Inference of Memory Bounds

Invalid memory accesses are one of the most prevalent and most serious software vulnerabilities. This project aims to detect and repair not only out-of-bounds WRITEs, but also out-of-bounds READs, which are a relatively newer problem that can leak highly sensitive information.

A prime example of out-of-bounds READs is the OpenSSL HeartBleed vulnerability, which could be used to compromised the SSL private keys of two thirds of all websites. This type of vulnerability is unaffected by mitigations such as ASLR and DEP.

In general, for a re-usable buffer with stale data, READs should be bounded to the valid portion of the buffer. This type of problem affects even memory-safe languages such as Java. For example, the Jetty web server leaked passwords and any other data contained in a previous HTTP request.

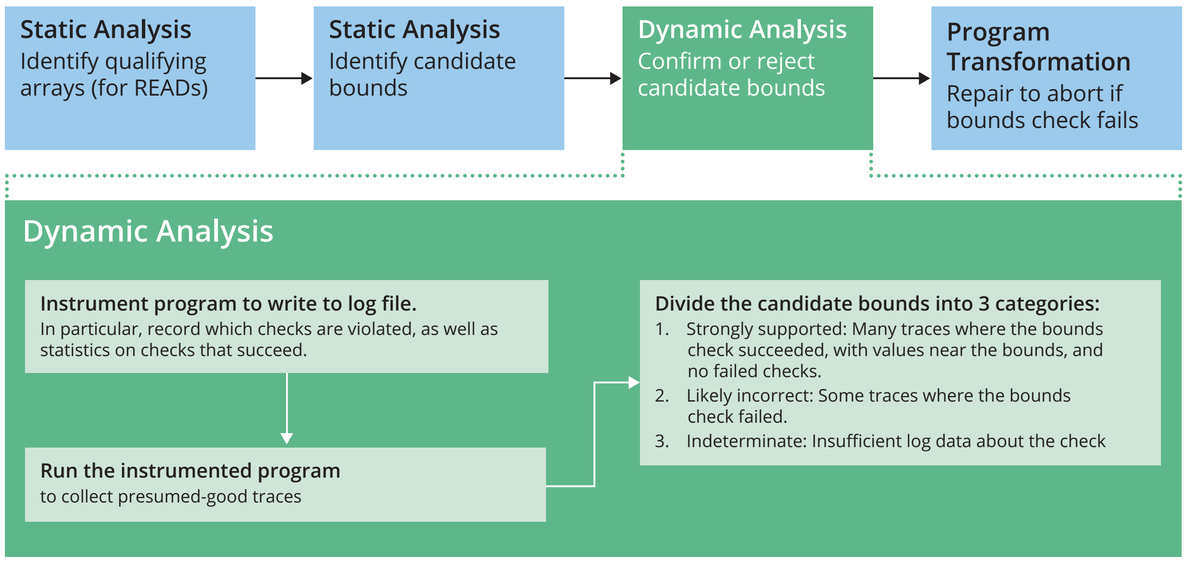
Example: Re-used buffer with state data

Buffer contents after the first HTTP request:

Buffer contents after the second HTTP request:

Upper bound for reading:
most recently written location

This project is also useful for a second problem: decompilation of binaries. The relations between reconstructed fields is usually is left for the human analyst to manually investigate. We will try to reconstruct information of the form "[*n, m*] is bounds of pointer *p*".



Strategies to propose candidate bounds:

1. (For reads) The most recently written position in the buffer.
2. Bounds of region allocated by malloc.
3. Pointer arithmetic with constant offset (e.g., field of a struct)—mainly for use in decompilation.
4. Analysis of memory accesses within loops and limits of the loop.
   - Exact if the number of iterations is known at start of loop.
   - Only a candidate bound if it is possible to break out of the loop early.
5. Invariants for structs (by typename or by allocation site).
   - Suppose that we discover that, in most of the program, one field of a struct supplies the bounds of another field of the struct.
   - Then we guess that this is an invariant and violations of it are errors.
6. If in most callsites of a function foo(int n, char *p, ...), the bounds on p is the closed interval [p, p+n-1], then propose that in the other callsites, the same bounds should apply.

How do we determine which arrays should be subject to the analysis for READs outside the valide portion of an arrary?

- We consider an array to be a *qualifying array* if every write to the array is at either index 0 or at the successor of the last written position.

How do we identify the valid portion of the array?

- Heuristic: It is from the start of the array up to and including the last written element of the array.

How often do qualifying arrays occur in real-world programs?

- Imprecision in static analysis might cause false negatives.
- To establish ground truth, we do a separate dynamic analysis (next column).

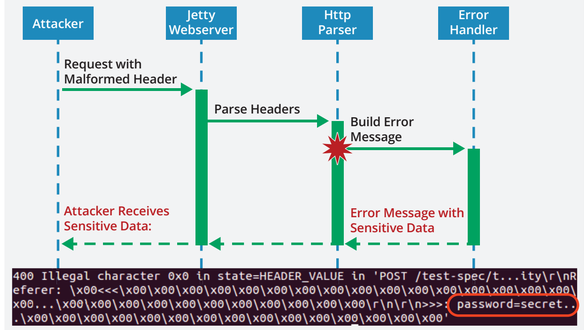
Stand-alone dynamic analysis for out-of-bounds READS:

We have written a Java agent to:

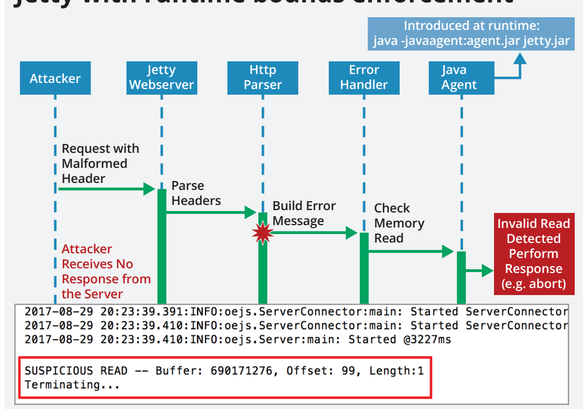
- Record the allocation site and the last written position (LWP) of each allocated ByteBuffer.
- Check whether each write to the ByteBuffer is consistent with definition of qualifying array.
- If all the writes have been qualifying, we flag any reads beyond LWP.

- Note that this dynamic analysis is different than the dynamic validation of statically-inferred candidate bounds.
- With this tool, we dynamically patch Jetty to prevent leakage of sensitive information, as shown below.

Unpatched Jetty



Jetty with runtime bounds enforcement



Will Klieber | weklieber@cert.org

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.

DM17-0739
Inference of Memory Bounds