IDS32-PL. Validate any integer that is used as an array index

Perl, unlike most other languages, uses arrays that are not declared with a particular length and that may grow and shrink in size as is required by subsequent code. In fact, when assigning a value to an element within the array, if the index provided is beyond the end of the array, the array grows to make it valid. Consider the following example:

my @array = (1, 2, 3);  # array initialized
print "Array size is $#array\n";  # 2 (index of last element)
$array[5] = 0;  # array grows so that reference is valid
print "Array size is $#array\n";  # 5
my $value = $array[7];  # array unchanged + uninitialized value warning
$array[-7];  # array unchanged + uninitialized value warning
if (exists $array[9]) {  # false, array unchanged
    print "That's a big array.\n";
}
print "Array size is $#array\n";  # 5
$value = $array[10][0];  # reading a value in list context grows array
print "Array size is $#array\n";  # 10!

This automatic growth occurs only if the index provided is positive and the array value is being written, not read, and not passed to a testing function like exists() or defined().

If an attacker is able to substitute a number to be used as an array index and provides the value 1000000000 (1 billion), then Perl will happily try to grow the array to 1 billion elements. Depending on the platform's capabilities, the attempt to grow the array might fail, or hang, or simply cause Perl to consume several gigabytes of memory for the lifetime of the array. Because a consequent denial of service could occur, attackers must not be permitted to control array indices.

Noncompliant Code Example

This noncompliant code example takes a set of users via standard input and adds them to an array, indexed by their UIDs. This program may, for instance, be fed the contents of the /etc/passwd file.

my @users;
while (<STDIN>) {
    my ($username, $dummy, $uid) = split( ":\n" );
    if (not (defined( $uid ) and defined( $username ))) {next;}
    if (not $uid =~ ^\d*$ ) {next;}
    $users[$uid] = $username;
}
# ... Work with @users

This code clearly skips input lines that do not contain a valid UID or user name. It also skips lines where the UID is not a positive number. However, a UID that is large might cause excessive growth of the @users array and provoke a denial of service.

Compliant Solution

This compliant solution enforces a limit on how large a UID may be. Consequently, the array may not contain more than $max_uid elements.

my @users;
my $max_uid = 10000;
while (<STDIN>) {
    my ($username, $dummy, $uid) = split( ":\n" );
    if (not (defined( $uid ) and defined( $username ))) {next;}
    if (not $uid =~ ^\d*$ ) {next;}
    if ($uid > $max_uid ) {next;}
    $users[$uid] = $username;
}
# ... Work with @users
Risk Assessment

Using unsanitized array index values may exhaust memory and cause the program to terminate or hang.

<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>IDS32-PL</td>
<td>low</td>
<td>likely</td>
<td>high</td>
<td>P3</td>
<td>L3</td>
</tr>
</tbody>
</table>

Bibliography

[Wall 2011] peridata