Automated Code Repair to Ensure Memory Safety

Software vulnerabilities constitute a major threat to DoD. Memory violations are among the most common and most severe types of vulnerabilities.

The main technique that we use (fat pointers) has been previously researched as a compiler pass to repair the intermediate representation (IR) of a program. Our work is novel in applying it as a source-code repair, which poses the difficulty of translating the repairs at the IR level back to source code.

Repair of source code	As a compiler pass
Repairs can be easily audited if desired.	Must trust the tool.
Repairs can be manually tweaked to improve performance.	Difficult to remediate performance issues caused by repair.
Changes to the source code are frequent and easily handled.	Changes to the build process may be difficult and costly.

The C preprocessor can include or exclude pieces of C code depending on the configuration chosen at compile time. We repair configurations separately and merge the results, as illustrated in Figure 3.

We are developing **automated techniques** to repair C source code to **eliminate memory-safety vulnerabilities.**

Figure 1(a): Original Source Code

```
1
2
3   #define BUF_SIZE 256
4   char nondet_char();
5
6   int main() {
7       char* p = malloc(BUF_SIZE);
8       char c;
9       while ((c = nondet_char()) != 0) {
10          *p = c;
11          p = p + 1;
12      }
13      return 0;
14  }
```

Figure 1(b): Repaired Source Code

```
1   #include "fat_header.h"
2   #include "fat_stdlib.h"
3   #define BUF_SIZE 256
4   char nondet_char();
5
6   int main() {
7       FatPtr_char p = fatmalloc_char(BUF_SIZE);
8       char c;
9       while ((c = nondet_char()) != 0) {
10          *bound_check(p) = c;
11          p = fatp_add(p, 1);
12      }
13      return 0;
14  }
```

We ensure spatial memory safety by replacing raw pointers with **fat pointers**, which include bound information.

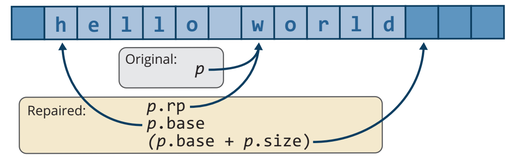
Before dereferencing a fat pointer, a bounds check is performed.

For each pointer type $T*$, we introduce a new struct definition:

```
struct FatPtr_T {
  T*     rp;   /* raw pointer */
  char*  base; /* of mem region */
  size_t size; /* in bytes */
};
```

Pointers stored in heap memory that is reachable by external binary code cannot be fattened. We identify such pointers using a whole-program points-to analysis with an allocation-site abstraction.

Figure 4. Example of fattening a pointer



Limitations: No guarantee of memory safety in the presence of concurrency and things that interact poorly with fat pointers.

Current status: Our tool works on small test cases. We are fixing remaining bugs and adding missing features to handle the SPEC2006 benchmarks.

FY20: Optimize to remove unnecessary fattenings and bound checks.

Future: Extend to other types of repairs and increase level of automation. Work with additional DoD transition partners.

Figure 2. Pipeline for repair of source code

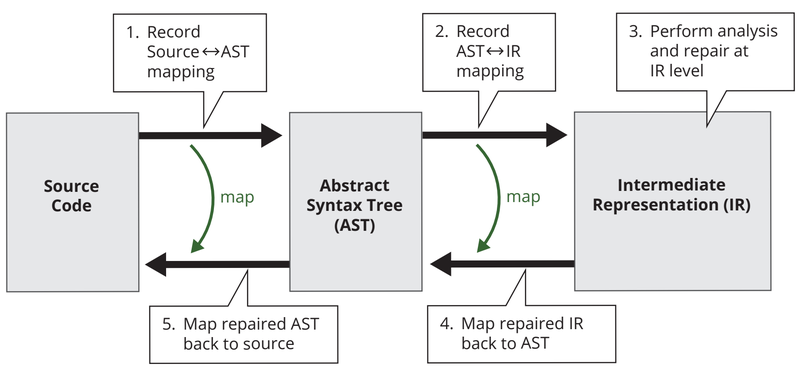
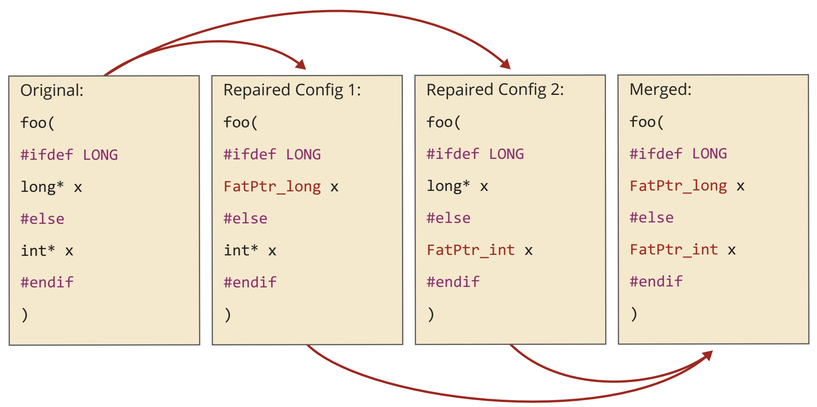


Figure 3. Merging of repairs for multiple build configurations.



Carnegie Mellon University
Software Engineering Institute

Will Klieber | weklieber@cert.org
Ryan Steele, Matt Churilla, Derek Leung, David Svoboda, Mike McCall, Ruben Martins

Distribution Statement A: Approved for Public Release;
Distribution is Unlimited

04_Automated_Code_Repair_to_Ensure_Memory_Safety_2.indd 1 10/10/19 4:30 PM

Copyright 2019 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.
External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use.
Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM19-1029