Chainspace - Replay Attack Demo

We describe a demo implementation of the replay attacks against Chainspace presented in the paper "Replay Attacks and Defenses Against Cross-shard Consensus in Sharded Distributed Ledgers" (Sections 4.3 and 4.4). Attacks against Omniledger (Sections 5.3 and 5.4) can be similarly implemented.

Implementation Overview

We implemented the replay attacks in Java as a fork of Byzcuit, which itself is a fork of Chainspace (https://chainspace.io/). We released Byzcuit as an open-source project, and the attacks are also publicly available (https://github.com/sheharbano/byzcuit/tree/replay-attacks) .

Specifically, we used the 'Consensus Testing' framework of Chainspace. The test framework allows us to set up a network on a single machine, with a given number of shards and nodes (which run on local ports). The nodes run the full consensus protocol and maintain state machines. To focus on consensus, the testing mode only supports processing of simple String-based transactions. The test framework allows us to run a client that can communicate and send transactions to the nodes.

To implement the attacks in a meaningful context, we extended the consensus testing framework to support a simple payment application that supports account creation and coin transfer. The attacks described below were tested for 1 client, and 2 shards with 4 nodes per shard.

Replay Attack on Phase 1 (Section 4.3)

The attacks shown in Table 1 are all premised on an attacker's ability to replay a pre-recorded pre-abort (or pre-accept) on a target shard, which results in the shards ending up in inconsistent states. We describe the attack below, providing screenshots from our demo.

- Submit transaction to create account for Bob (balance: 30 coins) on Shard 0 and Alice (balance: 10 coins) on Shard 1.

```
=================
Accounts Table
=================

Account ID: bob Balance: 30 Status: 0 Shard: 0

------------------
```

```
=================
Accounts Table
=================

Account ID: alice Balance: 10 Status: 0 Shard: 1

------------------
```

- Submit transaction to transfer 5 coins from Bob to Alice.
    - Shard 1 (managing Alice's account) locally decides to commit and sends pre-accept to Shard 0 (managing Bob).

- Similarly, Shard 0 (managing Bob's account) locally decides to commit and sends pre-accept to Shard 1 (managing Alice's account).

- **Correct Execution:** In a correct execution, both shards will observe 2 pre-accept (1 from itself, and 1 from the other shard) and decide to commit. As a result:

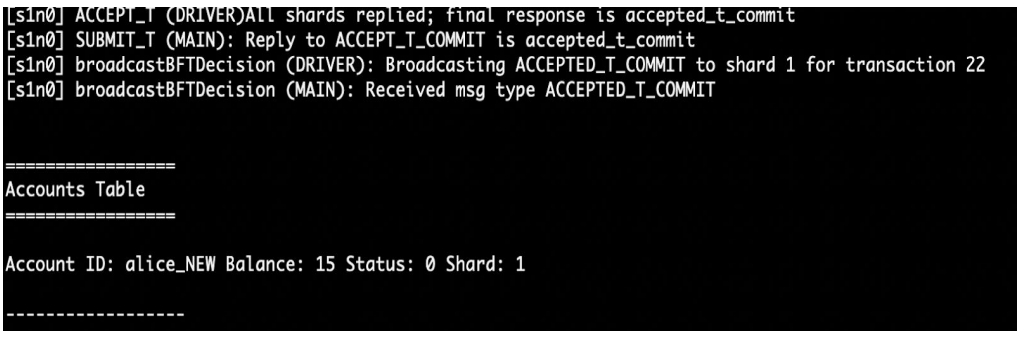
    - Shard 1 will mark Alice's account (balance 10) as inactive and purge it from its table (nodes do not keep track of a growing list of inactive objects). It will create a new account object for Alice with the updated balance of 15 coins.

    - Similarly, Shard 1 will mark Bob's account (balance 30) as inactive and purge it from its table (nodes do not keep track of a growing list of inactive objects). It will create a new account object for Bob with the updated balance of 25 coins.

- **Incorrect Execution due to Replay Attack**: We modified the code so the pre-accept from Shard 1 (managing Alice's account) to Shard 0 (managing Bob's account) is replaced with pre-abort.

    - As a result, though Shard 0 (managing Bob's account) locally decides to commit but ultimately aborts the transaction after receiving the (injected) pre-abort from Alice. Shard 0 unlocks Bob's account so it's available for future transactions.

    - Protocol execution at Shard 1 (managing Alice's account) is the same as in correct execution, i.e., Shard 1 marks Alice's account (balance 10) as inactive and purges it from its table (nodes do not keep track of a growing list of inactive objects). It will create a new account object for Alice with the updated balance of 15 coins.

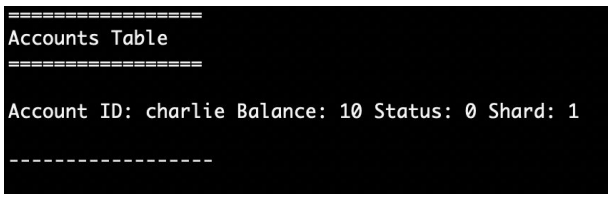


- To summarize, Bob transferred 5 coins to Alice out of thin air (i.e. these 5 coins are not deducted from Bob's balance). As described in the paper, this attack exploits lack of binding between the messages emitted at different phases of the protocol, creating inconsistent state machine executions on different shards. As such, the attack does not rely on any strict timing assumptions. Indeed, in the attack described above (as well as others described in Table 1 in the paper), the same entity could control the accounts of both Alice and Bob, as well as the 'client', and generate coins out of thin air!

Replay Attack on Phase 2 (Section 4.4)

The attacks described in Table 2 of the paper are premised on the attacker's ability to replay accept messages to a shard that only manages a transaction's output object(s) and has no context about the prior protocol execution steps. Upon receiving accept message, a shard creates the transaction's output objects (ones that fall under its management). We describe the attack below, providing screenshots from our demo. Logs from the demo are attached.

- Submit transaction to create account for Dave (balance: 30 coins) on Shard 0 and Charlie (balance: 10 coins) on Shard 1.



- Submit transaction to transfer 5 coins from Dave to Charlie.
    - Shard 1 will mark Charlie's account (balance 10) as inactive and purge it from its table (nodes do not keep track of a growing list of inactive objects). It will create a new account object for Charlie with the updated balance of 15 coins.

- Similarly, Shard 0 will mark Dave's account (balance 30) as inactive and purge it from its table (nodes do not keep track of a growing list of inactive objects). It will create a new account object for Bob with the updated balance of 25 coins.

```
=================
Accounts Table
=================

Account ID: dave_NEW Balance: 25 Status: 0 Shard: 0

------------------
```

- Submit transaction (i.e. accept message) to create account for Charlie with balance of 10 coins on Shard 1.
    - This transaction should be aborted because an account object with same ID has become inactive and cannot be re-activated. However, as the inactivated Charlie account has been purged from the shard's table, this transaction will go through, creating 10 coins in Charlie's account out of thin air.

    - We have implemented the demo attack for just one attack sequence, but it is trivial to repeat these steps a large number of times, generating coins out of thin air into Charlie's account. This attack is possible because (1) shards do not keep track of inactivated objects, and (2) a shard has no context of the prior protocol executions that led to the creation of output objects. The first limitation has no easy solution because nodes have to ultimately prune inactive objects to achieve linear scalability (which is the selling point of these protocols). Byzcuit addresses the second limitation by introducing 'dummy objects' that allow the output shards to witness all protocol steps and decide whether account creation is legitimate in the given context.

```
=================
Accounts Table
=================

Account ID: charlie_NEW Balance: 15 Status: 0 Shard: 1
Account ID: charlie Balance: 10 Status: 0 Shard: 1

------------------
```

Running the Demo

Installation

- git clone https://github.com/sheharbano/byzcuit.git
- git checkout replay-attacks
- pip install -e chainspaceapi
- pip install -e chainspacecontract
- >> From 'byzcuit/chainspacecore'
    - mvn package assembly:single

- >> Note: # If the above line generates error “[ERROR] javac: invalid target release: 1.8”, then explicitly set the path as follows:
    - Find the path:
        - /usr/libexec/java_home -v 1.8
    - Open terminal, and set the path determined in the previous step:
        - export JAVA_HOME=<path>

Running the demo

- >> To start the network of nodes
    - contrib/core-tools/easystart.mac.sh

- >> To start the console client, from ‘byzcuit/chainspacecore’ run the following and follow prompts.
    - ./runconsoleclient.sh