OOP31-PL. Do not access private variables or subroutines in other packages

Perl provides no mechanism to hide variables or functions. Although it provides mechanisms such as `my()` that limit the scope of variables, it does nothing to prevent code from accessing any variable or method that is available and dereferenceable from its current scope.

By convention, packages may indicate that a method or variable is not to be used outside the class by prefixing the method or variable name with an underscore (_). [Conway 2005] Perl provides no inherent enforcement of this convention; however, it is followed by many modules in CPAN and other developers. This convention must not be violated.

Noncompliant Code Example

This noncompliant code example provides a Car package that registers cars and keeps track of all cars that it creates.

```perl
{ package Car;

    my %_all_cars;

    sub new {
        my ($class, $type) = @_;
        my $self = {type=>$type};
        bless $self, $class;
        %_all_cars{$self} = 1;
        return $self;
    };

    sub DESTROY {
        delete %_all_cars{shift()};
    };

    sub type {
        my $self = shift;
        return $$self{type};
    }

    sub _get_all_cars {
        return %_all_cars;
    }
}

my $mine = Car->new("Transam");
my $type = $mine->type();
print "I drive a $type.
"
my $yours = Car->new("Corvette");
$type = $yours->type();
print "You drive a $type.
"

my %cars = Car::_get_all_cars();
my @all = keys %cars;
my $count = $#all + 1;
print "There are $count cars on the road.
";
```

This program behaves as expected, correctly reporting 2 cars on the road. However, it clearly violates encapsulation, because the _get_all_cars() method is considered private within the Car class.

Compliant Solution

This compliant solution adds a public method and invokes it instead of any private method.
{  
    package Car;  

    my %_all_cars;  

    sub count_cars {  
        my @all = keys( %_all_cars);  
        return 1 + $#all;  
    }  

    # ... other methods of Car  

    # ...  
    my $count = Car::count_cars();  
    print "There are $count cars on the road.\n";  
}

Exceptions

OOP3:EX0: A class may access private subroutines in classes it is inherited from.

OOP3:EX1: This rule does not apply to classes or modules that do not obey this convention. For instance, the POSIX package provides an API to the system calls endorsed by the POSIX standard. Many of these calls begin with _ but are not meant to be private.

Risk Assessment

Using deprecated or obsolete classes or methods in program code can lead to erroneous behavior.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP3-PL</td>
<td>Medium</td>
<td>Probable</td>
<td>Medium</td>
<td>P8</td>
<td>L2</td>
</tr>
</tbody>
</table>

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Diagnostic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl::Critic</td>
<td>Subroutines::ProtectPrivateSubs</td>
</tr>
<tr>
<td></td>
<td>Variables::ProtectPrivateVars</td>
</tr>
</tbody>
</table>

Bibliography

[Conway 2005]
[CPAN] Elliot Shank, Perl-Critic-1.116 Subroutines::ProtectPrivateSubs, Variables::ProtectPrivateVars
[CPAN] Ragwitz, Florian, POSIX
[Open Group 2008]