C++ static code analysis: Unnecessary expensive copy should be avoided when using auto as a placeholder type

2-3 minutes

Unintentional expensive copy should be avoided when using `auto` as a placeholder type.

When using `const auto` as a placeholder type, you might unintentionally forget to add an ampersand(&) after the `auto` keyword. This can silently create a pointless copy and possibly have a bad impact on the performance of your code depending on the size of the created object and its context.

For example, if it happens in a range-based for loop context, it is going to lead to creating as many useless objects as the size of the range.

This rule will detect a declaration of an unmodified local variable with expensive to copy type and `auto` as a placeholder type that is initialized with a non-temporary object.

Noncompliant Code Example

```
void printVec(const std::vector<std::string>&
namesOfTheEntirePopulation) {
  for (const auto name : namesOfTheEntirePopulation) { //
Noncompliant
    std::cout << name;
  }
}

void ignore(const std::vector<std::string>& vec);
void ignoreAgain(const std::vector<std::string>& vec);

void ignore(VecWrapper vec) {
  const auto namesOfCPPHaters = vec.getNamesOfCPPHaters(); //
Noncompliant
  ignore(namesOfCPPHaters);
  ignoreAgain(namesOfCPPHaters);
}
```

Compliant Solution

```
void modifyName(std::string& a);
void printVec(std::vector<std::string>&
namesOfTheEntirePopulation) {
  for (const auto& name : namesOfTheEntirePopulation) { //
Compliant
    std::cout << name;
  }

  for (auto name : namesOfTheEntirePopulation) { // Compliant: a
copy is needed to avoid modifying the original list of names
    modifyName(name);
    std::cout << name;
  }
}

void ignore(const std::vector<std::string>& vec);
void ignoreAgain(const std::vector<std::string>& vec);

void ignore(VecWrapper vec) {
  const auto& namesOfCPPHaters = vec.getNamesOfCPPHaters();
// Compliant
  ignore(namesOfCPPHaters);
  ignoreAgain(namesOfCPPHaters);
}
```