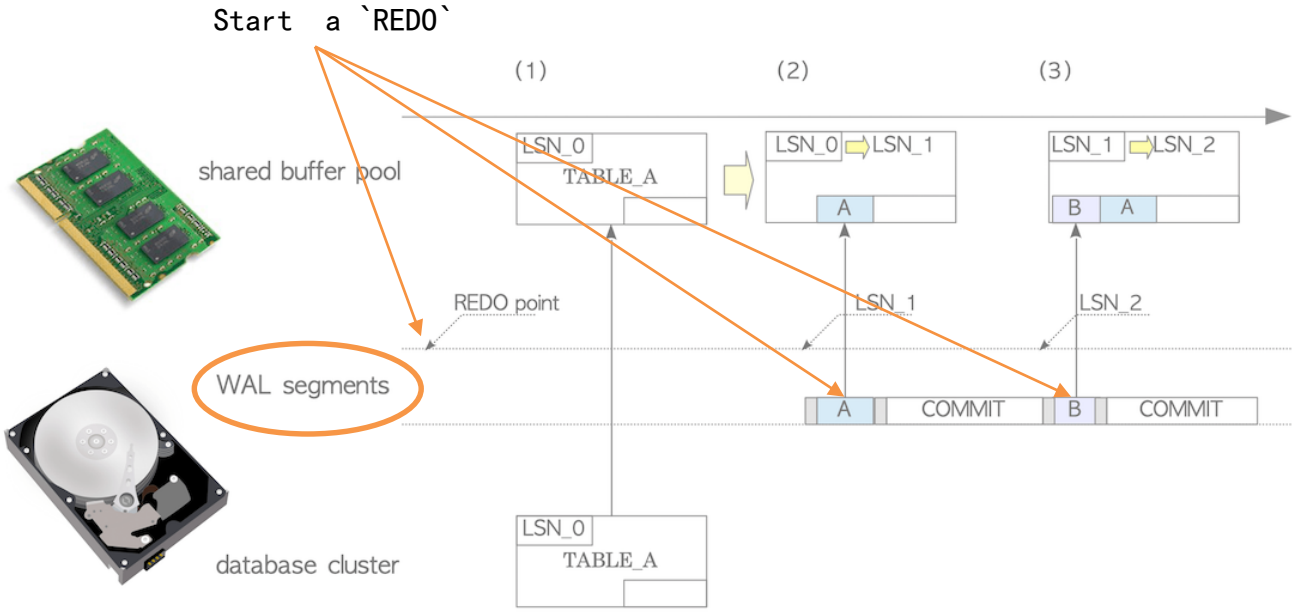
Write Ahead Log

How does WAL work in Postgres?



Write Ahead Log

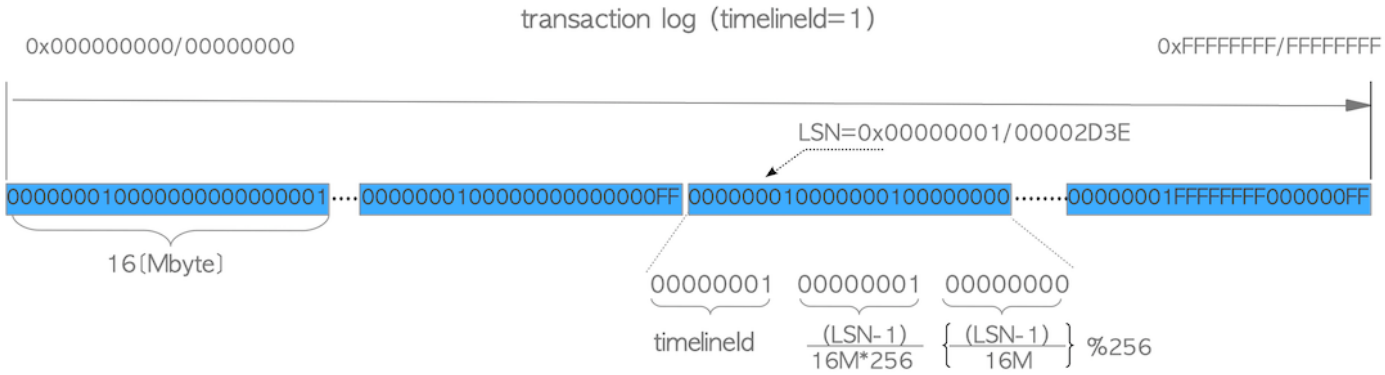
How does WAL work in Postgres?



Write Ahead Log

WAL Segment Files

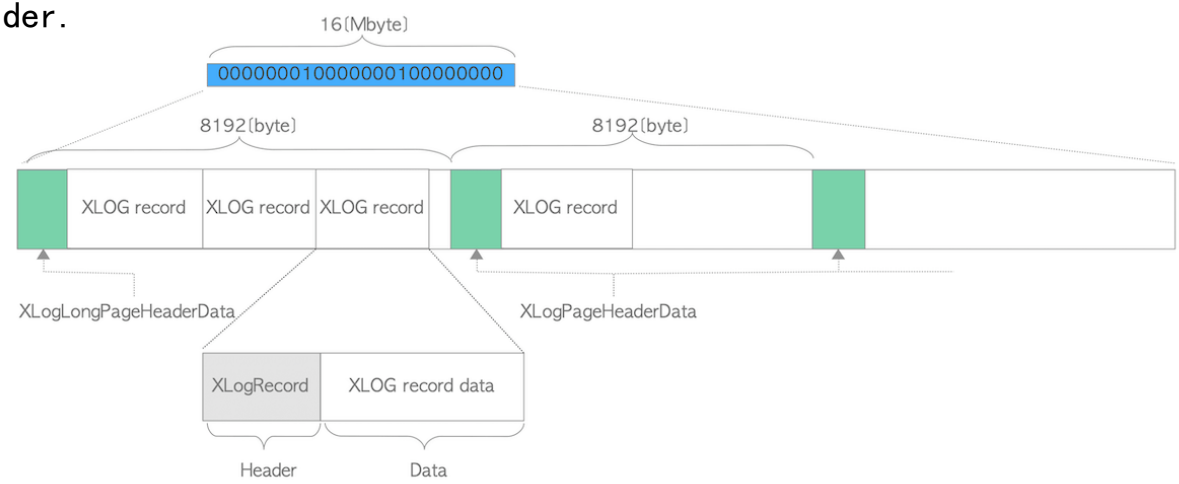
- The first WAL segment file is 000000010000000000000001. If the first one has been filled up with the writing of XLOG records, the second one 000000010000000000000002 would be provided.
- after 0000000100000000000000FF has been filled up, next one 000000010000000100000000 will be provided
- Similarly, after 0000000100000001000000FF has been filled up, 000000010000000200000000 will be provided, and so on.



Write Ahead Log

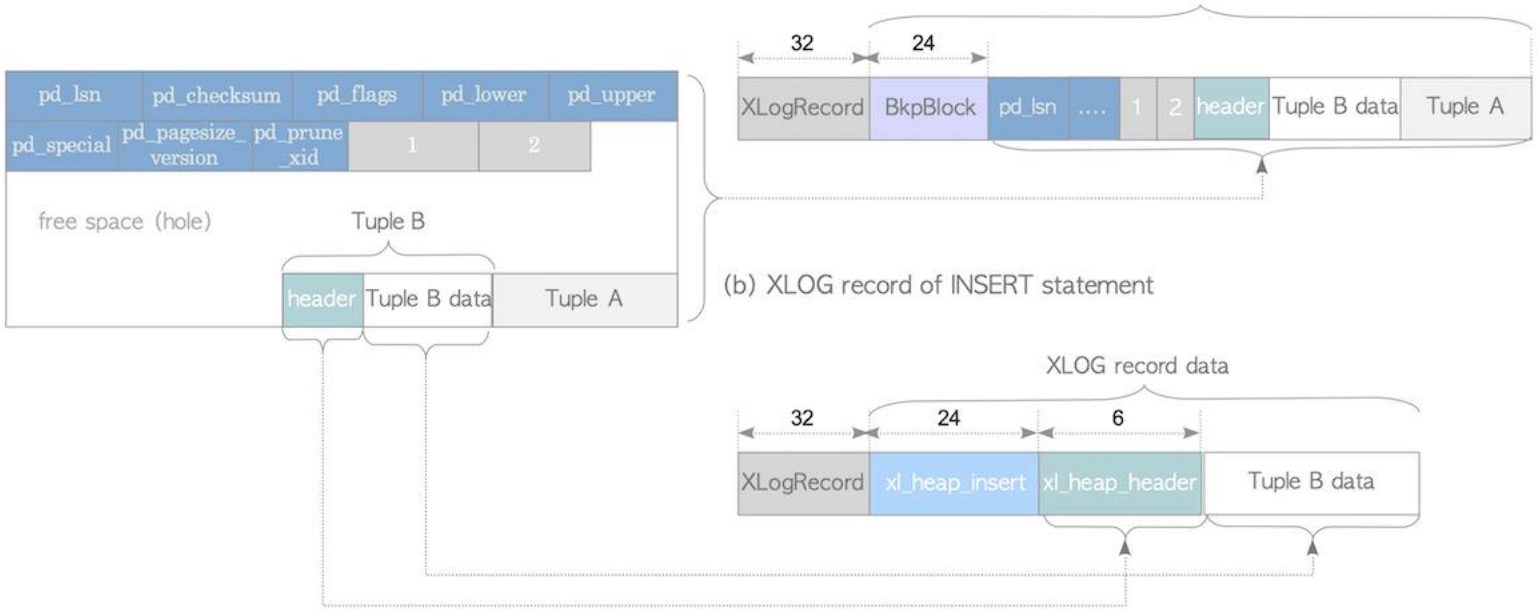
The Layout of WAL Segment

- A WAL segment is a 16 MB file, by default, and it is internally divided into pages of 8192 bytes.
- The first page has a header-data defined by the structure *XLogLongPageHeaderData*, while the headings of all other pages have the page information defined by the structure *XLogPageHeaderData*.
- Following the page header, XLOG records are written in each page from the beginning in descending order.



Write Ahead Log

The Layout of XLOG Record



Write Ahead Log

What is Checkpoint?

- The checkpointer is a background process, and its process starts when one of the following occurs:
  - The interval time set for *checkpoint_timeout* (default 300 seconds) from the previous checkpoint has been gone over.
  - The total size of the WAL segment files in the *pg_wal* has exceeded the value of the parameter *max_wal_size* (default 1GB/64 files).
  - Its process also does it when a superuser issues *CHECKPOINT* command manually.

Write Ahead Log

What is pg_control File?

- The *pg_control* file contains the fundamental information of the checkpoint. If it is broken or unreadable, the recovery process cannot start up.
- three items to be required in the next section are shown in the following:
  - State - The state of database server at the time of the latest checkpointing starts, such as '*start up*', '*shut down*' and '*in production*' etc.
  - Latest checkpoint location - LSN Location of the latest checkpoint record.
  - Prior checkpoint location - LSN Location of the prior checkpoint record.
- A *pg_control* file is stored in the global subdirectory under the base-directory; its contents can be shown using the *pg_controldata* utility.
- TDE and KMS may need to add some critical information to pg_control file.

Write Ahead Log

Reference

- *http://www.interdb.jp/pg/*

Impacts of TDE

General Overview

Security is great...but

- There is a saying that I always put in my mind.

  "The more secured it is, the less convenient it becomes…"
- This statement is so true that it can be applied to many different things, especially in software.
- Some application requires you to enter a special pin code that will be sent to your cellphone after you provide your password
  - => more work to complete login
- Some application uses TLS certificate to authenticate servers. What if the certificate expires?
  - => more maintenance work to ensure security
- So… what negative impacts does TDE have?

Impacts Of TDE

TDE is Great! But...

- Data is encrypted by TDE to protect against someone who stole the physical hard disk.
- That is great because many organizations require data to be encrypted on disk.
- But at the same time, doing so would break many other software or tools who also rely on the physical data files on disk.
- To allow the other software or tools to understand the encrypted data, they must also have access to your encryption keys.
- Hm... But we are taught not to share the keys with others because there is always a risk of keys being stolen.

Impacts Of TDE

TDE is Great! But…

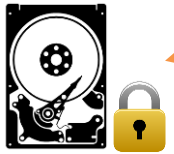
We want to **prevent** hacker from stealing our data

hacker

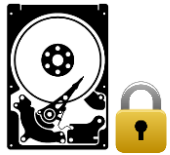
Request data via psql:
This is OK because PG will decrypt the data for you

PostgreSQL

TDE

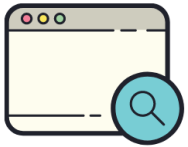


Encrypted Buffer Data

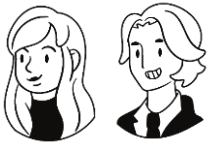


Encrypted WAL Data

We **want** this application to **access** our data, but it can't because it is encrypted!
… therefore TDE broke this app!



Application / Tools



You

PostgreSQL Front End Tools

Tools that come with PG

- Most of the front-end tools are located in your bin/ folder after you finish building the PostgreSQL.
- It looks like in the picture below:
- We will talk about some of these that actually depends on the actual physical data in order to work.

```
clusterdb     dropuser          pg_basebackup   pg_controldata   pg_isready      pg_restore      pg_upgrade      postmaster
createdb      ecpg              pgbench         pg_ctl          pg_receivewal   pg_rewind       pg_verifybackup psql
createuser    initdb            pg_checksums    pg_dump         pg_recvlogical  pg_test_fsync   pg_waldump      reindexdb
dropdb        pg_archivecleanup pg_config       pg_dumpall      pg_resetwal     pg_test_timing  postgres        vacuumdb
```

Pg_waldump

WAL utility tool

- Pg_waldump is a utility tool that can be used to display human-readable rendering of the WAL files in a PostgreSQL cluster.
- This tool is mainly used for debugging and educational purposes.
- The tool requires you to specify a start “LSN” (Logical Sequence Number), in which you can use this SQL function:

SELECT pg_current_wal_insert_lsn()”

to get the current LSN of the database

```
caryh@HGPC01:~/highgo/git/HES.TDE/postgres$ highgo/bin/pg_waldump --help
pg_waldump decodes and displays PostgreSQL write-ahead logs for debugging.

Usage:
  pg_waldump [OPTION]... [STARTSEG [ENDSEG]]

Options:
  -b, --bkp-details      output detailed information about backup blocks
  -e, --end=RECPTR       stop reading at WAL location RECPTR
  -f, --follow           keep retrying after reaching end of WAL
  -n, --limit=N          number of records to display
  -p, --path=PATH        directory in which to find log segment files or a
                         directory with a ./pg_wal that contains such files
                         (default: current directory, ./pg_wal, $PGDATA/pg_wal)
  -q, --quiet            do not print any output, except for errors
  -r, --rmgr=RMGR        only show records generated by resource manager RMGR;
                         use --rmgr=list to list valid resource manager names
  -s, --start=RECPTR     start reading at WAL location RECPTR
  -t, --timeline=TLI     timeline from which to read log records
                         (default: 1 or the value used in STARTSEG)
  -V, --version          output version information, then exit
  -x, --xid=XID          only show records with transaction ID XID
  -z, --stats[=record]   show statistics instead of records
                         (optionally, show per-record statistics)
  -?, --help             show this help, then exit

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
caryh@HGPC01:~/highgo/git/HES.TDE/postgres$
```

Pg_waldump

Common Usage

```
postgres=# CREATE TABLE test_table(x integer);
CREATE TABLE
postgres=# INSERT INTO test_table(x) SELECT y FROM generate_series(1, 10) a(y);
INSERT 0 10
postgres=# SELECT pg_current_wal_insert_lsn()
postgres-# ;
 pg_current_wal_insert_lsn
---------------------------
 0/15E4C38
(1 row)

postgres=# INSERT INTO test_table(x) SELECT y FROM generate_series(1, 5000000) a(y);

INSERT 0 5000000
postgres=#
postgres=# SELECT pg_current_wal_insert_lsn()
postgres-# ;
 pg_current_wal_insert_lsn
---------------------------
 0/147F76A0
(1 row)
```

- pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -b
- pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -b -r XLOG -n 100
- pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -b -r XLOG -f
- pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -b -n 100
- pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38

Pg_waldump

Working Examples

pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -n 20

```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres$ highgo/bin/pg_waldump -p test/pg_wal/ -s 0/15E4C38 -e 0/147F76A0 -n 20
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4C38, prev 0/015E4C00, desc: INSERT off 11 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4C78, prev 0/015E4C38, desc: INSERT off 12 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4CB8, prev 0/015E4C78, desc: INSERT off 13 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4CF8, prev 0/015E4CB8, desc: INSERT off 14 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4D38, prev 0/015E4CF8, desc: INSERT off 15 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4D78, prev 0/015E4D38, desc: INSERT off 16 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4DB8, prev 0/015E4D78, desc: INSERT off 17 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4DF8, prev 0/015E4DB8, desc: INSERT off 18 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4E38, prev 0/015E4DF8, desc: INSERT off 19 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4E78, prev 0/015E4E38, desc: INSERT off 20 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4EB8, prev 0/015E4E78, desc: INSERT off 21 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4EF8, prev 0/015E4EB8, desc: INSERT off 22 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4F38, prev 0/015E4EF8, desc: INSERT off 23 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4F78, prev 0/015E4F38, desc: INSERT off 24 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4FB8, prev 0/015E4F78, desc: INSERT off 25 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4FF8, prev 0/015E4FB8, desc: INSERT off 26 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E5038, prev 0/015E4FF8, desc: INSERT off 27 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E5078, prev 0/015E5038, desc: INSERT off 28 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E50B8, prev 0/015E5078, desc: INSERT off 29 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E50F8, prev 0/015E50B8, desc: INSERT off 30 flags 0x00, blkref #0: rel 1663/12709/16384 blk 0
```

Pg_waldump

Working Examples

pg_waldump -p $PGDATA/pg_wal -s 0/15E4C38 -e 0/147F76A0 -b -n 20 -b

```
caryh@HGPC01:~/highgo/git/postgres.community2/postgres$ highgo/bin/pg_waldump -p test/pg_wal/ -s 0/15E4C38 -e 0/147F76A0 -b -n 20
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4C38, prev 0/015E4C00, desc: INSERT off 11 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4C78, prev 0/015E4C38, desc: INSERT off 12 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4CB8, prev 0/015E4C78, desc: INSERT off 13 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4CF8, prev 0/015E4CB8, desc: INSERT off 14 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4D38, prev 0/015E4CF8, desc: INSERT off 15 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4D78, prev 0/015E4D38, desc: INSERT off 16 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4DB8, prev 0/015E4D78, desc: INSERT off 17 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4DF8, prev 0/015E4DB8, desc: INSERT off 18 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4E38, prev 0/015E4DF8, desc: INSERT off 19 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4E78, prev 0/015E4E38, desc: INSERT off 20 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4EB8, prev 0/015E4E78, desc: INSERT off 21 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4EF8, prev 0/015E4EB8, desc: INSERT off 22 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4F38, prev 0/015E4EF8, desc: INSERT off 23 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4F78, prev 0/015E4F38, desc: INSERT off 24 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4FB8, prev 0/015E4F78, desc: INSERT off 25 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E4FF8, prev 0/015E4FB8, desc: INSERT off 26 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E5038, prev 0/015E4FF8, desc: INSERT off 27 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E5078, prev 0/015E5038, desc: INSERT off 28 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E50B8, prev 0/015E5078, desc: INSERT off 29 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
rmgr: Heap        len (rec/tot):     59/    59, tx:        488, lsn: 0/015E50F8, prev 0/015E50B8, desc: INSERT off 30 flags 0x00
       blkref #0: rel 1663/12709/16384 fork main blk 0
```

Pg_waldump

What if you run pg_waldump on a TDE enabled PostgreSQL?

- It will simply not work…
- The tool tries to read the header of the WAL record to identify the total size of the WAL
- But it is encrypted, so it misinterpret the size incorrectly (-353482682 in the image below)
- We have just broken one of the many front end tools that PG provides and there are several others that are also broken

```
caryh@HGPC01:~/highgo/git/HES.TDE/HGES$ highgo/bin/pg_waldump -p tdedb/pg_wal/ -s 0/1579698 -e 0/1579698
pg_waldump: fatal: WAL segment size must be a power of two between 1 MB and 1 GB, but the WAL file "the WAL file "" header specifies -353482682 bytes
caryh@HGPC01:~/highgo/git/HES.TDE/HGES$
```

What should we do?

Well… Make them work of course

- Whatever we did in the PostgreSQL backend to encrypt the WAL and buffer data, we need the same thing in the front-end tools as well.
- The front-end tools need to have access to the same key-management routines such that it can also obtain a KEK so that it can unlock the encrypted DEKs to actually access the data.
- This is one of the reasons why TDE is going slowly in the community and also very “hard” as it also creates impacts on other software and tools that depend on the data.

Introduction to TDE Project

One of The Final Project Options For This Course

Group Organization

Let’s discuss !

HIGH GO

融知与行 瀚且高远

THANKS