

ARBAC Verification Lab

Stefano Calzavara

April 2, 2020

1 Goals

In this lab class you will develop an ARBAC analyser for small policies. You are required to implement a program which parses the specification of a role reachability problem and returns its solution. You can implement the program using any programming language of your choice.

2 Specification

Your program must parse a role reachability problem expressed via the syntax in Table 1; for the sake of simplicity, we do not show the trivial pre-condition, which is always satisfied, represented as `TRUE`. You can assume that each section of the file is written on a single line, which simplifies parsing. Your program must return 1 when the role reachability problem is solvable and 0 otherwise.

An example role reachability problem is given in Table 2 for reference. For readability, the `CA` section of the example was split on multiple lines, but you can safely assume that this does not happen in practice.

Implementation suggestions:

1. Python is a great tool for this task, since it makes it easy to parse the input format (see the `split` function) and build appropriate data structures
2. Keep track in a data structure of all the user-to-role assignments which you have already explored, so that you don't explore the same part of the search space multiple times (this also prevents your program from looping)
3. Once you show that a role is reachable, terminate the exploration of the search space to save running time. If the search takes too long, force its termination after a timeout
4. Implement the forward slicing and the backward slicing algorithms discussed in the previous class to make the analysis scale. Observe the importance of slicing for performance!

Once your program seems correct, download the policies available on the course website¹. The flag of the challenge is the binary string obtained by concatenat-

¹<https://secgroup.dais.unive.it/wp-content/uploads/2020/04/policies.zip>

```

Roles  Role1 Role2 ... RoleN ;
Users  User1 User2 ... UserM ;
UA     <UserA,RoleB> ... <UserX,RoleY> ;
CR     <RoleA,RoleB> ... ;
CA     <RoleA,Pos1&...&PosJ&-Neg1&...&-NegK,RoleT> ... ;
Goal   RoleG ;

```

Table 1: Syntax of Role Reachability Problems

```

Roles  Teacher Student TA ;
Users  stefano alice bob ;
UA     <stefano,Teacher> <alice,TA> ;
CR     <Teacher,Student> <Teacher,TA> ;
CA     <Teacher,-Teacher&-TA,Student>
      <Teacher,-Student,TA>
      <Teacher,TA&-Student,Teacher> ;
Goal   Student ;

```

Table 2: Example of Role Reachability Problem

ing the bits encoding the solution to the provided role reachability programs, respecting their order. For example, in the case of two role reachability problems P_1 and P_2 with solutions 1 and 0, the flag would be the binary string 10. In the writeup, just put a link to a permanent remote repository where you include the code of your analyzer. Ensure the code is appropriately commented and consider the inclusion of a separate PDF report where you discuss selected aspects of your implementation or analysis.

If your analyzer cannot deal with a few policies, you can still bruteforce the missing bits of the flag and pass the assignment! However, please do your best to ensure the analyzer works on as many policies as possible!