First-Aid: Providing Emergency Treatments to Memory Bugs in Software during Production Runs

Qi Gao Wenbin Zhang Yan Tang Feng Qin
The Ohio State University

1 Motivation

Memory management related bugs in C/C++ programs are one major type of software defects that severely reduce software dependability at production runs. These bugs such as buffer overflows and dangling pointers can corrupt memory data, leading to software failures such as program crashes and hangs. Furthermore, malicious users often launch attacks via exploiting memory bugs.

One way to handle memory bugs that are manifested at production runs is relying on developers to diagnose the bugs and generate bug-fixing patches. However, the time window from a bug report to the installation of bug-fixing patches could be long due to the difficulties of diagnosis [5] and concerns on the correctness of the patches. Previous studies show that it takes a few weeks or even months for users to install patches on software in production [1]. During this window, users either have to run the software with bugs and bare the problems such as intermittent crashes and potential attacks, or shut down the software partially or completely and experience possibly expensive downtime. *Therefore, it is critical to have online "treatments", before applying vendor's patches, to memory bugs at production runs so that programs can continue execution correctly.*

There are a few studies on tolerating software failures caused by one or more types of memory bugs. Failure-oblivious computing [7, 6] can tolerate buffer overflows by dropping overflowed writes and providing manufactured or cached values for overflowed reads. In general this method is unsafe. DieHard [2] and Archipelago [8] are two interesting preventive approaches for tolerating memory bugs. Although both approaches can probabilistically tolerate memory bugs, they require much more memory resources than the original programs do. Rx [3] tolerates memory bugs by applying environmental changes upon failures. However, Rx's diagnostic process is primitive – not accurately identifying buggy memory objects. As a result, it has to apply memory changes for all objects during recovery and thus may incur higher runtime overhead. Furthermore, failures caused by the same memory bugs can occur repeatedly since Rx disables the environmental changes after surviving failures.

Exterminator [4] pioneered in automatically diagnosing and fixing memory bugs at runtime. Based on heap randomization techniques it diagnoses memory bugs by comparing multiple heap images. While this approach is effective in diagnosing memory bugs, the resource usage is large if performing in the iterative or replicated mode (multiple replicas or multiple runs, each with at least double heap size). While in cumulative mode, it requires tens of samples with the same bug triggered in all of them. These factors significantly restrict Exterminator being widely adopted in production runs. Additionally, it is unclear how it handles the cases where multiple memory bugs manifest themselves in the programs.

2 Our Idea

Figure 1 shows the idea of First-Aid. In the normal run, First-Aid periodically takes checkpoints for the program. Upon failures, First-Aid works around the occurring bugs through three steps: the lightweight on-site bug diagnosis, the patch generation, and the patch application. Furthermore, First-Aid provides useful diagnostic information to developers for nailing down the root causes and fixing the bugs.

Bug diagnosis. In this step, First-Aid rolls back the program to previous checkpoints, tentatively applies some diagnostic treatment to all or partial memory objects, and re-executes the program. Based on the re-execution results (e.g., a failed or successful run), First-Aid decides whether to perform another re-execution with different diagnostic treatment from the same or different previous checkpoints. It repeats this process until it identifies the bug type and bug-triggering memory objects.

The diagnostic treatment can tolerate different types of memory bugs and provide the evidence of occurring bugs. Based on this information, we can identify the bug type and bug-triggering memory objects. For example, the diagnostic treatment for dangling pointer writes is to skip free operations and fill the freed objects with "canary", i.e., certain content pattern that is unlikely used by programs. If this treatment survives the program from failures and the "canary" is modified, it is very likely that dangling pointer writes occur on some freed memory objects. Table 1 lists all the diagnostic treatment for different bugs, including buffer overflow, dangling pointer read, dangling pointer write, double free, invalid free, and uninitialized read.

In some cases, a program may contain multiple bugs that lead to a failure, which makes the situation more complicated. To address this problem, First-Aid identifies one bug at a time by masking the manifestation of other bugs. For example, for diagnosing whether there are uninitialized read bugs in a program, First-Aid applies

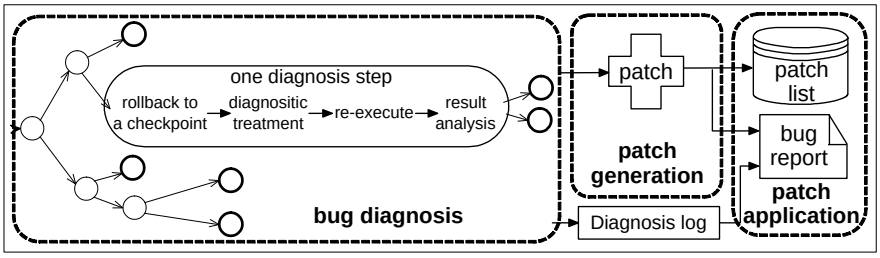

Figure 1: A working scenario of First-Aid

Bug type	Common reason(s) for the bug	Diagnostic treatment	Emergency treatment	Patch point
buffer overflow	1.underestimate required length when allocation; 2.miscalculation of offset when accessing	add padding and fill padding with canary	allocate a larger buffer	allocation
dangling pointer read	1.free buffer too early (before legitimate access);	1. skip free 2. skip free and fill with canary	skip free	deallocation
dangling pointer write	2.forget to set the pointer to NULL after free	skip free and fill with canary		
double free		skip free		
uninitialized read	assume new allocated buffers contain all zero	1. fill buffer with zero 2. fill buffer with canary	fill buffer with zero	allocation

Table 1: Types of memory bugs and corresponding treatment

the treatment for other types of bugs. If the re-execution fails, it is likely that the program contain uninitialized read bugs. As a result, First-Aid will apply its treatment for future diagnostic process. If the re-execution succeeds, the program unlikely contains uninitialized reads.

At the end of the diagnosis process, there are three possible results of the occurring bugs: (1) non-deterministic bugs; (2) memory-management-related bugs; (3) non-memory-management-related bugs. First-Aid will only handle the second case and pass the bug type and bug-triggering objects to the next two steps.

Patch generation. Based on the bug type information reported by the diagnosis step, First-Aid generates corresponding runtime patches. Similar to our previous work Rx, the patches are based on common mistakes programmers often make in memory management. For example, buffer overflow bugs are usually caused by underestimation of the buffer size needed for storing data or miscalculation of the offset when accessing data. A corresponding patch is to conservatively allocate a larger buffer for memory objects, which protect its neighbor regions from corruption and thus avoid failures. Table 1 shows the patches in First-Aid for handling common memory bugs. Unlike the diagnostic treatment, the patch for each bug does not fill canary in memory objects or paddings and thus incurs less runtime overhead.

Patch application. To prevent the same bug from recurring, First-Aid applies the patch to memory objects that have the same call-sites of allocation or deallocation as the bug-triggering objects. It is effective because memory objects with the same call-sites of allocation or deallocation often have similar characteristics such as leaking or overflow [4]. Additionally, patch application in First-Aid is much more efficient than Rx. Without detailed bug diagnostic information, Rx has to apply patches to all memory objects, which is expensive, and disables the patches after the programs pass the buggy regions. As a result, Rx cannot prevent the same bug from occurring even if the patch has been found and applied once. Furthermore, First-Aid applies the same patch to other processes that are running the same program so that they can prevent triggering the same bug.

Instead of hiding bugs from users or developers, First-Aid assists developers in diagnosing and fixing the bugs off-site by providing detailed bug reports. When the patch is applied online, First-Aid generates a bug report including the patch information and the diagnosis log. With the detailed call-site information in the patch, developers can easily locate the bug-related memory management code, which may greatly help diagnosing the root cause. If First-Aid fails to generate a runtime patch, developers can quickly narrow down the search scope to bugs other than memory management related ones.

References

[1] A. Joshi et al. Detecting past and present intrusions through vulnerability-specific predicates. In *SOSP'05*.

[2] E. D. Berger et al. Diehard: probabilistic memory safety for unsafe languages. *SIGPLAN Not.*, 41(6):158–168, 2006.

[3] F. Qin et al. Rx: Treating bugs as allergies – a safe method to survive software failures. In *SOSP'05*.

[4] G. Novark et al. Exterminator: automatically correcting memory errors with high probability. In *PLDI'07*.

[5] J. Tucek et al. Triage: diagnosing production run failures at the user's site. In *SOSP'07*.

[6] M. Rinard et al. A dynamic technique for eliminating buffer overflow vulnerabilities (and other memory errors). In *ACSAC'04*.

[7] M. Rinard et al. Enhancing server availability and security through failure-oblivious computing. In *OSDI'04*.

[8] V. B. Lvin et al. Archipelago: trading address space for reliability and security. In *ASPLOS'08*.