# 设计部署文档

## 一、项目设计

#### 1.1 任务解析

##### 1.1.1 任务要求

本小组的任务是实现“方案二：爆破限制绕过”。具体地，我们需要：构建一个具有若干用户的网站，网站登录验证需采用一种验证码限制方案。构造一份口令字典用于爆破，口令个数不少于10000个，其中正确的口令在末尾。

##### 1.1.2 实现简析

本小组根据方案二的要求，针对验证码的生成端（客户端/服务端）分别实现了两套爆破方案：

**客户端生成验证码**

客户端生成验证码，即验证码的生成和验证均在用户浏览器中实现。针对这类验证码认证方式，爆破人员可以分析网页的javascript代码，在网页的源码中必定能够找到正确的验证码从而绕过验证；

**服务端生成验证码**

服务端生成验证码，即验证码的生成和验证均在网站部署的服务器中实现。服务端生成验证码后，以session记录验证码值，如果服务端同时设置session的过期时间并且没有其他逻辑错误，爆破人员很难通过暴力破解的方式绕过验证码。我们使用tensorflow训练了一个CNN模型识别验证码中的数字从而绕过验证。

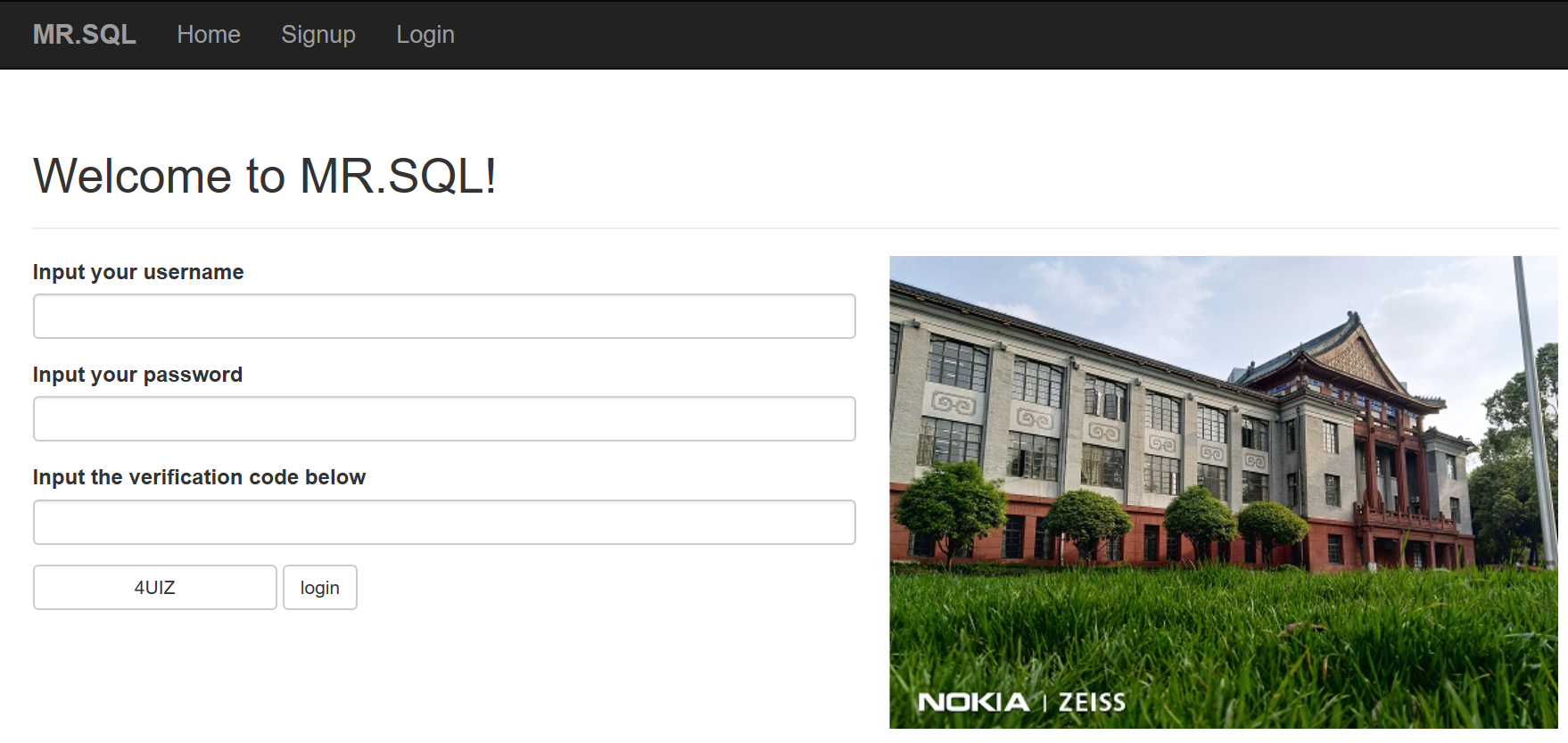
#### 1.2 具体实现

##### 1.2.1 Task One

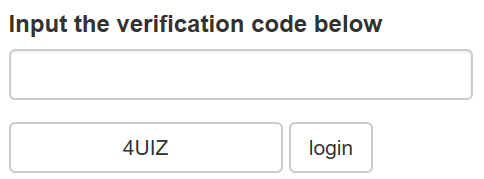
该任务的任务目标为：1. 搭建一个Web服务，前端生成4位验证码（数字和字母的随机组合）并验证；2.绕过前端JavaScript的验证码检测多线程暴力破解指定用户的口令。具体实现如下：

**Web服务搭建**：

* 使用Flask搭建了一个简易的Web服务，仅含有登录功能，网页截图如下所示；

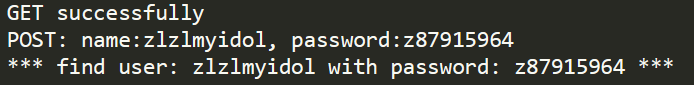


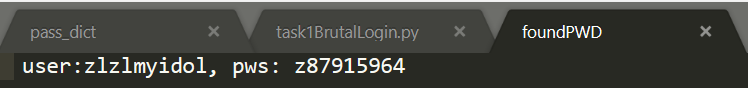
* 使用bootstrap提供的网页样式；
* 使用sqlite3存储用户名和密码；
* 在网页前端用JavaScript实现4位验证码（数字和字母的随机组合）的生成，每次刷新网页时重新生成，并在前端验证输入的verification code和正确的验证码，如下：



**爆破限制绕过**：

* 将过去泄露的用户密码合成了一份静态的密码爆破字典；
* 使用 requests 模块对网站登录界面进行请求；
* 使用 b4 模块对响应内容提取，获得当前 CSRF 值；
* 使用同一 session构造请求，指定username、password、scrf\_token、code、verification、submit=’login’，其中code、verification分别代表输入的验证码和正确的验证码，因请求不经过JS判断，故这两个键值任意；
* 逐行读取字典文件，并利用多线程实现POST暴力破解口令，将正确的口令保存到foundPWD文件，控制台输出和保存文件如下图：





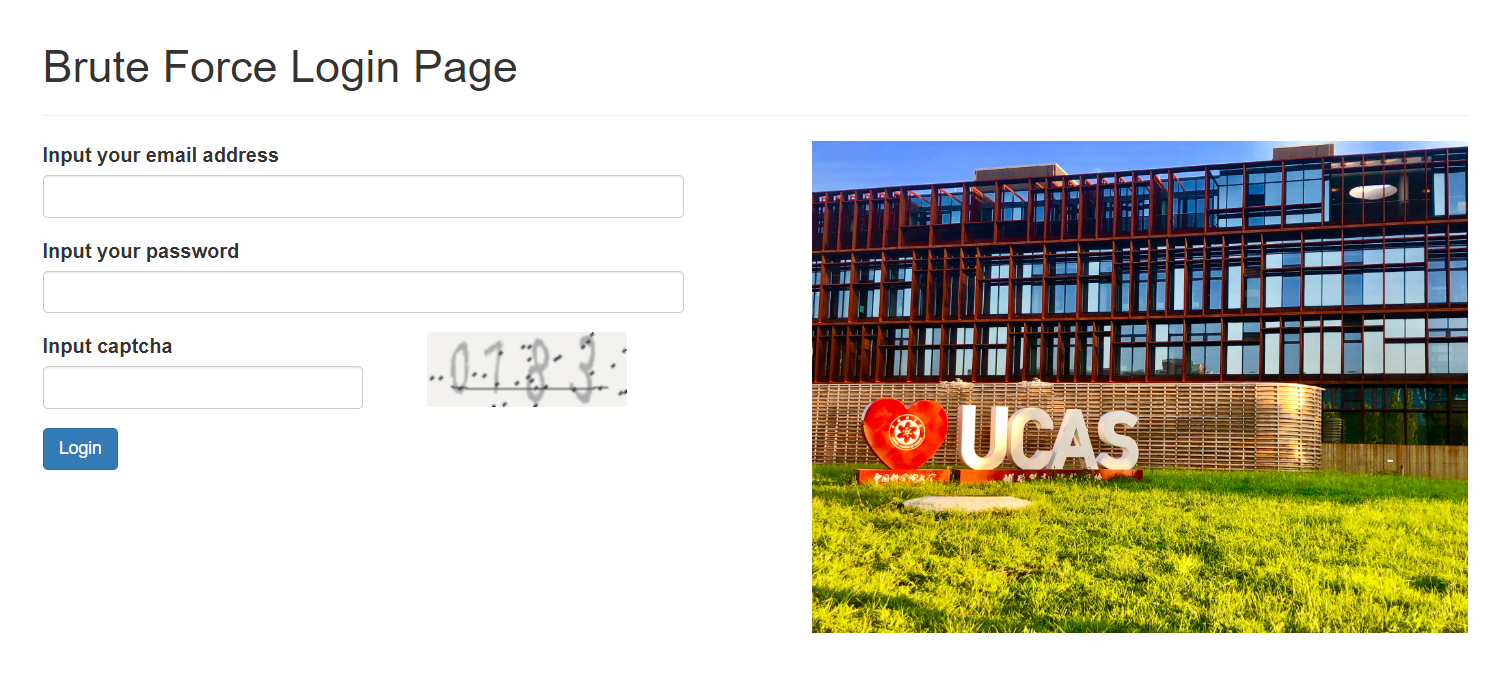
* 效率：在测试机上（处理器：Intel i5-8265U，内存8G），利用有着50000条口令的字典文件（正确口令在末尾），爆破时间约400s，相比逐行爆破节省了12.5%的时间。

##### 1.2.2 Task Two

该任务的任务目标为：1. 搭建一个Web服务，后端生成图片验证码；2. 使用深度学习识别图片验证码进行绕过。具体的实现如下所述：

**Web服务搭建**：

* 使用Flask搭建了一个简易的Web服务，仅含有登录功能，网页截图如下所示；



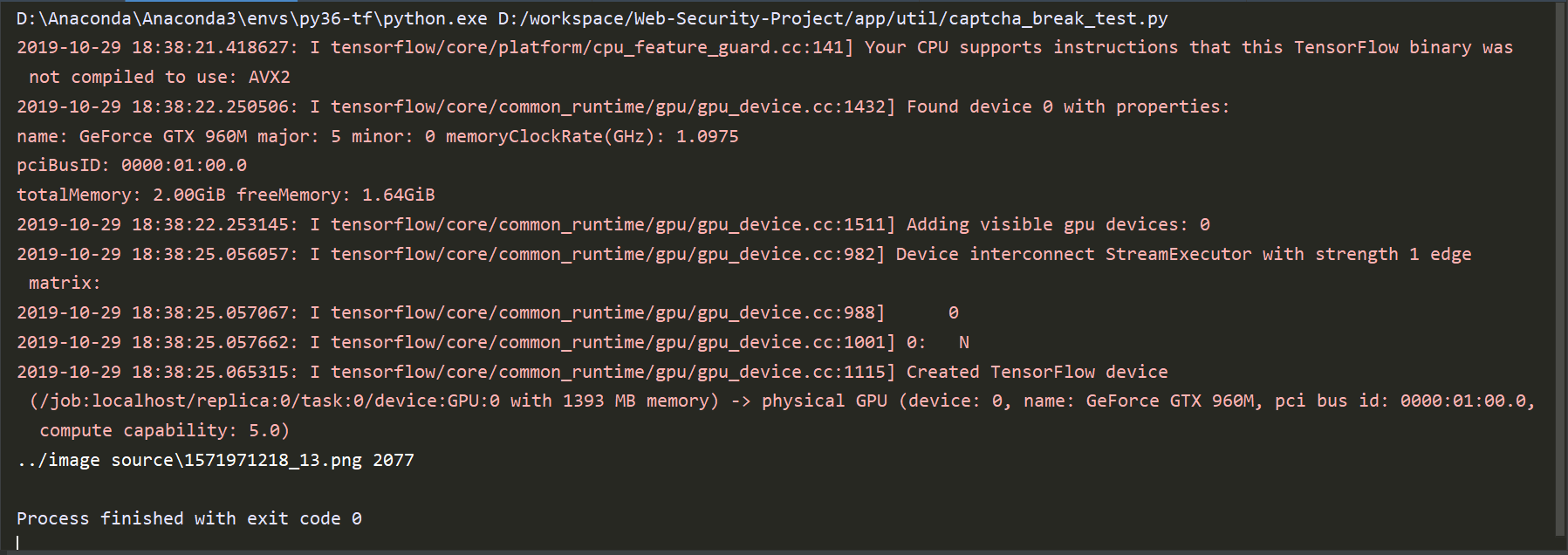
* 使用bootstrap提供的网页样式；
* 使用sqlite3存储用户名和密码；
* 使用python captcha模块随机生成长度为4、只包含数字的图片验证码，示例验证码如下图所示；



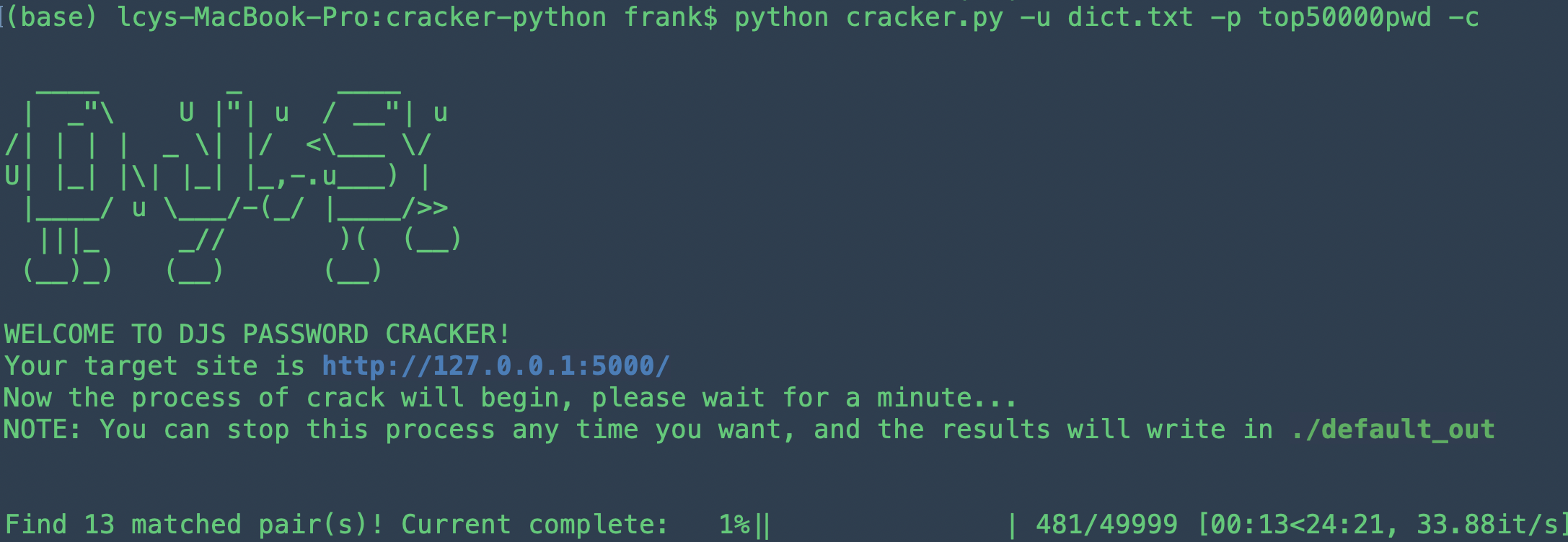
* 使用Session管理用户当前的验证码，每次请求后都将刷新验证码，并且Session的过期时间设置为2分钟；

**爆破限制绕过**：

* 借助tensorflow训练了一个5层的CNN模型，得到的模型识别准确率为99.375%，实例验证码的识别结果如下所示；



* 将过去泄露的用户密码合成了一份静态的密码爆破字典；
* 使用 requests 模块对网站登录界面进行请求；
* 使用 b4 模块对响应内容提取，获得当前 CSRF 值；
* 使用同一 session 请求验证码图片，写入文件；
* 使用训练出来的 CNN 模型对验证码进行识别，之后进行爆破，爆破过程如下图所示；



## 二、部署

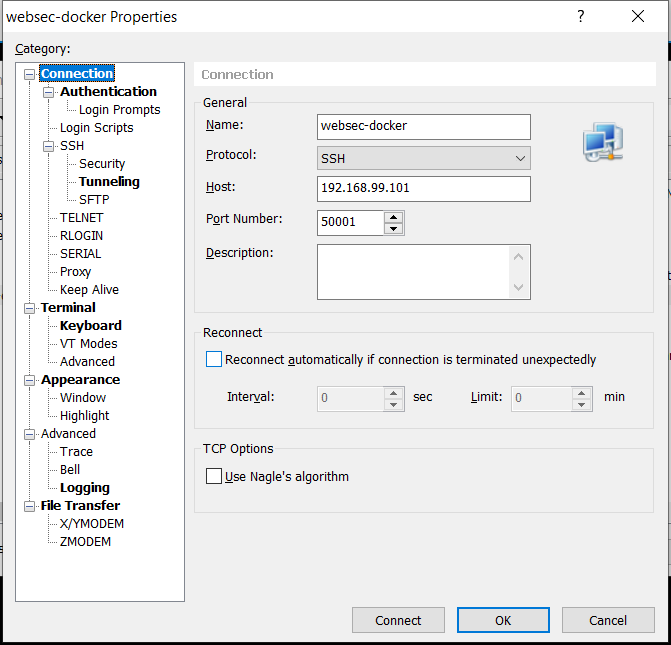
将打包的容器导入docker：

# 导入docker  
cat websec.tar | docker import - websec/ubuntu:14.04  
# 查看docker当前所有的镜像  
docker images

启动容器（假设容器的ID为CONTAINER\_ID），并且配置docker的50001端口映射到容器的22端口，开启ssh服务（我们尝试着让ssh服务自启，当时失败了，索性这可以在启动容器后通过一句简单的命令启动）：

# 第一次启动容器  
docker run -it -p 50001:22 websec/ubuntu:14.04 /bin/bash  
service ssh start  
# 已经创建过容器  
docker start CONTAINER\_ID  
docker attach CONTAINER\_ID  
service ssh start

现在，可以通过ssh（用户名为root，密码为123456）连接容器并执行命令，假设docker的虚拟ip地址为192.168.99.101，下面为xshell的连接配置截图：



项目的根目录为：

~/Web-Security-Project/

开启task1的Web服务：

cd ~/Web-Security-Project/app/  
python task1.py

开始task1的爆破过程：

cd ~/Web-Security-Project/cracker-python/  
python task1BrutalLogin.py

开启task2的Web服务：

cd ~/Web-Security-Project/app/  
python task2.py

开始task2的爆破过程：

cd ~/Web-Security-Project/cracker-python/  
python cracker-python/cracker.py *--site*="http://127.0.0.1:5000/" *--user*=cracker-python/user.txt *--pass*=cracker-python/top50000pwd -c