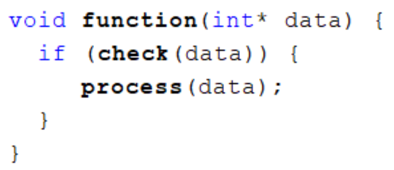
Pointer time of check – time of use

Pointer time of check – time of use (TocTou) code check belongs to a group of code checks, by default, focusing on untrusted interface functions. These functions represent program/library interface with the outside world and take user-provided input, requiring more stringent security measures to prevent exploitation.

TocTou is a class of software security vulnerabilities caused by a race condition in checking user-provided data. As an example, we can take a look at the function in figure 1 which illustrates this vulnerability. Both `check` and `process` functions take user provided pointer as an argument and dereference it to access its value. An adversary could modify pointer value in between functions calls resulting in invalid data being passed to the `process` function.



```
void function(int* data) {
  if (check(data)) {
     process(data);
  }
}
```

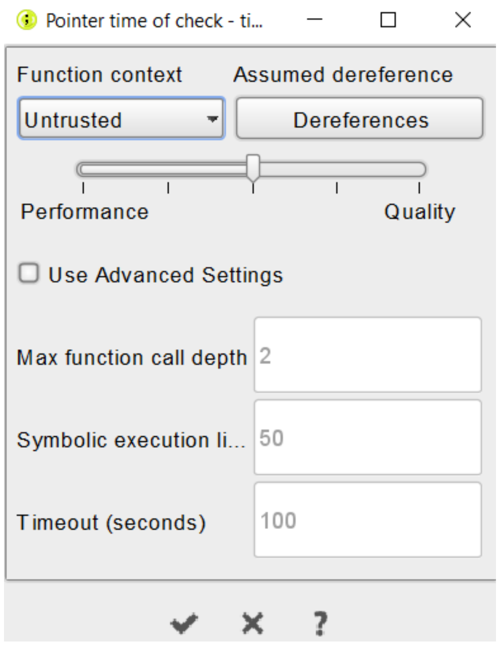
Figure 1 Example of TocTou vulnerability

In general, for TocTou the order of check and use in two dereferences doesn't matter as all combinations are potentially vulnerable. The general solution for this vulnerability would include dereferencing pointer once, storing data locally, and performing all checks and further processing on a local copy of data.

Pointer time of check – time of use code check can help in identifying functions containing TocTou vulnerabilities. To set up code checker configuration, we need to:

- set target context
- set list of functions which dereference pointer arguments
- adjust performance slider

The button to open the configuration dialog for code check is available on Code check panel next to its name.


Pointer time of check - ti...

Function context: Untrusted

Assumed dereference: Dereferences

Performance — Quality

☐ Use Advanced Settings

Max function call depth: 2

Symbolic execution li...: 50

Timeout (seconds): 100

Figure 2 Pointer time of check – time of use configuration

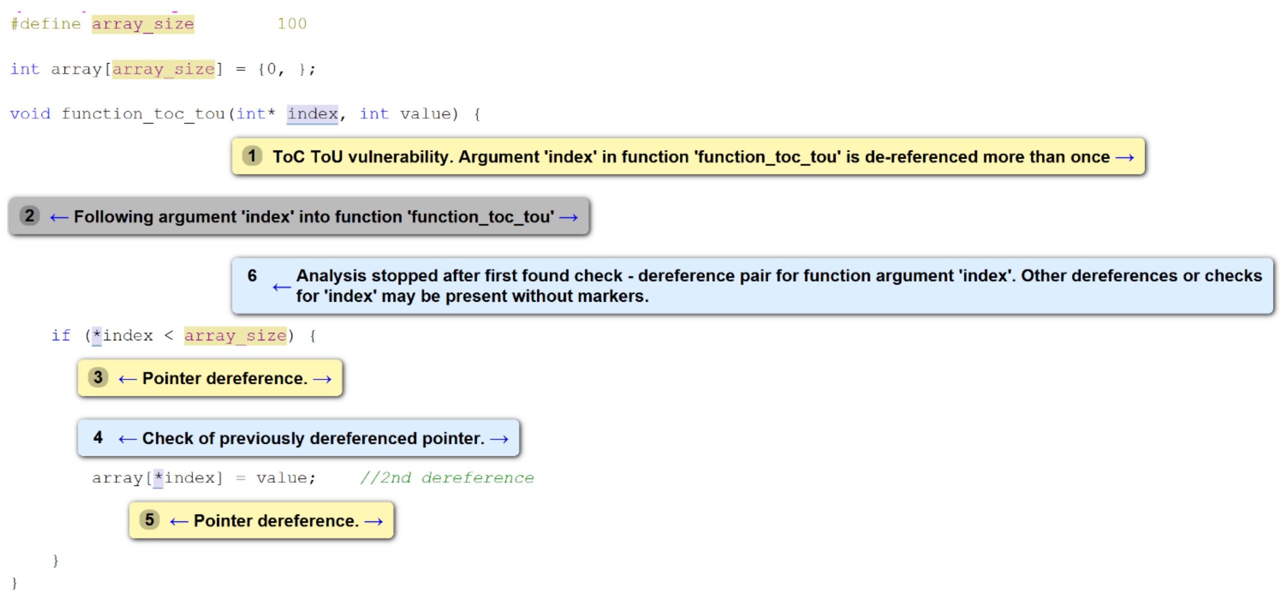
In True Code, context serves as a way to logically group functions based on some characteristics (i.e. list of functions representing application interface). The procedure of creating/editing contexts is explained in more detail in the True Code manual. During the analysis of function, if code check encounters a case where a pointer is passed as an argument to a function that is not part codebase (for example library function) it relies on assumed dereference list. Assumed dereferences list, as the name implies, is a user-settable list of functions for which code check assumes a pointer dereference if it gets passed to them as an argument. By default, this list includes standard C library functions for working with memory such as memcpy, memcmp, etc. The performance slider enables the user to choose between different presets of advanced settings. Advanced settings include:

- maximum function call depth
- symbolic execution limit
- timeout

The maximum function call depth sets a limit on how deep in the call stack, from the start function, pointer function arguments will be followed. The symbolic execution limit and timeout both serve as a way to limit the maximum runtime of the code checker per function. In general, lower values of advanced settings reduce processing time but as a tradeoff code check may not find certain vulnerabilities.

Once the checker configuration has been set up, it can be run to analyze all pointer arguments of functions in a selected context. All reported findings will be available to the user as annotations through Annotation tab of True Code. Users can right-click on any annotation in the table and select 'Show" to get a more detailed overview of the detected issue. It is important to note that Pointer time of check – time of use code check stops analysis of a function and reports detected TocTou finding as soon as one is encountered. For this reason, it is recommended to fix detected bugs and re-run analysis to check if the issue is fixed and if there were more previously unreported issues.

An example of generated TocTou annotation is shown in figure 3.



```
#define array_size        100

int array[array_size] = {0, };

void function_toc_tou(int* index, int value) {
                    1 ToC ToU vulnerability. Argument 'index' in function 'function_toc_tou' is de-referenced more than once →
2 ← Following argument 'index' into function 'function_toc_tou' →
                    6 ← Analysis stopped after first found check - dereference pair for function argument 'index'. Other dereferences or checks
                        for 'index' may be present without markers.
    if (*index < array_size) {
        3 ← Pointer dereference. →
        4 ← Check of previously dereferenced pointer. →
        array[*index] = value;     //2nd dereference
            5 ← Pointer dereference. →
    }
}
```

Figure 3 Example of Pointer time of check – time of use annotation