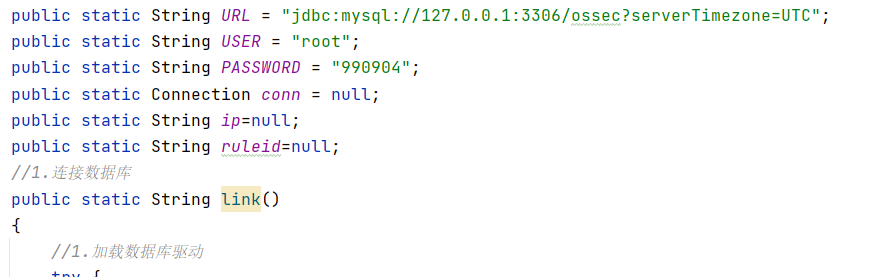
主要分为两个部分:

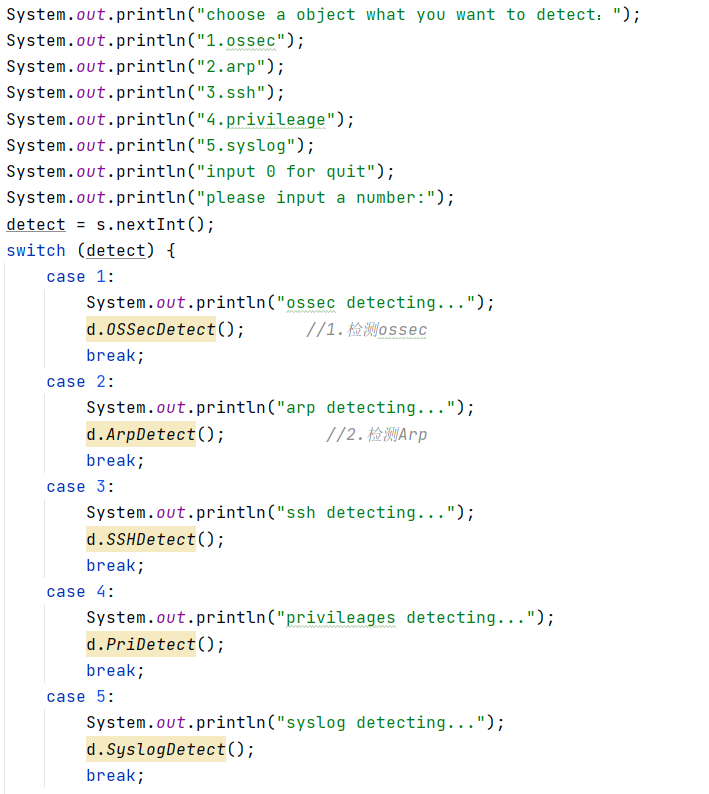
从程序的运行角度讲授细节

**第一部分：**java文件，对ossec中的数据进行提取过滤，以攻击树的形式展现攻击进行的程度。

修改掉其中db.java的数据库配置即可运行。



运行之后，输入一个数字代表进行某项检测，我以3为例（ssh）,以下是main.java的内容。



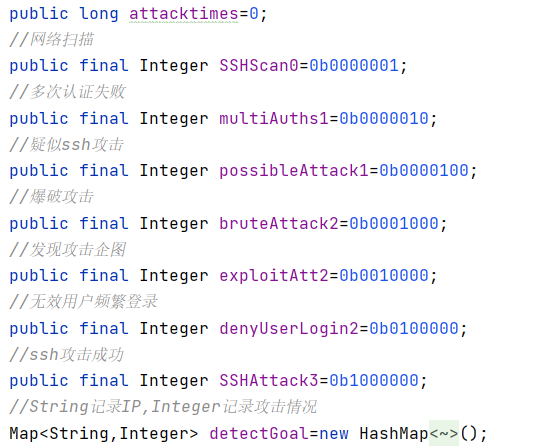
调用了d.sshdetect()函数，在db.java中被定义。他的工作是提取ossec中alert表的rule\_id（代表对应的规则），src\_ip两项。调用ssh.detect去处理。



Ssh.detect函数在detectsshattack.Java中被定义，

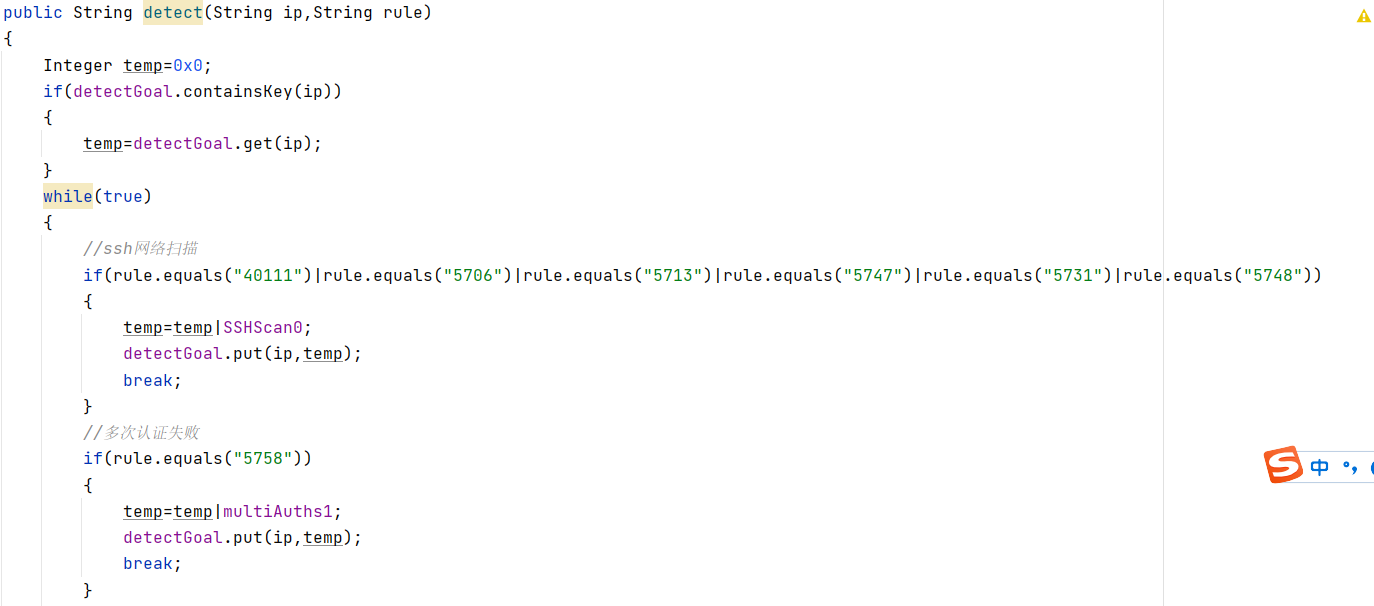
在介绍ssh.Detect函数之前，我们需要理解一个结构map<ip,rule\_group>,第一项是ip,第二项是规则结合，以rule\_group=0b000101为例，其中的每一位代表一种攻击。（**其中任何一位为1代表有过对应的攻击**）

如下图，这种情况下，代表该ip有过SSHScan0,possibleAttack1攻击，“SSHscan0”字符最后的一个0代表攻击级别。



以src\_ip=a,rule\_id=5706为例。他命中了SSHscan0,所以map多了一项（a,0b001）;

如果之后再检测到了一条记录src\_ip=a,rule\_id=5758，则map中的（a,0b001）变为（a,0b011）。



以上大致就是检测过程，其他ossec,arp等检测同理。

**第二部分：python文件，这一部分主要是做一个dnn模型，在kdd99异常检测数据集上做一个检测。**

**这部分思路比较固定先对kdd99数据进行一个预处理，然后送去dnn模型训练，测试。**

**其中需要关注的就是DNN.py这个单元，如果需要更换模型的话，直接修改dnn.create\_DNN，重新修改模型即可，其他的无需去改动。CreateDNN()函数如下：**

