

# Multi-trace Concolic Execution Framework

Kyel Ok, Joonwon Choi, and Chanwoo Chung

{kyelok, joonwonc, cwchung}@mit.edu

## 1 Introduction

Concolic execution is a powerful tool to automatically detect bugs that are difficult to encounter in a normal workflow of an application. Since rarely used parts of an application is often less tested, an adversary could exploit bugs in such places by carefully formulating the inputs to the application. Concolic execution provides an automated way of preventing such attacks by testing many branches of the system workflow and catching bugs that are rarely encountered otherwise.

While standard concolic execution is great for single-user applications, it is difficult to achieve high code coverage when testing multi-user applications due to non-determinism introduced. In a single-user application, the state of an application, i.e., the path conditions, is only dependent on the inputs of the single user; by changing the inputs of the user, concolic execution can cover all the path conditions possible. However, for a multi-user application, the state of an application is a function of all of the users sharing the same resources. Thus, without observing the other users, concolic execution performed on a single user would result in a non-deterministic behavior.

An example of a non-deterministic behavior observed on a multi-client application would be when a client tries to register a new user on a web-based application. If the application was a single-client application, i.e., only accepts one connection at all times, and if it reset its database before each concolic execution run, registering the same user would always work deterministically. However, if there was even one other client having unobservable interactions with the server, registering the same user may fail if that client registers the user beforehand.

Non-deterministic behavior in multi-client applications is an artifact of unobservability of the entire system. Since a computer is a deterministic machine, there is no true randomness or non-determinism in its applications. There are only seemingly non-deterministic results that are caused by the lack of the ability to quantize all of the factors of an event. For example, in the scenario for user registration discussed earlier, if the concolic execution system knew all the inputs of the other clients it could deterministically conclude if registering a user would fail or not. In other words, if a concolic execution system could precisely observe all the interactions of all the users, it could deterministically define the current state.

Based on the notion of non-determinism being the direct result of unobservability, we propose a multi-trace concolic execution framework that can achieve multi-user observability. We propose to have a high-level process that determines

the number of clients interacting with an application, inputs each client can use, and the timing difference between the actions of the clients. The path conditions for the high-level process would expand to include the actions of all the users and therefore be able to observe the actions of each user.

## 2 Multi-trace concolic execution framework

### 2.1 Base framework

Multi-trace concolic execution we propose starts several parallel concolic execution client threads that interact with a server. The idea is that by having each thread represent a single client interacting with the server, we can simultaneously assign and control all the clients sharing the same resources.

We systematically expand the inputs to a multi-client application using four growing variables.

- *Concrete values.* A list of possible concrete values for any single client is congregated when they are discovered from attempting to branch in new ways as done in Lab 3<sup>1</sup>.
- *Permutation.* Then, using such inputs, we form all possible permutations of the inputs that can be assigned across the client threads.
- *Processes.* Once each of the unique permutation has been tested, we increase the number of the processes, and form a new permutation of the input set for the bigger number of processes.
- *Time offset.* Lastly, we try different start times of the client threads for the exact same inputs to capture the full state of an application.

Since clients interacting with a server may produce different results depending on the order and the timing of commands executed, including the time variable is beneficial. Finally, this exponential and infinite growth of the input space is continuously exploited to explore the states of a multi-client application until the number of parallel processes reaches a user-defined maximum.

### 2.2 Heuristics for a speed boost

**Removing the time offset** The first heuristic we use to speed up the multi-trace concolic execution system is removing the time difference variable and executing all the client threads in parallel. We apply our prior knowledge that most multi-client bugs are caused by a tight race condition between clients. Thus, it is most likely to reveal a bug when multiple clients execute commands simultaneously. We confirm this theory by manually adjusting the time offset between client threads and verifying the number of bugs the multi-trace concolic execution system finds.

As seen in Fig. 1, the results are non-deterministic as the same offset between clients may or may not produce bugs. We suspect that the non-determinism is

---

<sup>1</sup> <http://css.csail.mit.edu/6.858/2014/labs/lab3.html>

caused by the lack of control over precise timing between the threads, i.e., the best we can do is `sleep()` on an OS that is not real-time, and the lack of observability of the CPU scheduler that services all the parallel threads in a linear fashion.

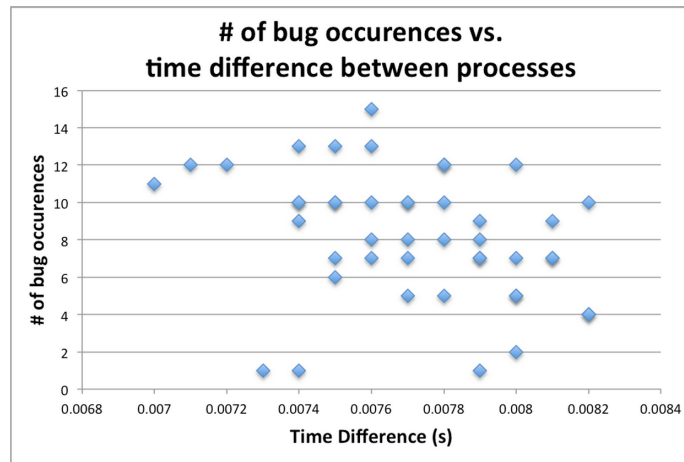


Fig. 1. Bug occurrences and time difference

**Constraining the concrete input values** Another heuristic we can apply is to limit the possible input values, i.e. the concrete values a client can use, to the ones that are more likely to cause a multi-client bug. Since we have prior knowledge that multi-client bugs are mostly caused by invoking shared resources, we can limit the inputs to those that include a path to sharing resources. In our experiments, this limitation results in a large speed boost while still being able to reveal the bugs found without it.

### 3 Experiments

We have run our multi-trace concolic execution system on `zoobar` application to automatically detect possible multi-client bugs that were missed in the standard single-trace concolic execution done in Lab 3.

We have found three new critical bugs and a few similar minor bugs in the `zoobar` application. The critical bugs are as extreme as crashing the web server to having `zoobars` lost. We omit the full description for minor bugs due to their similarity to the critical bugs.

### 3.1 Critical bugs found

Details of these bugs below are explained in appendix A.

- Server crashes when two users try to register with the same userid.
- When user A transfers zoobars to user B twice, only one of them is executed while both are logged.
- When user A and B transfer zoobars to user C at the same time, zoobars of A and B are withdrawn while only one of them is credited to user C.

### 3.2 Timing tests

We have compared the time taken for multi-trace concolic execution with and without the heuristics discussed earlier. Table 2 shows the results where the heuristics can result in up to a 1.2 times speed boost. The maximum number of processes was limited to two for this run.

	normal	with heuristic
Bug 1	45 seconds	4 seconds
Bug 2	37 seconds	28 seconds
Bug 3	101 seconds	82 seconds

**Fig. 2.** Speed boost by heuristic

### 3.3 Attack script

We have developed attack scripts to exploit the bugs found using our multi-trace concolic execution framework. Using these scripts, we could verify that bugs can be emulated in the real browser. Details of emulated results are in appendix B.

## 4 Conclusions

We have developed multi-trace concolic execution to find multi-client bugs in web applications that allow several users to interact with shared resources. We used the developed concolic execution framework to uncover three new bugs in the zoobar applications, and have developed attack scripts to exploit the automatically found bugs.

## A Appendix: bugs found in the Zoobar application

### A.1 Registering two users at once

```
def register(username, password):
    db = person_setup()
    person = db.query(Person).get(username)
    if person:
        return None
    newperson = Person()
    newperson.username = username
    newperson.password = password
    db.add(newperson)
    db.commit()
    return newtoken(db, newperson)
```

**Fig. 3.** Registration function in Zoobar

As seen in Fig. 3, Before `db.commit()` is called, `register()` is invoked twice which will add the same user twice with `db.add()` this will cause the second `add()` to crash the second user's browser.

### A.2 Sending zoobar twice

Suppose *user A* invokes two `transfer()` functions before `persondb.commit()` is called. As seen in Fig. 4, `persondb` gets written with the same value twice and transfer will record two transfers when only one took place. For example, *A* invokes transfer of 5 zoobars twice. *B* receives only 5, but transfer log will show two instances of sending 5 which means *A* can claim he/she has sent 10 zoobars to *B* while he had only sent 5 zoobars.

### A.3 Sending zoobar twice by different users

We could find the other bug in the `transfer` function in Fig. 4. Suppose *user A* starts with 100 zoobars, *B* starts with 10 zoobars, and *C* starts with 10 zoobars. *A* sends *C* 100 zoobars. Right before `persondb.commit()` is called *B* finishes `get(sender)` and `get(recipient)` which returns 10 zoobars for *C* and *B*. Now, *A* finishes `persondb.commit()`, which causes *C* to be at 110 and *A* at 0. Now, *B* calls `persondb.commit()` which causes *B* to go down to 0, and cause *C* to go down to 20 zoobars instead of 130. Thus, the resulting zoobar counts would be:

- A: 0 zoobars
- B: 0 zoobars
- C: 20 zoobars

```

def transfer(sender, recipient, zoobars):
    persondb = person_setup()
    senderp = persondb.query(Person).get(sender)
    recipientp = persondb.query(Person).get(recipient)

    sender_balance = senderp.zoobars - zoobars
    recipient_balance = recipientp.zoobars + zoobars

    if sender_balance < 0 or recipient_balance < 0:
        raise ValueError()

    senderp.zoobars = sender_balance
    recipientp.zoobars = recipient_balance
    persondb.commit()

    transfer = Transfer()
    transfer.sender = sender
    transfer.recipient = recipient
    transfer.amount = zoobars
    transfer.time = time.asctime()

    transferdb = transfer_setup()
    transferdb.add(transfer)
    transferdb.commit()

```

**Fig. 4.** Transfer function in Zoobar

## B Appendix: attack result

Fig. 5, 6, and 7 shows the attack results from the bugs we mentioned in appendix section A.

### Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

**Fig. 5.** Client-side error of bug 1

```

/home/httpd/mit-6858-project/lab5/zoobar/debug.py:23 :: __try : caught exception in function login:
Traceback (most recent call last):
File "/home/httpd/mit-6858-project/lab5/zoobar/debug.py", line 20, in __try
    return f(*args, **kwargs)
File "/home/httpd/mit-6858-project/lab5/zoobar/login.py", line 81, in login
    cookie = user.addRegistration(username, password)
File "/home/httpd/mit-6858-project/lab5/zoobar/login.py", line 29, in addRegistration
    token = auth.register(username, password)
File "/home/httpd/mit-6858-project/lab5/zoobar/auth.py", line 38, in register
    db.commit()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 721, in commit
    self.transaction.commit()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 354, in commit
    self._prepare_impl()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 334, in _prepare_impl
    self.session.flush()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 1818, in flush
    self._flush(objects)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 1936, in _flush
    transaction.rollback(_capture_exception=True)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/util/langhelpers.py", line 58, in __exit__
    compat.reraise(exc_type, exc_value, exc_tb)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/session.py", line 1900, in _flush
    flush_context.execute()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/unitofwork.py", line 372, in execute
    rec.execute(self)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/unitofwork.py", line 525, in execute
    uow.
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/persistence.py", line 64, in save_obj
    table.insert()
File "/usr/lib/python2.7/dist-packages/sqlalchemy/orm/persistence.py", line 541, in _emit_insert_statements
    execute(statement, multiparams)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/base.py", line 662, in execute
    params)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/base.py", line 761, in _execute_clauseelement
    compiled_sql, distilled_params
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/base.py", line 874, in _execute_context
    context)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/base.py", line 1024, in _handle_dbapi_exception
    exc_info)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/util/compat.py", line 196, in raise_from_cause
    reraise(type(exception), exception, tb=exc_tb)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/base.py", line 867, in _execute_context
    context)
File "/usr/lib/python2.7/dist-packages/sqlalchemy/engine/default.py", line 324, in do_execute
    cursor.execute(statement, parameters)

```

Fig. 6. Server-side error of bug 1

## Zoobar Foundation for Sensible Discourse

Supporting the loyal minds of the new world order


[Log out grader](#)

[Home](#) | [Users](#) | [Transfer](#)

User:

[View](#)

grader's zoobars:3



Time	Sender	Recipient	Amount
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:00 2014	grader	attacker	1
Wed Nov 19 22:29:02 2014	grader	attacker	1

Fig. 7. Attack result of bug 2