INFO 101 – Introduction to Computing and Security / Tutorial – weeks 8 and 9

Prof. Dr. Rui Abreu and Prof. Dr. Franz Wotawa
University of Porto, Portugal and Technical University of Graz, Austria
rui@computer.org, wotawa@ist.tugraz.at

December 3, 2020

1 Introduction

This time the tutorial has two objectives, i.e.:

- Show challenges arising when using shell scripts and discussing their consequences with respect to security.
- Show how to program recursive functions in LIPS

2 Bash scripts and security

We already discussed security in the previous weeks. We also had a look at certain attacks and how they really can be carried out. In particular, I showed an example considering shell scripts. In this section, I summarize the example and discuss the findings and conclusions to be drawn.

Let us start with the following bash script named `exec`, which has the purpose of taking a command as the first input and executing it.

```
#!/bin/bash

echo "I am a simple script"
echo "using parameter ${1}"
echo ""

$1
```

When calling this script with `exec "ls -lsa .."` we would get the content of the directory (together with details) from the directory immediately above the current working directory. You would see something like:

```
ruimaranhao:labs ruimaranhao$ exec "ls -lsa .."
I am a simple script
using parameter ls -lsa ..

total 40
 0 drwxr-xr-x   7 ruimaranhao  staff   224 15 Nov 11:10 .
 0 drwxr-xr-x  12 ruimaranhao  staff   384 19 Nov 19:23 ..
16 -rw-r--r--@  1 ruimaranhao  staff  6148 20 Nov 08:52 .DS_Store
 0 drwxr-xr-x  27 ruimaranhao  staff   864 20 Nov 09:59 assignments
```

```
16 -rw-r--r--   1 ruimaranhao  staff  6086 29 Okt 20:53 emtpy.pdf
 8 -rw-r--r--   1 ruimaranhao  staff  1132 29 Okt 20:53 emtpy.tex
 0 drwxr-xr-x  29 ruimaranhao  staff   928 20 Nov 18:06 labs
 ruimaranhao:labs ruimaranhao$
```

Let us now call us the script with the parameter `"pwd"`:

```
ruimaranhao:labs ruimaranhao$ secScript "pwd"
I am a simple script
using parameter pwd

/Users/ruimaranhao/Lanzhou_University/2020/course_material/INFO_101/laboratory/labs
ruimaranhao:labs ruimaranhao$
```

Again we would receive a content that reflects the shell command provided as parameter. Hence, the script is working. Now consider a different shell script we call `secScript` that should print the content of the first parameter.

```
#!/bin/bash

echo "I am a simple script"
echo "using parameter ${1}"
echo ""

echo $1
```

When calling `secScript` with `"I am a test"`, the script returns:

```
ruimaranhao:labs ruimaranhao$ secScript "I am a test"
I am a simple script
using parameter I am a test

I am a test
```

Again this is the behavior we expect. We can try several other calls:

```
ruimaranhao:labs ruimaranhao$ secScript I am a test
I am a simple script
using parameter I

I
ruimaranhao:labs ruimaranhao$
```

In this case and because of not using quotes `" "` we have not one but 4 parameters (arguments) going to the shell script but only the first one is used. The result itself is correct with respect to our expectations. But now let us try the following `text;ls -lsa ..`:

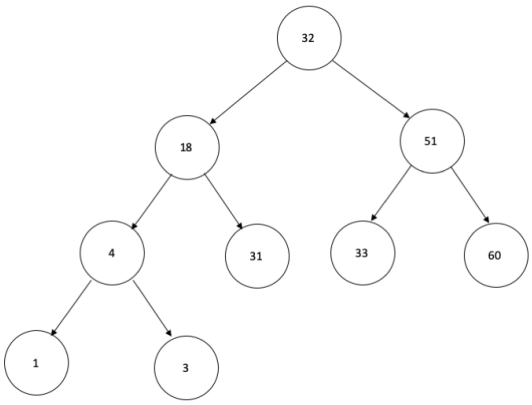
```
ruimaranhao:labs ruimaranhao$ secScript text;ls -lsa ..
I am a simple script
using parameter text

text
total 40
 0 drwxr-xr-x   7 ruimaranhao  staff   224 15 Nov 11:10 .
 0 drwxr-xr-x  12 ruimaranhao  staff   384 19 Nov 19:23 ..
16 -rw-r--r--@  1 ruimaranhao  staff  6148 20 Nov 08:52 .DS_Store
```

```
 0 drwxr-xr-x  27 ruimaranhao  staff   864 20 Nov 09:59 assignments
16 -rw-r--r--   1 ruimaranhao  staff  6086 29 Okt 20:53 emtpy.pdf
 8 -rw-r--r--   1 ruimaranhao  staff  1132 29 Okt 20:53 emtpy.tex
 0 drwxr-xr-x  29 ruimaranhao  staff   928 20 Nov 18:57 labs
ruimaranhao:labs ruimaranhao$
```

In this case the second part of the parameter was interpreted as a command and executed! This should have not happened and allows us basically to pass every command to the shell script! Note that the `';'` between `text` and `ls -lsa ..` is used to separate shell commands that should be in one line. If we use such a script on a server communicating over the internet, we might face security issues there. In order to avoid security trouble, we would need to sanitize the inputs (parameters or arguments) passed to a shell script. This can be done using a function that restricts inputs, e.g., using `egrep` for limiting input to text only.

3 LISP programming

Let us write a program for binary trees. A binary tree is a data structure comprising nodes, where each of them has either none or exactly two successors. Binary trees are often used in computer science, e.g., for storing data for search. A node that is not the successor of another node, is called the root. Nodes with no successors are leaf nodes. Usually, binary trees are graphically represented with the root node on top and the leaf nodes at the bottom. The following binary tree stores numbers, where each child tree on the left only comprises nodes with smaller values and on the right nodes with higher values. Such a data structure can be used for optimizing search.



Let us write LISP programs for this data structure:

- A predicate `is-valid` that returns true, if the given tree is really a binary tree accordingly to our definition.
- A predicate `is-element` that checks whether a given number is stored in a given tree.
- A function `leafs` that return only leaf nodes.
- A function `sum` that computes the sum of a values stored in such a tree, where we assume that only integer values are stored.

Before discussing the programs, we have to think of how to represent binary trees in LISP. We can easily do this by introducing a mapping from binary tree nodes to lists. A node n (where n is the value stored) having a left and right successor node n_l and n_r respectively, can be stored as `(N NL NR)`.

If a node is a leaf node, we set NL and NR to NIL. The binary tree given in the figure before, can be represented in LISP as follows:

```
(32 (18 (4 (1 nil nil) (3 nil nil)) (31 nil nil)) (51 (33 nil nil) (60 nil nil)))
```

Hence, in this example we store binary trees using nested lists where the first element of each list has to be an atom, and the second and third a list having the same properties.

The following programs implement the suggested LISP functions and predicates:

```
;;;; Lisp functions for binary trees (lab 101)
;;;; Note: binary tree nodes are stored using a list of length 2 where the first
;;;; element is the node name or content, the second element the left successor node,
;;;; and the third element the right successor node.

(defun element (a-list)
   (car a-list))

(defun left-tree (a-list)
   (cadr a-list))

(defun right-tree (a-list)
   (caddr a-list))

;;;; Check whether the given list is a binary tree
(defun is-valid (a-binary-tree)
  (if (null a-binary-tree)
      nil
      (if (eq (length a-binary-tree) 3)
          (if (or (null (left-tree a-binary-tree)) (null (right-tree a-binary-tree)))
               (and (null (left-tree a-binary-tree))
                       (null (right-tree a-binary-tree))
               (and (atom (element a-binary-tree))
                        (is-valid (left-tree a-binary-tree))
                        (is-valid (right-tree a-binary-tree))))
           nil)
       )
   )
)

;;;; Answers whether a given element is stored in the binary tree
(defun is-element (a-element a-binary-tree)
  (if (null a-binary-tree) nil
     (if (eq a-element (element a-binary-tree)) t
(or (is-element a-element (left-tree a-binary-tree))
           (is-element a-element (right-tree a-binary-tree))
       )
     )
  )
)

(defun is-leaf (a-binary-tree)
   (if (eq (length a-binary-tree) 3)
       (and (null (left-tree a-binary-tree)) (null (right-tree a-binary-tree)))
       nil ))
```

```
;;;; Returns all leafs of a binary tree
(defun leafs (a-binary-tree)
   (if (is-leaf a-binary-tree)
       (list a-binary-tree)
       (append (leafs (left-tree a-binary-tree)) (leafs (right-tree a-binary-tree)))
   )
)

;;;; Returns the sum of all values stored in the binary tree assuming that only
;;;; integers are stored
(defun sum (a-binary-tree)
   (if (null a-binary-tree) 0
       (+ (element a-binary-tree)
          (sum (left-tree a-binary-tree))
          (sum (right-tree a-binary-tree)))))
```