Since user address space can be modified by any thread or application at any time, if the kernel wants to use something in its address space, it first copies the memory block in its address space and then works on the copied memory block, because of this separation kernel may further need to re-fetch the block which may lead to multiple read or double-fetch. The reason kernels do not use single fetch is that it does not know how many bytes it needs beforehand and fixed length buffer would waste memory and degrade performance, therefore in the first fetch kernel fetches the header which indicates how many bytes it needs to read, and after some sanity checks (e.g. whether the size is greater than some value) then re-fetches the block with that header, but the problem comes to light when during the first fetch some user space thread changes the header, then after the second fetch, kernel gets the whole block but with a wrong header. Detecting these double-fetches is somewhat difficult because there are many patterns that may lead to double-fetching, the example above was the size checking pattern. Some other patterns are dependency look-up, signature/protocol check, information guessing and etc. This paper introduces a way called DEADLINE to automatically handle this situation.
What Deadline basically does is that it first fetches all the multiple reads and then symbolically checks each code path to determine whether these two fetches make a double-fetch bug.

Deadline redefines the double-fetch into 3 parts:

1. Two fetches from an overlapping user space memory
2. The two fetches must have a relation on the overlapped region, can be either control-dependence (for example checking the version of TLS) or data-dependence (a variable is modified before the second fetch)
3. We cannot prove that the relation in the first fetch would be the same after the second fetch

If all conditions are satisfied then we can say there is a double-fetch (meaning not all the multiple reads are double-fetch bugs).

The ideal solution to find double-fetches which uses top-down approach has three steps:

1. Find all fetches in the kernel
2. For the whole kernel construct the Control Flow Graph
3. Perform pair-wise reachability tests for each of fetches

Step 2 and 3 are not possible to do in the kernel mode.

Deadline uses a bottom-up approach which consists of two steps:

1. Find all fetches in the kernel
2. For each fetch, in the function that the fetch occurred, scan its reaching instructions which are fetch or functions which may have fetches.

After this part is completed then it uses symbolic checking to detect if the three conditions are satisfied.

For example, for the overlapping region:

F1 indicating the first fetch, F2 indicating the second fetch

assert $F2.A <= F1.A < F2.A + F2.S$ OR assert $F1.A <= F2.A < F1.A + F1.S$

Meaning that either the address of first fetch is between the address of second fetch and an offset size from the address of the second fetch.

Deadline tries to linearize the instruction paths, for instructions that are not loops or branches, it behaves like reading one instruction after another, but when reaches a loop it uses loop unrolling which only unrolls once, and one of its limitations is that it only analyses one path of the branch.

The basic idea to reduce the double-fetch bugs, is to re-assure the control-dependence and data-dependence between the two fetches.

There are four generic patterns to do this:

1. Override after second fetch (override the overlapped region with the value from the first fetch, discarding the value from the second fetch)
2. Abort on change (check whether something is modified after the first fetch, if occurred then abort)
3. Refactor overlapped copies into incremental copies (incremental fetch, fetch first region then fetch the offset of that fetch)
4. Refactor overlapped copies into a single-fetch

These are some generic patterns and may not work with every double-fetch. There are some limitations such as files not compilable under LLVM, Loop unrolling limit and Symbolic checking limitations.

In Short, Deadline uses a precise modeling of double-fetch bugs and has high accuracy and high scalability. It can be used on not only kernels, but hypervisors, browsers, etc.