Temp RGC Ref No. T22114416

RESEARCH GRANTS COUNCIL

Application for Allocation from the Early Career Scheme for 2016/17

Application Form (ECS1)

- **Please read the Explanatory Notes ECS(2) (Aug 15) carefully before completing this form.**
- **To safeguard the interests of the researcher and the institution, awardee institution bears the primary responsibility for prevention, detection and investigation of research misconduct, including but not limiting to misusing of funds, data falsification, plagiarism and seeking duplicate funding for projects which the applicant has already completed partially or entirely. Concerning research grant applications, the institution is strongly advised to use anti-plagiarism software before submitting the application to the RGC.**

PART I SUMMARY OF THE RESEARCH PROPOSAL

[To be completed by the applicant(s)]

1. Particulars of the Project

(a) (i) Name and Academic Affiliation of Principal Investigator:

Name	Post	Unit/ Department/ Institution
Dr Cui, Heming	Assistant Professor	Department of Computer Science/The University of Hong Kong

(ii) The applicant is within

- [✓] **first year** **(entering into employment between 13 November 2014 to 1 May 2016)**
- ☐ **second year** **(entering into employment between 13 November 2013 to 12 November 2014)**
- ☐ **third year** **(entering into employment between 13 November 2012 to 12 November 2013)**

of his/her full time academic job as an Assistant Professor or career equivalent involving teaching and research duties and the applicant is in substantiation track / tenure track position at the time of submission of the proposal and has not been granted ECS grants or awards.

The applicant confirms that he/she had not been funded under previous Early Career Scheme exercises.

(iii) Title of Project: FALCON: Modeling, Detecting, and Defending against Concurrency Attacks

(iv) Nature of Application

New ☑	**Re-submission** ☐	**Continuation** ☐

(b) (i) Primary Field: Software **& Code** 2206
Secondary Field: **& Code**

(ii) A maximum of five keywords to characterise the work of your proposal (a maximum of 30 characters for each keyword)

1) Software security
2) Concurrency bugs
3) Program slicing
4) State machine replication
5) Multithreading

(iii) Project Duration: 36 Months

(iv) Total Amount Requested: $ 1,224,960

(c) Abstract of Research comprehensible to a non-specialist (a maximum of one A4 page in standard RGC Format for attaching PDF documents or a maximum of 400 words for direct input in the text box):

Driven by the rise of multi-core hardware, multithreaded programs become a mainstream of software. Unfortunately, despite much effort, these programs are still notoriously difficult to get right; they are plagued with concurrency bugs (e.g., data races on memory). Worse, recent research reveals that many real-world concurrency bugs have led to "concurrency attacks": hackers have leveraged concurrency bugs to corrupt the critical memory of a program, to bypass security authentications, and to construct various vulnerabilities (e.g., privilege escalations). Moreover, concurrency attacks weaken or even completely bypass most existing security defense techniques, because these techniques are mainly designed for single-threaded programs. State-of-the-art neither has a rigorous model to describe how concurrency attacks manifest nor has any detection or defense approach.

This FALCON project tackles concurrency attacks by pursuing three objectives. First, we will develop a general, rigorous model to describe how concurrency attacks manifest. A main challenge for this model is figuring out how the memory corrupted by concurrency bugs propagates to vulnerable instructions. To address this challenge, we will conduct an extensive study on real-world concurrency attacks, summarize general elements on how bugs propagate to attacks, and leverage our expertise on program analysis methods to develop our model.

Second, following the model, we will construct a systematic approach for detecting attacks during the testing phase. A main challenge for this detection approach is precisely identifying several diverse types of vulnerable instructions in program source code. To tackle this challenge, we have presented a preliminary algorithm to identify one type of vulnerable instructions in our ASPLOS 2013 paper.

Third, to deal with concurrency attacks that may be missed by detection tools, we will construct a runtime defense infrastructure to protect general programs during their deployment phase. This infrastructure requires several reliability techniques, including replication techniques for tolerating program failures, and checkpoint techniques for recovering program execution state (e.g., memory) from failures. In preparation for this infrastructure, we have presented an easy-to-use replication technique in our SOSP 2015 paper.

We envision that our expertise on program analysis algorithms and our preliminary results on replication techniques will help us achieve all the three objectives. By greatly improving the reliability and security of multithreaded programs, this project will benefit almost all computer users and software service companies. This project will also spur new research questions, ideas, and tools in broad areas, including reliability, security, and concurrency.

(d) Special funding template

- ☐ **Employment of Relief Teacher under Humanities and Social Sciences Panel (see Enclosure I) (only available for applications under Humanities and Social Sciences Panel)**
- ☐ **Support for Academic Research related to Public Policy Developments (see Enclosure II)**

PART II DETAILS OF THE RESEARCH PROPOSAL

[To be completed by the applicant(s)]

RESEARCH DETAILS

1. Impact and objectives

(a maximum of 800 words in total for the long-term impact and project objectives)

(a) Long-term impact:

Multithreaded programs, ranging from real-world applications such as Apache and MySQL, to OS kernels such as Linux and Android, are already ubiquitous. Unfortunately, despite decades of effort from both academia and industry, these programs are still extremely difficult to get right. Because multithreaded programs can run into too many possible thread interleavings at runtime, concurrency bugs can easily hide in the untested thread interleavings but show up in the deployment phase, causing severe program behaviors such as wrong outputs or program crashes.

Worse, recent research shows that many concurrency bugs have led to "concurrency attacks": hackers first triggered a concurrency bug to corrupt a program's global memory (e.g., a Linux user ID), and then they leveraged the corrupted memory to construct various vulnerabilities (e.g., call to setuid(corrupted user ID)). Given crafted program inputs, many of such attacks have been easily triggered with only a few repeated executions. Once a concurrency attack succeeds, even fixing the concurrency bug by upgrading program source code or executable cannot recover from the attack because hackers have already broken in.

Concurrency attacks are extremely difficult to tackle due to two main reasons. First, most existing security defense techniques are weakened or completely bypassed by concurrency attacks, because these techniques are mainly designed for single-threaded programs and they are not good at reasoning about concurrent behaviors. Second, existing concurrent defense techniques, which mainly prevent illegal file access in TOCTTOU (time-of-check-to-time-of-use) attacks, are insufficient to prevent concurrency attacks. The reason is that concurrency attacks can be triggered by much broader types of instructions (e.g., general memory access) than TOCTTOU attacks. In addition, concurrency attacks have led to much broader types of security vulnerabilities (e.g., privilege escalation and malicious code injection) than TOCTTOU attacks. In sum, state-of-the-art neither has a rigorous model to describe how concurrency attacks manifest nor has any detection or defense approach.

This FALCON project uses a systematic, thorough methodology to tackle concurrency attacks with three objectives, including developing a concurrency attack model, constructing a detection approach, and building a runtime defense infrastructure. To show the feasibility of this project, we have presented a preliminary program analysis method for the second objective in ASPLOS 2013. In addition, we have presented our initial results on a fault-tolerant replication technique for the third objective in SOSP 2015.

The success of this project will greatly improve the reliability and security of real-world multithreaded programs, potentially benefiting almost all computer users and software service companies (e.g., PCCW, Alibaba, and Amazon). Due to this benefit, we name our project after falcon, one of the strongest birds that help human hunt prey. We envision that our models, algorithms, and software implementations will spur following research questions, ideas, and tools from widespread research areas, including security, reliability, and concurrency. Our techniques developed in this project can also be broadly applied to relevant research topics that suffer from both concurrency and security, including TOCTTOU attacks, timing channel attacks, and byzantine fault-tolerance.

(b) Objectives

[Please list the objectives in point form]

1. [To develop a general, rigorous concurrency attack model].

We will conduct an extensive study on real-world multithreaded programs, summarize general elements on how concurrency bugs propagate to attacks, and leverage our expertise on precise program analysis methods to develop the first concurrency attack model.

2. [To construct a systematic concurrency attack detection approach].

With the concurrency attack model, we will construct an approach to detect as many as concurrency attacks for the software testing phase. This approach will leverage recent automated program analysis techniques to identify concurrency bugs in program source code and vulnerable instructions these bugs may propagate to.

3. [To build a runtime defense infrastructure].

To defend against concurrency attacks that may be missed by detection tools, we will build a defense infrastructure for the software deployment phase. This infrastructure will leverage recent advanced replication techniques to tolerate concurrency attacks and checkpoint techniques to recover program execution state (e.g., memory and files) from attacks.

2. Background of research, research plan and methodology:

(a maximum of seven A4 pages in total in Standard RGC Format for items (a) and (b))

(a) Background of research

(b) Research plan and methodology

Attached 7 pages(s) as follows

1 Research Background

This section introduces the background of concurrency bugs and their consequence on concurrency attacks (§1.1), presents others' related work (§1.2), and then introduces the PI's related work (§1.3).

1.1 Concurrency Bugs and Concurrency Attacks

Real-world multithreaded programs are plagued with various types of concurrency bugs [49], including data races, atomicity violations, deadlocks, and order violations. These bugs can easily lead to severe program behaviors such as crashes, hangs, and wrong outputs.

Worse, recent research [74] on 46 real-world concurrency bugs from CVE [2] and open source software's bug databases reveals that hackers have leveraged concurrency bugs to construct *concurrency attacks*, including corrupting critical memory [1], injecting malicious code [44], and escalating privilege [5]. These 46 attacks affect a wide range of multithreaded programs, ranging from applications such as KDE, Internet Explorer, Firefox, and Google Chrome, to OS kernels such as Windows, MacOS X, Linux, and iPhone OS.

Unfortunately, most existing defense techniques are weakened or completely bypassed by concurrency attacks because these techniques are mainly designed for single-threaded programs. For example, taint-tracking [30, 51, 52, 56], a runtime technique that tracks sensitive input data by associating security tags as metadata for each data byte, is weakened when running with multithreading programs because a race on data can corrupt the tags. §1.2 will discuss how concurrency attacks weaken various defense techniques in detail.

1.1.1 Our Preliminary Study on Concurrency Attacks

To guide the development of this Falcon project, we have studied another 23 real-world concurrency bugs and their consequence on concurrency attacks. We have written scripts and constructed inputs to reproduce all these 23 concurrency bugs: 14 bugs caused Denial of Service (DoS) attacks in application programs (Apache, MySQL, Tomcat, and PgSQL) by causing segmentation faults, three bugs caused heap overflows in applications programs (Apache and htmlCleaner), five bugs caused privilege escalations in MySQL or Linux kernels, and one bug bypassed security checks in a security library Libsafe and led to stack overflows. §2.1 will give two concrete bug examples.

We have two new discoveries in this study. First, many concurrency bugs can be triggered and exploited easily with crafted inputs. For instance, in a MySQL concurrency attack [4], we chose nested `select` statements to trigger the bug and corrupted another MySQL user's privilege table by only 18 repeated executions. In a Linux kernel exploit [5], we easily got root privilege with crafted systems call arguments. Second, although many concurrency bugs we studied were already fixed by upgrading program code or executable, the programs may still be in danger because hackers may have already broken in. Therefore, studying existing known concurrency bugs is still crucial.

1.1.2 Motivation of Objectives in this Proposal

To the best of our knowledge, state-of-the-art has no clue on handling concurrency attacks due to three major difficulties. First, people may still consider concurrency bugs and attacks as orthogonal behaviors because they lack a general, rigorous model to describe how concurrency bugs propagate to vulnerable instructions. Second, detecting a concurrency bug requires a systematic approach to carefully choose relevant inputs and thread interleavings, and to precisely identify vulnerable instructions in program code. Third, defending against concurrency attacks at runtime requires a reliable defense infrastructure. This proposal addresses each difficulty with an objective.

1.2 Related Work by Others

Reliability tools. There are various prior works on concurrency error detection [31, 47, 48, 57, 76, 79], diagnosis [54, 55, 60], and correction [38, 39, 68, 70]. These works mainly focus on the concurrency bugs themselves, and our proposed objectives focus on the security consequence of concurrency bugs. Therefore, these works are complementary to our objectives.

TOCTTOU attacks. TOCTTOU (time-of-check-to-time-of-use) attacks [12, 64, 65, 69] target mainly the file system interface, and leverage the non-atomic nature on time-of-check (e.g., `access()`) to time-of-use (e.g., `open()`) of a file to gain illegal file access. The concurrency attacks we target in this proposal are much more challenging than TOCTTOU attacks for two main reasons. First, TOCTTOU mainly causes illegal file access, while concurrency attacks can cause a much broader range of security vulnerabilities, ranging from gaining root privileges [5], injecting malicious code [5], to corrupting critical memory [1]. Second, concurrency attacks are mainly triggered by global memory access, much more difficult to track than the file access in TOCTTOU attacks.

Security defense techniques. Defense techniques for single-threaded programs have been well studied, including meta-data tracking [30, 51, 52, 56], anomaly detection [29, 59], address space randomization [62], and static analysis [11, 16, 32, 36, 67]. However, with the presence of multi-threading, these tools can be largely weakened. For instance, concurrency bugs in global memory may corrupt meta data tags in metadata tracking techniques. Anomaly detection lacks a concurrency model to reason about concurrency bugs and attacks. Static analysis has shown to find numerous bugs in real-world programs, however, when facing a program's huge amount of thread interleavings, static analysis usually reports too many false positives and bury the real ones.

Symbolic execution. Symbolic execution is an advanced program analysis technique that can systematically explore a program's execution paths to find bugs. Researchers have built scalable and effective symbolic execution systems to detect errors [15–17, 19, 33–35, 61, 73], block malicious inputs [20], and preserve privacy in error reports [18]. **Objective 2** will leverage symbolic execution to automatically find inputs that may trigger concurrency bugs and attacks.

Program slicing. Program slicing [63] is a general technique to prune irrelevant statements from a program or trace. Recently, systems researchers have leveraged or invented slicing techniques to block malicious input [20], synthesize executions for better error diagnosis [78], infer source code paths from log messages for postmortem analysis [77], and identify critical inter-thread reads that may lead to concurrency errors [80]. **Objective 2** will leverage program slicing to track the propagation from bugs to attacks.

State machine replication (SMR). SMR is a powerful fault-tolerant technique in distributed systems [41, 58]. SMR runs the same program on a set of physical machines (replicas), and it uses the Paxos [40, 42, 66] distributed consensus protocol to ensure consistent inputs for replicas as long as a majority of replicas is working correctly (i.e., even if minor replicas fail, SMR still guarantees availability of a program). **Objective 3** will leverage SMR to build a defense infrastructure.

1.3 Related Work by the PI

The PI is an expert on program analysis algorithms and tools [24, 27, 71], reliable and secure multithreading runtime systems [22, 23, 25, 75], and fault-tolerant distributed systems [26]. The PI's works are published in premier conferences on systems software (OSDI 2010, SOSP 2011, SOSP 2013, and SOSP 2015) and programming languages (PLDI 2012 and ASPLOS 2013). In connection to this project, the PI has developed a precise program slicing algorithm (a key technique for **Objective 2**) and two reliable runtime systems (key techniques for **Objective 3**).

2 Research Plan and Methodology

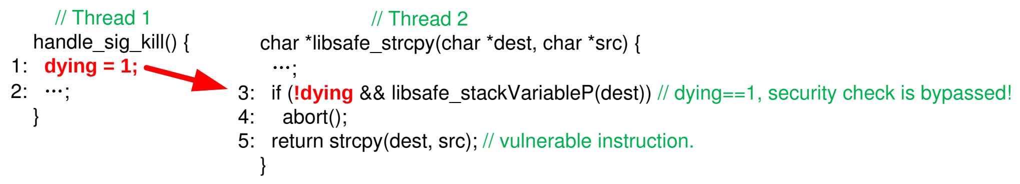
This Falcon project tackles concurrency attacks with a thorough, systematic methodology. To this end, this section presents three objectives, including a general, rigorous concurrency attack model (§2.1), a systematic concurrency attack detection approach (§2.2) by following this model, and a runtime defense infrastructure (§2.3). Each objective includes our preliminary results. Finally, this section describes our research plan (§2.4).

2.1 Objective 1: Modeling Concurrency Attacks

State-of-the-art lacks a general, rigorous model to describe how concurrency attacks manifest. In this section, §2.1.1 gives two concurrency attack examples in our preliminary study (§1.1.1), and then §2.1.2 proposes our model based on this study.

2.1.1 Concurrency Attack Examples in Our Preliminary Study

Figure 1 shows the code of a concurrency attack in Libsafe [43], a popular stack overflow protection library. Libsafe provides its own safe memory and string operations functions. For instance, the `libsafe_strcpy()` function checks whether a stack variable is passed in as the function argument `dest` before it calls the actual `strcpy()`. If a program receives a kill signal, Libsafe's internal thread (Thread 1) sets a global variable `dying`, then security checks are disabled in `libsafe_strcpy()`. Unfortunately, `dying` was not protected by mutex locks. In our study, we triggered a data race on this variable, bypassed the security check, and overflowed Thread 2's stack by passing malicious code to `dest`. Ironically, this Libsafe library is no longer "safe" when facing concurrency attacks.

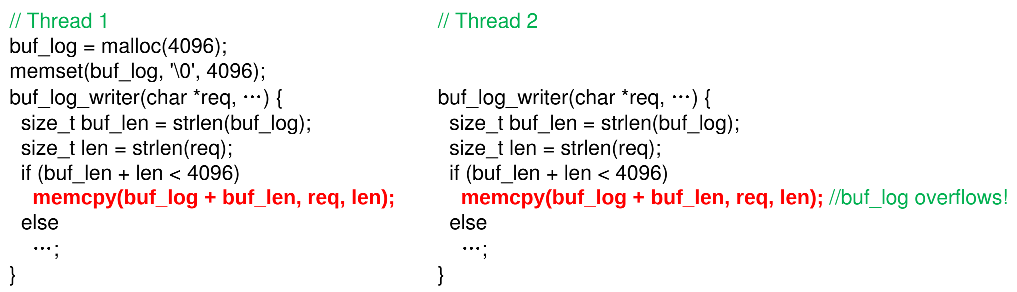


```
   // Thread 1
  handle_sig_kill() {
1:  dying = 1;
2:  …;
  }
```

```
     // Thread 2
   char *libsafe_strcpy(char *dest, char *src) {
     …;
3:   if (!dying && libsafe_stackVariableP(dest)) // dying==1, security check is bypassed!
4:     abort();
5:   return strcpy(dest, src); // vulnerable instruction.
   }
```

Figure 1: A concurrency attack in the Libsafe security library.

Figure 2 shows the code of a concurrency attack in the widely used Apache [7] web server for maintaining users' HTTP pages. Apache spawns a number of threads, each serves a HTTP request. Each thread also records the request in a global log buffer in heap and flushes this buffer to a log file when the buffer is full. However, developers missed a mutex lock to protect this buffer, so the buffer can overflow and corrupt adjacent memory, including the log file's descriptor. In our study, we corrupted this descriptor and made Apache write a request record to mess up another user's HTTP page.



```
// Thread 1
buf_log = malloc(4096);
memset(buf_log, '\0', 4096);
buf_log_writer(char *req, …) {
  size_t buf_len = strlen(buf_log);
  size_t len = strlen(req);
  if (buf_len + len < 4096)
    memcpy(buf_log + buf_len, req, len);
  else
    …;
}
```

```
// Thread 2


buf_log_writer(char *req, …) {
  size_t buf_len = strlen(buf_log);
  size_t len = strlen(req);
  if (buf_len + len < 4096)
    memcpy(buf_log + buf_len, req, len); //buf_log overflows!
  else
    …;
}
```

Figure 2: A concurrency attack in the Apache web server.

2.1.2 Preliminary Concurrency Attack Model

Two key questions on modeling concurrency attacks are: (1) what are the general elements in concurrency attacks, and (2) how does the corrupted memory in a concurrency bug propagate to vulnerable instructions in source code? Since existing reliability tools (§1.2) are already effective on detecting concurrency bugs, if our model can address these two questions, we can build software tools to effectively identify which concurrency bugs have the potential to lead to attacks, and we can safely ignore those don't.

Although the two aforementioned examples appear to be diverse, we have identified three general elements in these concurrency attacks. First, it is necessary to have corrupted global memory (e.g., the `dying` variable in Libsafe, and the `buf_len` buffer in Apache) caused by concurrency bugs. Second, it is necessary to have a vulnerable instruction (e.g., `strcpy()` or `setuid()`) to perform an attack. Third, the corrupted values in global memory must cause abnormal behavior on the vulnerable instructions. For example, the abnormal behavior in Libsafe is to leverage the corrupted memory to bypass the stack overflow check, an abnormal control-flow of the execution. For Apache, we directly corrupt (i.e., overflow) the buffer log, an abnormal data-flow of the execution.

In addition to these two examples, the 46 concurrency attacks in recent study [74] and the 23 ones our own preliminary study (§2.1.1) also have the three general elements. For the first element, these attacks have four types of concurrency bugs, including data races, atomicity violations, deadlocks, and order violations. For the second element, we have collected five types of vulnerable instructions, including memory operations (e.g., `strcpy()`), pointer deferences (e.g., when the pointer is NULL), privilege operations (e.g., `setuid()`), file operations (e.g., `access()`), and miscellaneous operations (e.g., `eval()` in shell scripts). For the third element, §2.2 will propose our systematic detection approach to detect abnormal control-flow and data-flow of executions.

Future work. Above model is based on our preliminary study on 23 concurrency attacks (§1.1.1). We will further study 10 times more concurrency attacks and refine our model by asking ourselves these research questions: (1) Are there other general elements or other types of vulnerable instructions? (2) Is there malicious data propagation among vulnerable instructions? (3) How does a concurrency attack in a program propagate to other programs through network or file system?

2.2 Objective 2: Detecting Concurrency Attacks

Our proposed detection approach is not only useful for developers to detect concurrency attacks during software testing phase, but also useful for analyzing concurrency bugs and their potential attacks in old releases of software programs. The reason is because even if concurrency bugs were already fixed in the latest software release, if concurrency attacks have occurred and hackers have broken in, simply fixing the bugs can not recover from the attacks (§2.1.1).

2.2.1 Workflow of Falcon's Detection Approach

We identify two major goals for our detection approach. First, a report should contain preconditions on program inputs so that developers can easily reproduce the concurrency attack. This goal is especially useful for developers to trigger attacks on rare inputs (e.g., nested `select` statements are required to trigger a concurrency attack to MySQL in our preliminary study study (§2.1.1).

Second, this approach should be precise in terms of having as few as false positives (reports but actually not a feasible attack) and false negatives (missing a real attack). To this end, this approach should capture only the data-flow and control-flow relevant for the propagation from corrupted memory to vulnerable instructions, and it should discard the irrelevant ones.

We plan to achieve the first goal with *symbolic execution* (§1.2), a program analysis technique that systematically explore a program's execution paths to find bugs. Unlike normal execution which runs on a concrete input, this technique marks inputs (e.g., command line arguments) as "symbolic" and tracks input preconditions on branch instructions (e.g., `if` and `while`) during the execution. If a branch instruction depends on inputs, symbolic execution forks a new execution, let the two executions go down different paths, and tracks input preconditions on each execution. This technique has shown to find new bugs in real-world systems programs [16].

We plan to achieve the second goal with *path slicing* (§1.2). Given a trace of executed instructions, path slicing starts from this essential instruction, goes backward the trace, and computes a subset of instructions that affect the reachability and operand values of the last instruction in the trace. This path slicing technique has been used to compute precise input preconditions to block malicious inputs [20].

Figure 3 shows the workflow of our proposed detection approach with three basic building blocks (shaded). This approach takes program source code as input. Specifically, we compile the code into LLVM [46] instructions, a popular form of instructions for program analysis tools. Symbolic execution marks a program input and bytes received from network as symbolic and explore program paths. For each execution, we collect all the the executed instructions as a trace, and run a concurrency bug detector on the trace. If a concurrency bug is detected, we check whether the trace contains a vulnerable instruction: if yes, we report a concurrency attack; if no, we feed the trace to the path slicer to collect branch instructions that are relevant to reach vulnerable instructions.

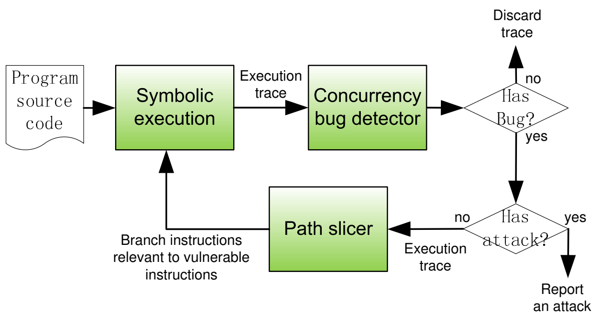


Figure 3: Workflow of detection approach. Three basic building blocks are shaded.

Algorithm 1: Path slicing algorithm

```
Input  : execution trace $trace$, concurrency bug instructions $bug$
Output: branch statements relevant to concurrency attacks
Slicing($trace$)
    $slice \leftarrow$ empty sequence
    for $i \in$ reverse($trace$) do
        if $i \in bug$ then
            break
        if any $e \in slice$ transitively depends on $i$ then
            $slice$.push_front($i$)
        else if $i$ is a branch instruction then
            for $j \in$ instructions $i$ reaches off $trace$ do
                if mayBeVulnerable($j$) then
                    $slice$.push_front($i$)
                    break
    return $slice$.branches()
```

Algorithm 1 shows the algorithm of the path slicer, our core technique to track the abnormal propagation (§2.1.2) from a concurrency bug to vulnerable instruction. This algorithm goes backward from the last execution of the trace and collects branch instructions relevant to a vulnerable instruction. To do so, `mayBeVulnerable()` does static analysis (§1.2) on the not-executed branches to detect vulnerable instructions. The path slicer then asks symbolic execution to explore these branch instructions with high priority so that we can increase the probability of hitting vulnerable instructions in following explorations.

Preliminary results. We have presented a preliminary path slicer (Algorithm 1) in ASPLOS 2013 [24] for detecting security bugs on one type of vulnerable instructions: privilege operations (e.g., `setuid()`). This path slicer has detected seven new security bugs in widely used programs.

Future work. We foresee three open challenges for our detection approach. First, how to generalize the path slicer to support all the five types of vulnerable instructions we collected (§2.1.2)?

Second, given the same vulnerable instruction, how to enhance the path slicer to smartly identify the vulnerable instructions among the safe ones (e.g., some `strcpy()` functions are safe if they only manipulate memory and do not involve program inputs)? Third, how to guide symbolic execution to explore malicious inputs and thread interleavings that are more likely to trigger concurrency bugs? Addressing these challenges will cultivate new techniques.

2.3 Objective 3: Defending against Concurrency Attacks

Our detection approach proposed in §2.2 is not designed to catch 100% concurrency attacks because exploring 100% program paths in symbolic execution is still an open research problem. Therefore, to practically prevent concurrency attacks taking over multi-threaded programs during the software deployment phase, this project proposes a runtime defense infrastructure.

Since a defense infrastructure requires tolerating failures of a program, this project plans to leverage *state machine replication (or SMR)*, a powerful fault-tolerance concept in distributed systems. SMR models a program as a deterministic state machine, where the states are important program data and the transitions are deterministic executions of program code under input requests. SMR runs replicas of the program on different physical computers and invokes a distributed consensus protocol (typically PAXOS [40, 42]) to ensure the same input sequence for replicas, as long as a majority of the replicas agrees on the sequence. Under the deterministic execution assumption, this majority of replicas must reach the same exact state despite various exceptions such as program failures. SMR is proven safe in theory [42, 66] and provides high availability in practice [6, 14].

Although SMR's fault-tolerant benefit makes it particularly attractive on building a general runtime infrastructure for defending against concurrency attacks, we foresee four research challenges to realize this infrastructure. First, to leverage existing SMR systems such as ZooKeeper [6], program developers have to shoehorn their programs into the narrowly defined state machine interfaces (e.g., ZooKeeper provides a file IO interface), a time-consuming and error-prone process.

Second, as introduced in §1.1, even given the same input, multithreaded programs can easily run into too many different thread interleavings (or *schedules*) at runtime depending on inter-thread communications (e.g., `pthread_mutex_lock`). Therefore, even if concurrency attacks do not manifest, different schedules can still easily cause replicas to diverge and no longer reach consensus.

Third, if all replicas run into the same buggy schedules and trigger concurrency attacks, our infrastructure should automatically recover the replicas and make them switch to another schedules.

Fourth, in addition to regular synchronizations, our infrastructure should also handle various common ad-hoc synchronizations (e.g., to enforce an execution order, one thread does busy wait until a global variable is true, and another thread sets the variable to be true).

2.3.1 Architecture of FALCON's Runtime Defense Infrastructure

Figure 4 shows our proposed SMR-based infrastructure. This infrastructure addresses the first challenge by enabling a general multi-threaded program run in it *transparently* without modifying this program. The major benefit of transparency is that program developers can focus on implementing their program's functionality, not how to defend against concurrency attacks.

To achieve transparency, this infrastructure interposes on the socket and the thread synchronization interfaces to keep replicas in sync. Specifically, it considers each incoming socket call (e.g., `recv()` a client's data) an input, and runs a PAXOS distributed consensus protocol [50] to ensure the same sequence of clients' socket calls for replicas. For efficiency, PAXOS assigns one replica as the primary to request input consensus; the other replicas are backup and they agree on the requests.

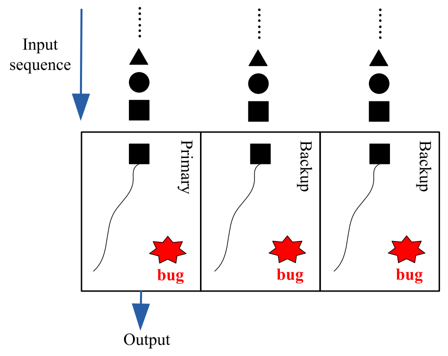


Figure 4: Defense infrastructure. Solid shapes are inputs; curve lines are schedules.

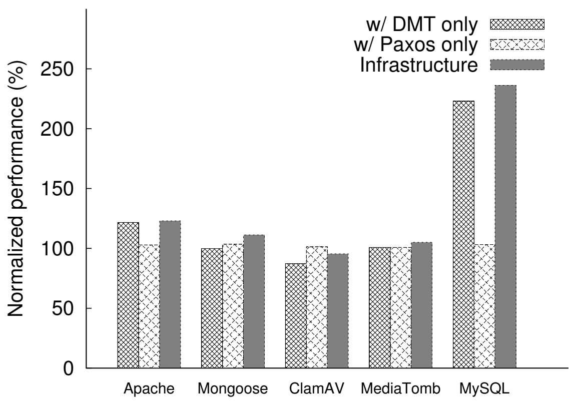


Figure 5: The infrastructure's performance normalized to programs' native execution.

To address the second challenge, this infrastructure schedules synchronizations using *deterministic multithreading (DMT)* [8–10, 13, 28, 37, 53]. This technique typically maintains a *logical time*, that advances deterministically on each thread's synchronization. By serializing thread synchronizations with a specific policy (e.g., round-robin or first-come-first-serve), DMT always enforces the same schedule on the same input (see Figure 4). The overhead of DMT is typically moderate because most program code is not synchronization and can still run in parallel.

To address the third challenge, we plan to leverage recent Linux process checkpoint/restore techniques, including process state (e.g., memory) [21] and file system state [3]. Our each checkpoint operation will be done on a backup replica (see Figure 4) so that it will not affect the other replicas to efficiently reach consensus on inputs and send responses. To handle all replicas running into the same buggy schedules (buggy schedules are extremely rare compared to correct schedules [49]), we will develop an "observer" program that runs on a dedicated machine to periodically check the status of replicas. If a majority of replicas fails, the observer restarts all replicas, restores program state on replicas with the latest checkpoint, and enforces a different DMT policy.

To address the fourth challenge, we plan to leverage recent advances on automatically annotating ad-hoc synchronizations [38, 72], and then we leverage a DMT scheduler (e.g., [25, 45]) to schedule these ad-hoc synchronizations in a deterministic order.

Preliminary results. We have developed preliminary systems to address the first and the second challenges. For the first challenge, we have presented CRANE [26], a transparent SMR system, in SOSP 2015. CRANE can support widely used server programs without modifying these programs, and it incurred 34.2% overhead in average for these programs (see Figure 5). For the second challenge, we have presented PARROT [25], an efficient DMT runtime, in SOSP 2013. PARROT incurred merely 12.7% overhead on a wide range of 108 popular multithreaded programs.

2.4 Research Plan

This project will require two PhD students S1 and S2 to work for three years. In the first year, S1 will develop and refine the concurrency attack model (part of **Objective 1**), and S2 will leverage the model to design the detailed workflow of the detection approach (part of **Objective 2**) by working closely with S1. In the second year, S1 will do an empirical study on how well the model represents real-world concurrency attacks (part of **Objective 1**), and S2 will implement the detection approach as a software tool (part of **Objective 2**). In the third year, S1 will implement the defense infrastructure (**Objective 3**), and S2 will study the detection tool on a broad range of real-world multithreaded programs to find new attacks.

(c) A maximum of two non-text pages of attached diagrams, photos, charts and table etc, if any.

(d) Reference (a maximum of three pages for references is allowed for listing the publications cited in Sections 1-2. All full references should be provided, including all authors of each reference.)

References

[1] Apache bug 25520. https://bz.apache.org/bugzilla/show_bug.cgi?id=25520.

[2] CVE: CWE-362. http://cvedetails.com/vulnerability-list/cweid-362/vulnerabilities.html.

[3] LXC. https://linuxcontainers.org/.

[4] Mysql bug 14747. https://bugs.mysql.com/bug.php?id=14747.

[5] Linux kernel bug on uselib(). http://osvdb.org/show/osvdb/12791.

[6] ZooKeeper. https://zookeeper.apache.org/.

[7] Apache. Apache web server. http://www.apache.org, 2012.

[8] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman. CoreDet: a compiler and runtime system for deterministic multithreaded execution. In *Fifteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '10)*, pages 53–64, Mar. 2010.

[9] T. Bergan, N. Hunt, L. Ceze, and S. D. Gribble. Deterministic process groups in dOS. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[10] T. Bergan, L. Ceze, and D. Grossman. Input-covering schedules for multithreaded programs. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, pages 677–692. ACM, 2013.

[11] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM*, 53:66–75, Feb. 2010.

[12] M. Bishop and M. Dilger. Checking for race conditions in file accesses. *Computing systems*, pages 131–152, Spring 1996.

[13] R. L. Bocchino, Jr., V. S. Adve, D. Dig, S. V. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian. A type and effect system for deterministic parallel java. In *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '09)*, pages 97–116, Oct. 2009.

[14] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 335–350, 2006.

[15] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. EXE: automatically generating inputs of death. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS '06)*, pages 322–335, Oct.–Nov. 2006.

[16] C. Cadar, D. Dunbar, and D. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the Eighth Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 209–224, Dec. 2008.

[17] G. Candea, S. Bucur, and C. Zamfir. Automated software testing as a service. In *Proceedings of the 1st Symposium on Cloud Computing (SOCC '10)*, 2010.

[18] M. Castro, M. Costa, and J.-P. Martin. Better bug reporting with better privacy. In *Thirteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '08)*, pages 319–328, Mar. 2008.

[19] V. Chipounov, V. Georgescu, C. Zamfir, and G. Candea. Selective Symbolic Execution. In *Fifth Workshop on Hot Topics in System Dependability (HotDep '09)*, 2009.

[20] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado. Bouncer: securing software by blocking bad input. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 117–130, Oct. 2007.

[21] criu. Criu. http://criu.org, 2015.

[22] H. Cui, J. Wu, C.-C. Tsai, and J. Yang. Stable deterministic multithreading through schedule memoization. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[23] H. Cui, J. Wu, J. Gallagher, H. Guo, and J. Yang. Efficient deterministic multithreading through schedule relaxation. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 337–351, Oct. 2011.

[24] H. Cui, G. Hu, J. Wu, and J. Yang. Verifying systems rules using rule-directed symbolic execution. In *Eighteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '13)*, 2013.

[25] H. Cui, J. Simsa, Y.-H. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. A. Gibson, and R. E. Bryant. Parrot: a practical runtime for deterministic, stable, and reliable threads. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Nov. 2013.

[26] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[27] H. Cui, R. Gu, C. Liu, and J. Yang. Repframe: An efficient and transparent framework for dynamic program analysis. In *Proceedings of 6th Asia-Pacific Workshop on Systems (APSys '15)*, July 2015.

[28] J. Devietti, B. Lucia, L. Ceze, and M. Oskin. DMP: deterministic shared memory multiprocessing. In *Fourteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '09)*, pages 85–96, Mar. 2009.

[29] A. Dinning and E. Schonberg. An empirical comparison of monitoring algorithms for access anomaly detection. In *Proceedings of the 2nd Symposium on Principles and Practice of Parallel Programming (PPOPP '90)*, pages 1–10, Mar. 1990.

[30] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, pages 1–6, 2010.

[31] D. Engler and K. Ashcraft. RacerX: effective, static detection of race conditions and deadlocks. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 237–252, Oct. 2003.

[32] D. Engler and M. Musuvathi. Static analysis versus software model checking for bug finding. In *Invited paper: Fifth International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI04)*, pages 191–210, Jan. 2004.

[33] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI '05)*, pages 213–223, June 2005.

[34] P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based whitebox fuzzing. In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, pages 206–215, 2008.

[35] P. Godefroid, M. Levin, and D. Molnar. Automated whitebox fuzz testing. In *NDSS '08: Proceedings of 15th Network and Distributed System Security Symposium*, Feb. 2008.

[36] S. Hallem, B. Chelf, Y. Xie, and D. Engler. A system and language for building system-specific, static analyses. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (PLDI '02)*, June 2002.

[37] N. Hunt, T. Bergan, , L. Ceze, and S. Gribble. DDOS: Taming nondeterminism in distributed systems. In *Eighteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '13)*, pages 499–508, 2013.

[38] G. Jin, W. Zhang, D. Deng, B. Liblit, and S. Lu. Automated concurrency-bug fixing. In *Proceedings of the Tenth Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 221–236, 2012.

[39] H. Jula, D. Tralamazza, Z. Cristian, and C. George. Deadlock immunity: Enabling systems to defend against deadlocks. In *Proceedings of the Eighth Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 295–308, Dec. 2008.

[40] L. Lamport. Paxos made simple. `http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf`.

[41] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, 1978.

[42] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[43] Libsafe. Libsafe. `http://directory.fsf.org/wiki/Libsafe`. URL `http://directory.fsf.org/wiki/Libsafe`.

[44] libsafe:bug. Libsafe - Safety Check Bypass Vulnerability. `http://www.securityfocus.com/archive/1/395999`. URL `http://www.securityfocus.com/archive/1/395999`.

[45] T. Liu, C. Curtsinger, and E. D. Berger. DTHREADS: efficient deterministic multithreading. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 327–336, Oct. 2011.

[46] LLVM. The LLVM compiler framework. `http://llvm.org`, 2013.

[47] S. Lu, J. Tucek, F. Qin, and Y. Zhou. AVIO: detecting atomicity violations via access interleaving invariants. In *Twelfth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '06)*, pages 37–48, Oct. 2006.

[48] S. Lu, S. Park, C. Hu, X. Ma, W. Jiang, Z. Li, R. A. Popa, and Y. Zhou. Muvi: automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 103–116, 2007.

[49] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Thirteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '08)*, pages 329–339, Mar. 2008.

[50] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www. scs. stanford. edu/dm/home/papers, 2007.

[51] A. Myers and B. Liskov. A decentralized model for information flow control. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, pages 129–142, Oct. 1997.

[52] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI '07)*, pages 89–100, June 2007.

[53] M. Olszewski, J. Ansel, and S. Amarasinghe. Kendo: efficient deterministic multithreading in software. In *Fourteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '09)*, pages 97–108, Mar. 2009.

[54] C.-S. Park and K. Sen. Randomized active atomicity violation detection in concurrent programs. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*, pages 135–145, Nov. 2008.

[55] S. Park, S. Lu, and Y. Zhou. CTrigger: exposing atomicity violation bugs from their hiding places. In *Fourteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '09)*, pages 25–36, Mar. 2009.

[56] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 135–148, 2006.

[57] S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. E. Anderson. Eraser: A dynamic data race detector for multithreaded programming. *ACM Transactions on Computer Systems*, pages 391–411, Nov. 1997.

[58] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[59] D. Schonberg. On-the-fly detection of access anomalies. In *Proceedings of the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, pages 285–297, 1989.

[60] K. Sen. Race directed random testing of concurrent programs. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI '08)*, pages 11–21, June 2008.

[61] K. Sen, D. Marinov, and G. Agha. CUTE: A concolic unit testing engine for C. In *Proceedings of the 10th European Software Engineering Conference held jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13)*, pages 263–272, Sept. 2005.

[62] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, Proceedings of the 11th ACM conference on Computer and communications security (CCS '04), pages 298–307, 2004.

[63] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages 3(3)*, pages 121–189, 1995.

[64] D. Tsafrir, T. Hertz, D. Wagner, and D. Da Silva. Portably solving file tocttou races with hardness amplification. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 13:1–13:18, 2008.

[65] E. Tsyrklevich and B. Yee. Dynamic detection and prevention of race conditions in file accesses. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, pages 17–17, 2003.

[66] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Computing Surveys (CSUR)*, 47(3):42:1–42:36, 2015.

[67] D. Wagner and D. Dean. Intrusion detection via static analysis. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 156, 2001.

[68] Y. Wang, T. Kelly, M. Kudlur, S. Lafortune, and S. Mahlke. Gadara: Dynamic deadlock avoidance for multithreaded programs. In *Proceedings of the Eighth Symposium on Operating Systems Design and Implementation (OSDI '08)*, pages 281–294, Dec. 2008.

[69] J. Wei and C. Pu. Tocttou vulnerabilities in unix-style file systems: an anatomical study. In *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies - Volume 4*, pages 12–12, 2005.

[70] J. Wu, H. Cui, and J. Yang. Bypassing races in live applications with execution filters. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[71] J. Wu, Y. Tang, G. Hu, H. Cui, and J. Yang. Sound and precise analysis of parallel programs through schedule specialization. In *Proceedings of the ACM SIGPLAN 2012 Conference on Programming Language Design and Implementation (PLDI '12)*, pages 205–216, June 2012.

[72] W. Xiong, S. Park, J. Zhang, Y. Zhou, and Z. Ma. Ad hoc synchronization considered harmful. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[73] J. Yang, C. Sar, P. Twohey, C. Cadar, and D. Engler. Automatically generating malicious disks using symbolic execution. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P '06)*, pages 243–257, May 2006.

[74] J. Yang, A. Cui, S. Stolfo, and S. Sethumadhavan. Concurrency attacks. In *the Fourth USENIX Workshop on Hot Topics in Parallelism (HOTPAR '12)*, June 2012.

[75] J. Yang, H. Cui, J. Wu, Y. Tang, and G. Hu. Determinism is not enough: Making parallel programs reliable with stable multithreading. *Communications of the ACM*, 2014.

[76] Y. Yu, T. Rodeheffer, and W. Chen. RaceTrack: efficient detection of data race conditions via adaptive tracking. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 221–234, Oct. 2005.

[77] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy. SherLog: error diagnosis by connecting clues from run-time logs. In *Fifteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '10)*, pages 143–154, Mar. 2010.

[78] C. Zamfir and G. Candea. Execution synthesis: a technique for automated software debugging. In *Proceedings of the 2010 ACM European Conference on Computer Systems (EUROSYS '10)*, pages 321–334, Apr. 2010.

[79] W. Zhang, C. Sun, and S. Lu. ConMem: detecting severe concurrency bugs through an effect-oriented approach. In *Fifteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '10)*, pages 179–192, Mar. 2010.

[80] W. Zhang, J. Lim, R. Olichandran, J. Scherpelz, G. Jin, S. Lu, and T. Reps. ConSeq: detecting concurrency bugs through sequential errors. In *Sixteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '11)*, pages 251–264, Mar. 2011.

PROJECT FUNDING

3. Cost and justification

(a) Estimated cost and resource implications:
[Detailed justifications should be given in order to support the request for each item below]
(a maximum of 500 words for each box)

	Year 1 ($)	Year 2 ($)	Year 3 ($)	Year 4 ($)	Year 5 ($)	Total ($)
(A) One-line Vote Items						
(i) Supporting Staff Costs						
Types						
Monthly salary x Nos. x Months						
Research Postgraduate Students						$1,104,960
15,000 * 2 * 12	360,000					
15,520 * 2 * 12		372,480				
15,520 * 2 * 12			372,480			

Justification:

This project will require two PhD students S1 and S2 for a period of three years. In the first year, S1 will develop and refine the concurrency attack model (part of Objective 1), and S2 will leverage the model to design the detailed workflow of the detection approach (part of Objective 2) by working closely with S1. In the second year, S1 will do an empirical study on how well the model represents real-world concurrency attacks (part of Objective 1), and S2 will implement the detection approach as a software tool (part of Objective 2). In the third year, S1 will build the defense infrastructure (Objective 3), and S2 will study the detection tool on broad real-world programs to find new attacks.

(ii) Equipment Expenses
[please itemize and provide quotations for each item costing over $200,000]

Justification:

The PI does not need extra computing equipment for this project. The PI's research group at HKU has three personal computers, each has 12 CPU cores. The HKU CS department's labs have plenty of personal computers to do research. The HKU CS department also has a moderate funding for each teacher to buy new personal computers to do research. Overall, the PI feels that current facilities are sufficient to conduct this proposed project.

Quotation Provided: **Yes** ☐ [✓ **No**]

(iii) Outsourcing Expenses of Research Work Outside Hong Kong
[please itemize your cost estimation with justification and provide quotations for work costing over $200,000]

Justification:

Quotation Provided: **Yes** ☐ [✓ **No**]

(iv) General Expenses
[please itemize and provide quotations for services/purchase costing over $200,000]

Oversea visits.	20,000	20,000	20,000	0	0	60,000

Justification:

The PI plans to visit CMU, NYU, and Cornell during the summers to perform emperical study on this project. Each visit will take about one month.

Quotation Provided: **Yes** ☐ [✓ **No**]

(v) Conference Expenses

Conference costs, including registration, travelling, and accommodation.	20,000	20,000	20,000	0	0	60,000

Justification:

Support for conference registration fee, travelling expenses, and accommodations for attending conferences (e.g., SOSP, OSDI, PLDI, and ASPLOS) held outside Hong Kong.

Sub-total for (A) (One-line Vote Items): $ 1,224,960

(B) Earmarked Items

(vi) Costs for Employment of Relief Teacher
[see Enclosure I for relief support under Humanities and Social Sciences Panel]

Rank

Monthly salary x Months

Justification:

Current Average Teaching Load: Total 0 classroom hours per academic year [please report UGC-funded programmes only]

(vii) High-performance Computing Services Expenses

Justification:

Quotation Provided: **Yes** ☐ **No** ☑

(viii) Research-related Software Licence /Dataset
[Please itemize and provide quotations for each item]

Justification:

Sub-total for (B) (Earmarked Items):	**$**	0
(ix) Total cost of the proposal (A) + (B)	**$**	1,224,960

(C) Deduction Items

Less :		
(x) Other research funds secured from other sources	**$**	0
Sub-total for C :	**$**	0
(xi) Amount requested in this application : (A) + (B) - (C)	**$**	1,224,960

(D) Academic Research related to Public Policy Developments

(xii) Percentage of the total cost of the proposal related to public policy developments ((A) + (B)) **[see Enclosure II for Support for Academic Research relating to Public Policy Developments]**	0%

(b) Declaration on the Equipment Procurement:

☑ **(i) I declare that no equipment is required**

OR

☐ **(ii) I declare that the equipment indicated in 3(a)(A)(ii) above is not available in the institution**

OR

☐ **(iii) I declare that all or some of the equipment (please provide details in the following text box) indicated in Section 3(a)(A)(ii) above is available in the institution but cannot be used by me in** view of the following reasons (a maximum of 500 words)

Reasons : (a maximum of 500 words)

(c) Declaration on employment of relief teacher:

☑ (i) I declare that no relief teacher is required

OR

☐ (ii) I declare that I currently do not hold any grant for employment of relief teacher of any on-going project under UGC/RGC funding schemes (excluding Humanities and Social Sciences Prestigious Fellowhsip Scheme (HSSPFS))

OR

☐ (iii) I declare that I hold grant for employment of relief teacher of the following on-going project(s) under UGC/RGC funding schemes (excluding HSSPFS) and undertake to submit the corresponding completion report(s) by 30 April 2016

(d) Declaration on high-performance computing services:

☑ (i) I declare that no high-performance computing services is required

OR

☐ (ii) I declare that the high-performance computing services indicated in Section 3(a)(B)(vii) above is not available in the institution

OR

☐ (iii) I declare that all or some of the high-performance computing services (please provide details in the following text box) indicated in Section 3(a)(B)(vii) above is available in the institution but cannot be used by me in view of the following reasons(a maximum of 500 words)

Reasons : (a maximum of 500 words)

(e) Declaration on the research-related software licence / dataset

☑ (i) I declare that no research-related software licence / dataset is

required

OR

☐ **(ii) I declare that the research-related software licence / dataset indicated in Section 3(a)(B)(viii) above is not available in the institution**

OR

☐ **(iii) I declare that all or some of the research-related software licence / dataset (please provide details in the following text box) indicated in Section 3(a)(B)(viii) above is available in the institution but cannot be used by me in view of the following reasons (a maximum of 500 words)**

Reasons : (a maximum of 500 words)

4. Existing facilities and major equipment available for this research proposal:
(a maximum of 400 words)

The PI's research group at HKU has three personal computers, each has 12 CPU cores. The HKU CS department's labs have plenty of personal computers to do research. The HKU CS department also has a moderate funding for each teacher to buy new personal computers to do research. Overall, the PI feels that current facilities are sufficient to conduct this proposed project.

5. Funds secured or to be secured

(a) Other research funds already secured for this research proposal:
[This amount will be deducted from the total cost of the project in Section 3 of Part II above.]

Source	Amount ($)

(b) Other research funds to be or are being sought for this research proposal.
[If funds under this item are secured, the amount to be awarded may be reduced]:

Source	Amount ($)

DECLARATION OF SIMILAR OR RELATED PROPOSALS

[Please refer to ECS2 for information required and implications for non-disclosure of similar or related proposals]

6. Re-submission of a proposal not supported previously

(a) Is this proposal a re-submission or largely similar to a proposal that has been submitted to but not supported by the UGC/RGC or other funding agencies?

Yes ☐ [✓ No]

If yes, please state the funding agency(ies) and the funding programme(s):

Reference No(s).: [for UGC/RGC projects only]:

Project title(s) : [if different from Section 1(a) of Part I above.]

Date(month/year) of application:

Outcome:

(b) If this application is the same as or similar to the one(s) submitted but not supported previously, what were the main concerns / suggestions of the reviewers then?

(c) Please give a brief response to the points mentioned in Section 6(b) above, highlighting the major changes that have been incorporated in this application.

7. Submission of a new proposal or proposal similar or related to on-going and completed projects, and proposals pending funding approval

(a)(I) Is there similar research being carried out by the PI?

Yes ☑ **No** ☐

(II) Is there related research being carried out by the PI?

Yes ☑ **No** ☐

If yes to 7(a)(I) or (II), please give a brief account including names of investigators, departmental and institutional affiliations, project title(s) and nature of the project(s): (a maximum of 400 words)

Project code: 201505159004.
Project Title: OWL: Automatically Identifying and Avoiding Concurrency Attacks.
Funding Scheme: HKU Seed Funding Programme for Basic Research.
PI: Dr. Heming Cui.
Award Amount (HKD): $120,000.
Project Period (Start - End): 30/05/2015 - 29/05/2017.

The goal of this ongoing OWL project is to invent automated and effective tools to identify and avoid concurrency attacks by leveraging Stable Multithreading (or StableMT) [SOSP '13, CACM '14], a reliable multithreading runtime technique invented by my collaborators and me. StableMT can greatly reduce the number of possible thread interleavings in multithreaded programs, the root cause that makes multithreading difficult to get right. By addressing this root cause, StableMT makes multithreaded programs almost as simple as single-threaded programs, thus we can easily leverage existing defense techniques to identify and avoid concurrency attacks.

One major limitation of this ongoing OWL project is that, to enjoy StableMT's reliability benefit, OWL assumes that its tools all run in the StableMT runtime environment, which may incur high overhead on some multithreaded programs. To overcome this limitation, this FALCON proposal pursues three objectives to tackle concurrency attacks in general (traditional) multithreading runtimes.

(b)(I) Is/Are there similar proposal(s) being submitted by PI (in both capacity as PI/PC or Co-I/Co-PI) to other competitive funding schemes of the RGC or other funding agency(ies) or the PI's institution(s)?

Yes ☐ [✓ No]

(II) Is/Are there related proposal(s) being submitted by PI (in both capacity as PI/PC or Co-I/Co-PI) to other competitive funding schemes of the RGC or other funding agency(ies) or the PI's institution(s)?

Yes ☐ [✓ No]

If yes to Section7(b)(I) or (II), please give the following details -

(c)(I) Is/Are there similar project(s)/work by PI (in both capacity as PI/PC or Co-I/Co-PI) already completed?

Yes ☐ [✓ No]

(II) Is/Are there related project(s)/work by PI (in both capacity as PI/PC or Co-I/Co-PI) already completed?

Yes ☐ [✓ No]

If yes to 7(c)(I) or (II) , please give the following details -

Project(s) submitted by PI:

8. Particulars of PI

(a) Investigator information:
Name and Academic Affiliation of Applicant:

Name	Post	Unit/ Department/ Institution	Current Member of RGC Council/Subject Panel as at application deadline (Yes or No)	RGC Council/ Name of RGC Subject Panel

PI	Dr Cui, Heming	Assistant Professor	Department of Computer Science/The University of Hong Kong	No

(b) Curriculum vitae (CV) of Applicant.
Please attach a maximum of two A4 pages in standard RGC format for attaching PDF documents or (a maximum of 800 words for direct input in the text box) in the following format.]

i) Name:
ii) Academic qualifications:
iii) Previous academic positions held(with dates):
iv) Present academic position:
v) Previous relevant research work:
vi) Publication records [Please refer to ECS 2 Part II Section 8 for the format required by the RGC]:
vii) Others (please specify):

Heming Cui

Assistant Professor, Computer Science, HKU *heming@cs.hku.hk*

Academic Qualification

- PhD, Computer Science, Columbia University, New York, USA, 2014.
- Master, Computer Science, Tsinghua University, Beijing, China, 2008.
- Bachelor, Computer Science, Tsinghua University, Beijing, China, 2005.

Academic Position

- January 2015 ∼ now, Assistant Professor, Computer Science, University of Hong Kong.
- Website: `http://www.cs.hku.hk/~heming`

Relevant Research Experience

- Fault-tolerant distributed system: [Crane SOSP '15].
- Reliable multithreading: [Tern OSDI '10], [Peregrine SOSP '11], [Parrot SOSP '13].
- Security rule violation detection: [Woodpecker ASPLOS '13].
- Precise static data race detection: [PLDI '12].
- Dynamic instrumentation for bypassing concurrency errors: [Loom OSDI '10].

Selected Publication

- **Heming Cui**, Rui Gu, Cheng Liu, Tianyu Chen, and Junfeng Yang. "Paxos Made Transparent". Proceedings of the 25th ACM Symposium on Operating Systems Principles (**SOSP '15**).
- Junfeng Yang, **Heming Cui**, Jingyue Wu, Yang Tang, and Gang Hu. "Determinism Is Not Enough: Making Parallel Programs Reliable with Stable Multithreading". Communications of the ACM 2014 (**CACM '14**).
- **Heming Cui**, Jiri Simsa, Yi-Hong Lin, Hao Li, Ben Blum, Xinan Xu, Junfeng Yang, Garth Gibson, and Randy Bryant. "Parrot: a Practical Runtime for Deterministic, Stable, and Reliable Threads". Proceedings of the 24th ACM Symposium on Operating Systems Principles (**SOSP '13**).
- **Heming Cui**, Gang Hu, Jingyue Wu, and Junfeng Yang. "Verifying Systems Rules Using Rule-Directed Symbolic Execution". Proceedings of the 18th International Conference on Architecture Support for Programming Languages and Operating Systems (**ASPLOS '13**).
- Jingyue Wu, Yang Tang, Gang Hu, **Heming Cui**, Junfeng Yang . "Sound and Precise Analysis of Parallel Programs through Schedule Specialization". Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (**PLDI '12**).
- **Heming Cui**, Jingyue Wu, John Gallagher, Huayang Guo, and Junfeng Yang. "Efficient Deterministic Multithreading through Schedule Relaxation". Proceedings of the 23rd ACM Symposium on Operating Systems Principles (**SOSP '11**).
- **Heming Cui**, Jingyue Wu, Chia-che Tsai, and Junfeng Yang. "Stable Deterministic Multithreading through Schedule Memoization". Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (**OSDI '10**).

- Jingyue Wu, **Heming Cui**, and Junfeng Yang. "Bypassing Races in Live Applications with Execution Filters". Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (**OSDI '10**).

Academic Talk

- **Paxos Made Transparent**
  - Hong Kong University of Science and Technology, October 2015
    `https://www.cse.ust.hk/pg/seminars/F15/cui.html`
- **PARROT: A Practical Runtime for Deterministic, Stable, and Reliable Threads**
  - DARPA PI meeting, January 2014
  - Tsinghua University in Beijing, China, December 2013
  - Institute of Software Chinese Academy of Science (ISCAS), December 2013
    `http://www.iscas.ac.cn/xwzx/xshd/201311/t20131125_3985178.html`
  - SOSP conference, November 2013
  - Carnegie Mellon University, October 2013
    `http://www.pdl.cmu.edu/SDI/2013/103113-b.html`
- **Verifying Systems Rules Using Rule-Directed Symbolic Execution**
  - ASPLOS conference, March 2013
  - DARPA site visit, September 2012
- **Efficient Deterministic Multithreading through Schedule Relaxation**
  - DARPA site visit, September 2012
  - SOSP conference, October 2011
- **Stable Deterministic Multithreading through Schedule Memoization**
  - OSDI conference, October 2010

(c) Plan(s) for collaboration in this application:

[Indicate the role and the specific task(s) the PI and each Collaborator , if any, is responsible for.]

I plan to collaborate with researchers from CMU and Columbia University. I have been collaborating with Prof. Garth Gibson from CMU and Prof. Junfeng Yang from Columbia University. The PI and his students will conduct the main research role of this project, including developing theoretical models, building tools, and implementing software. The collaborators from CMU and Columbia will provide high-level suggestions on our project and apply our tools in real-world applications.

(d) Number of hours per week to be spent by the PI in the proposal: 20 hour(s)

GRANT RECORD OF PRINCIPAL INVESTIGATOR

9. Details of Research Projects

Details of research projects undertaken by the PI (in a PI/PC or Co-I/Co-PI capacity) including (a) completed research projects funded from all sources (irrespective whether from UGC/RGC) in the past five years; and (b) on-going research projects funded from all sources (irrespective whether from UGC/RGC); (c) terminated projects funded by UGC/RGC in the past five years; and (d) unsuccessful proposals or withdrawn projects submitted to UGC/RGC in the past five years.

Project Ref No.	Status of proposal/ project (Completed, On-going, Terminated, Unsuccessful, Withdrawn)	Project Title	PI/PC/Co-I/Co-PI	Funding Source(s) and Amount($)	UGC/RGC Funding (Yes or No)	Start Date (if applicable)	Estimated Completion Date (if applicable)	Number of Hours Per Week Spent by the PI in Each On-going Project*
201505159004	On-going	OWL: Automatically Identifying and Avoiding Concurrency Attacks	PI	HKU $ 120,000	N	30-05-2015	29-05-2017	10

27206015	Unsuccessful	CRANE: A Practical Fault-tolerant System For Replicating Execution States of Multi-threaded Programs	PI	ECS	$ 0	--	--	0

*** The PI is not required to report on the time spent in the capacity of Co-I in GRF projects.**

Please provide the objectives of each project listed in (a), (b) and (c) above.

Project Ref No.:	201505159004
Objective:	The goal of this ongoing OWL project is to invent automated and effective tools to identify and avoid concurrency attacks by leveraging Stable Multithreading (or StableMT) [SOSP '13, CACM '14], a reliable multithreading runtime technique invented by my collaborators and me. StableMT can greatly reduce the number of possible thread interleavings in multithreaded programs, the root cause that makes multithreading difficult to get right. By addressing this root cause, StableMT makes multithreaded programs almost as simple as single-threaded programs, thus we can easily leverage existing defense techniques to identify and avoid concurrency attacks. One major limitation of this ongoing OWL project is that, to enjoy StableMT's reliability benefit, OWL assumes that its tools all run in the StableMT runtime environment, which may incur high overhead on some multithreaded programs. To overcome this limitation, this FALCON proposal pursues three objectives to tackle concurrency attacks in general (traditional) multithreading runtimes.

ANCILLARY INFORMATION

10. Research Ethics/Safety Approval and Access to Government/ Official/ Private Data and Records

[Please refer to ECS2 Part II Section 10 for the responsibilities and implications]

(a) Research Ethics/Safety Approval

(i) I confirm that the research proposal ☐ **involves /** ☑ **does not involve human subjects.**

(ii) Please tick the appropriate boxes to confirm if approval for the respective ethics and/or safety issues is required and has been / is being obtained from the PI's institution. PIs are encouraged to seek necessary approval before application deadline as far as possible

	Approval not required	Approval obtained	Approval being sought
(1) Human research ethics	☑	☐	☐
(2) Animal research ethics	☑	☐	☐
(3) Biological safety	☑	☐	☐
(4) Ionizing radiation safety	☑	☐	☐
(5) Non-ionizing radiation safety	☑	☐	☐
(6) Chemical safety	☑	☐	☐

(iii) If approval is required by other authorities, please indicate *below* the names of the authorities and the prospects of obtaining such approval. If not applicable, please put down "N.A.".

N.A.

(b) Access to Government/ Official/ Private Data and Records

(i) Is access to Government / official / private data and records critical to the research proposal?

☐ **Yes**

☑ **No**

If approval is required, please indicate below the names of the agency(ies) of obtaining such approval.

(ii) Please tick in the appropriate boxes to confirm if approval for access to the related data/records has been / is being obtained from the relevant agency(ies). If approval has been obtained, please provide evidence.

List of agency(ies)	Approval not required	Approval obtained	Approval being sought

[Note: PIs are encouraged to seek necessary approval before application deadline as far as possible.]

11. Proposed reviewers

(a) List of proposed reviewers:

The PI assumes full responsibility for reporting all the relationship(s) between himself/herself with each of the nominated external reviewers. To avoid any possible or preceived conflict of interests, nomination of external reviewer(s) having any of the following relationship(s) with PI should be avoided

(i) Is currently employed / was employed from 14 November 2013 to 13 November 2015 by the institution of the PI;)

(ii) Holds adjunct, honorary or visiting position(s) in the institution of the PI;

(iii) Serves as consultant / advisor to a committee or department of the institution of the PI;

(iv) Submitted applications as Project Coordinator, Principal Investigator (PI), Co-PI, Co-Investigator or Collaborators in the same funding exercise;

(v) Has pre-reviewed the application;

(vi) Has advisor / advisee relationship (such as tutor and PhD student relationship) with the PI;

(vii) Has co-authorship of patents with the PI;

(viii) Has close personal relationship (e.g. partner, spouse, immediate family member, long-term close friend) with the PI;

(ix) Has co-authorship of paper or publications with the PI from 14 November 2012 to 13 November 2015;

(x) Is a collaborator (in the capacity of Co-PI or Co-I) in research projects or programmes held by the PI from 14 November 2012 to 13 November 2015; or

(xi) Serves the same editorial board with an appointor-appointee relationship.

[Please refer to Part II Section 11 of ECS2 for responsibilities and implications.]

(i) Title/Name/Post/Unit/Department/Institution：	Prof Chen, Haibo/Professor/School of Software/Shanghai Jiao Tong University
Address/Tel./Fax/E-mail：	China/haibochen@sjtu.edu.cn
Area of Expertise：	Concurrency, distributed systems, software security.
(ii) Title/Name/Post/Unit/Department/Institution：	Prof Lu, Shan/Associate Professor/Computer Science/University of Chicago
Address/Tel./Fax/E-mail：	U.S.A./shanlu@uchicago.edu
Area of Expertise：	Concurrency, software reliability, program analysis.

(iii) Title/Name/Post/Unit/Department/Institution:	Prof Chen, Wenguang/Professor/Computer Science/Tsinghua University
Address/Tel./Fax/E-mail:	China/cwg@tsinghua.edu.cn
Area of Expertise:	High-performance computing, concurrency, and reliability.

(b) Declaration of any past and present relationship between the PI and the nominated reviewers [minimum one tick per reviewer]:

Nature of relationship (please elaborate)	**Reviewer**				
	(i)	**(ii)**	**(iii)**	**(iv)**	**(v)**
Has co-authorship of paper or publications with the PI in the period from 14 November 2008 to 13 November 2012	☐	☐	☐	☐	☐
Was a collaborator (in the capacity of Co-PI or Co-I) in research projects or programmes held by the PI from 14 November 2008 to 13 November 2012	☐	☐	☐	☐	☐
Partnership / co-organisers of major events with the PI from 14 November 2008 to 13 November 2015	☐	☐	☐	☐	☐
Teacher at undergraduate studies	☐	☐	☐	☐	☐
Fellow members of the same editorial board	☐	☐	☐	☐	☐
Others (please specify)	☐	☐	☐	☐	☐
None	☑	☑	☑	☐	☐

(c) Indicate the name of PI and the nature of the relationship declared in (b) (e.g. when and where the relationship was / is developed, name / nature of project(s), publication(s) or event(s) involved):

(1) Nature of relationships to : Prof Chen, Haibo
N.A.

(2) Nature of relationships to : Prof Lu, Shan
N.A.

(3) Nature of relationships to : Prof Chen, Wenguang
N.A.

12. Release of completion report, data archive possibilities and public access of publications resulting from research funded by the RGC

(a) Is the proposed project likely to generate data set(s) of retention value?

Yes ☐ [✓ **No**]

If yes, please describe the nature, quantity and potential use of the data set(s) in future.

Nil

(b) Are you willing to make the data set(s) available to others for reference twelve months after the publication of research results or the completion of this proposed project?

Yes ☐ **No** ☐

I understand that the RGC will release the completion report to the public and only considers data archiving requests after the completion of the RGC-funded project. The RGC has full discretion in funding the archiving requests. Data sets archived with RGC funds will require users to acknowledge the originator and the RGC. The originator will also be provided with copies of all publications derived from the use of the data.

I undertake that upon acceptance of a paper for publication,

(i) I will check whether the publisher already allows (A) full open access to the publisher's version, or (B) my depositing a copy of the paper (either the publishers version or the final accepted manuscript after peer-review) in the institutional repository for open access;
(ii) if both (i) (A) and (B) are not allowed, I will request the publisher to allow me to place either version in my institutional repository for restricted access immediately upon publication or after an embargo period of up to twelve months if required by the publisher; and
(iii) subject to the publisher's agreement on (i) or (ii) above, I will deposit a copy of the publication in my institutional repository as early as possible but no later than six months after publication or the embargo period, if any.

13. Education Plan, Dissemination Plan and Supporting documents

(A maximum of 20 words for each box to caption each uploaded pdf documents)
[Note: To ensure fairness in the independent assessment process, PIs should not disclose his/her identity or institution in the Education Plan.]

Appendix 1: Education Plan (up to one A4 page)

Appendix 2：Dissemination Plan

Education Plan

Our proposed project will include training undergraduate students as future researchers. At the monthly rate of HKD $2,500 per undergraduate student, a maximum budget of HKD $100,000 can provide 40 man-months. On average, two graduate students per semester will be able to participate in this project. An undergraduate student can join our project in the following ways:

1. **Final-year project**. The PI will advise final-year undergradaute students, and a project could be: (1) studying the applicality of our tools and systems on a variety of multi-threaded programs, ranging from web servers, SQL servers, to key-value stores; (2) implementing eluation scripts that can automatically run our tools and systems with real-world programs and collect performance results; (3) in-depth analysis of performance results, including memory usage, CPU contention, file systems IO bottlenecks, etc. By conducing these hand-on tasks, undergraduate students can get good experience on understanding and using real-world software, performance diagnoses, and programming with scripting languages. We believe these are great ways to get the students interested in research and build a concrete sense on real-world software.
2. **Research seminar**. The PI will hold regular weekly research seminars with the postgraduate students as well as undergraduate students. Each seminar will either discuss on the design, implementation, and evaluation of a top systems paper, or will inivite external experts to present their softwrare systems or industrial experience. By participating these seminars, students can learn the whole process of doing research, implementing production software, and even paper writings.
3. **One-on-one meeting**. The PI will also have regular one-on-one meetings with each undergraduate student involved in the project. The PI will consult the progress, issues, and questions from each student and give concrete suggestions. The goal is to make sure the PI knows the performance of the research students well and make sure the students are on the right track.
4. **Student mentor system**. Each research postgraduate student will act as a mentor for an undergradaute student researcher. This could help undergradaute student easy to get into research and solve technical problems in research projects. The postgraduate students can also learn how to act as a mentor in this mentoring process.
5. **Case study**. A few undergradaute students with strong motivation and programming skills will be invited to apply existing top reliability and security tools into our runtime infrastructure. They will learn how to install a tool on one of our replicas, and how to inspect the results reported from these tools, and how to analyze the reliability and security implication of these results. This process can help undergradaute students to get familiar with various research fields broadly.

Dissemination Plan

Publication. We expect to publish our research work in top conferences and journals to receive maximum impact. The PI has published several papers in top conferences in two major research fields: (1) operating systems and distributed systems, including [CRANE SOSP '15], [PARROT SOSP '13], [PEREGRINE SOSP '11], [TERN OSDI '10], and [LOOM OSDI '10]; and (2) programming lanauges, including [WOODPECKER ASPLOS '13] and [Wu PLDI '12]. The PI has also published in a Computer Science flag-ship journal: the Communications of the ACM, including [StableMT CACM '14]. We believe that the proposed research will continue to produce research impact and be present in these conferences and journals.

Open source. As a tradition in our research, we would like to share our code and results with the research community and industry. We have made several systems and their evaluation results open source, including [CRANE SOSP '15]: `http://github.com/columbia/crane`, [PARROT SOSP '13]: `http://github.com/columbia/smt-mc`, and [LOOM OSDI '10]: `http://github.com/columbia/loom`. We expect to make this proposed project open source after it is done.

Media. We expect to share our thoughts and results with the media and people out of our research fields. Because our previous research addresses important problems and we have built practical systems, these systems have been featured in websites such as `ACM Tech News`, `TG Daily`, and `Phys.org`. One could google "heming cui software reliability phys.org" and she/he will find an article that introduces our StableMT systems with the title "Major Breakthrough Improves Software Reliability and Security". We will continue to share our results with media in this proposed project.