

NSD SECURITY DAY03

1. [案例1：部署audit监控文件](#)
2. [案例2：加固常见服务的安全](#)
3. [案例3：使用diff和patch工具打补丁](#)

1 案例1：部署audit监控文件

1.1 问题

本案例要求熟悉audit审计工具的基本使用，完成以下任务操作：

1. 使用audit监控/etc/ssh/sshd_config
2. 当该文件发生任何变化即记录日志
3. 通过手动和ausearch工具查看日志内容

1.2 方案

审计的目的是基于事先配置的规则生成日志，记录可能发生在系统上的事件（正常或非正常行为的事件），审计不会为系统提供额外的安全保护，但她会发现并记录违反安全策略的人及其对应的行为。

审计能够记录的日志内容：

- a) 日期与事件以及事件的结果
- b) 触发事件的用户
- c) 所有认证机制的使用都可以被记录，如ssh等
- d) 对关键数据文件的修改行为等都可以被记录

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：配置audit审计系统

1) 安装软件包，查看配置文件（确定审计日志的位置）

```
01. [root@proxy ~]# yum -y install audit //安装软件包
02. [root@proxy ~]# cat /etc/audit/auditd.conf //查看配置文件，确定日志位置
03. log_file = /var/log/audit/audit.log //日志文件路径
04. [root@proxy ~]# systemctl start auditd //启动服务
05. [root@proxy ~]# systemctl enable auditd //设置开机自启
```

2) 配置审计规则

可以使用auditctl命令控制审计系统并设置规则决定哪些行为会被记录日志。

[Top](#)

语法格式如下：

- 01. [root@proxy ~] # auditctl - s //查询状态
- 02. [root@proxy ~] # auditctl - l //查看规则
- 03. [root@proxy ~] # auditctl - D //删除所有规则

定义临时文件系统规则：

- 01. #语法格式：auditctl -w path -p permission -k key_name
- 02. # path为需要审计的文件或目录
- 03. # 权限可以是r,w,x,a(文件或目录的属性发生变化)
- 04. # Key_name为可选项，方便识别哪些规则生成特定的日志项
- 05.
- 06. [root@proxy ~] # auditctl -w /etc/passwd -p wa -k passwd_change
- 07. //设置规则所有对passwd文件的写、属性修改操作都会被记录审计日志
- 08. [root@proxy ~] # auditctl -w /etc/selinux/ -p wa -k selinux_change
- 09. //设置规则，监控/etc/selinux目录
- 10. [root@proxy ~] # auditctl -w /usr/sbin/fdisk -p x -k disk_partition
- 11. //设置规则，监控fdisk程序
- 12.
- 13. [root@proxy ~] # auditctl -w /etc/ssh/sshd_conf -p warx -k sshd_config
- 14. //设置规则，监控sshd_conf文件

如果需要创建永久审计规则，则需要修改规则配置文件：

- 01. [root@proxy ~] # vim /etc/audit/rules.d/audit.rules
- 02. -w /etc/passwd -p wa -k passwd_changes
- 03. -w /usr/sbin/fdisk -p x -k partition_disks

步骤二：查看并分析日志

1) 手动查看日志

查看SSH的主配置文件/etc/ssh/sshd_conf，查看audit日志信息：

- 01. [root@proxy ~] # tailf /var/log/audit/audit.log
- 02. type=SYSCALL msg=audit(1517557590.644: 229228) : arch=c000003e
- 03. syscall=2 success=yes exit=3
- 04. a0=7fff71721839 a1=0 a2=1fffffffffff0000 a3=7fff717204c0
- 05. items=1 ppid=7654 pid=7808 auid=0 uid=0 gid=0 euid=0 suid=0
- 06. fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts2 ses=3 comm="cat"

[Top](#)

```

07.  exe="/usr/bin/cat"
08.  subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="sshd_config"
09.  .. ..
10.  #内容分析
11.  # type为类型
12.  # msg为( time_stamp: ID) , 时间是date +%s (1970-1-1至今的秒数)
13.  # arch=c000003e, 代表x86_64 (16进制)
14.  # success=yes/no, 事件是否成功
15.  # a0-a3是程序调用时前4个参数, 16进制编码了
16.  # ppid父进程ID, 如bash, pid进程ID, 如cat命令
17.  # auid是审核用户的id, su - test, 依然可以追踪su前的账户
18.  # uid, gid用户与组
19.  # tty: 从哪个终端执行的命令
20.  # comm="cat"      用户在命令行执行的指令
21.  # exe="/bin/cat"   实际程序的路径
22.  # key="sshd_config"  管理员定义的策略关键字key
23.  # type=CWD        用来记录当前工作目录
24.  # cwd="/home/username"
25.  # type=PATH
26.  # ouid( owner's user id)   对象所有者id
27.  # guid( owner's groupid)   对象所有者id

```

2) 通过工具搜索日志

系统提供的ausearch命令可以方便的搜索特定日志, 默认该程序会搜索/var/log/audit/audit.log, ausearch options -if file_name可以指定文件名。

```

01.  [root@proxy ~]# ausearch -k sshd_config -i
02.  //根据key搜索日志, -选项表示以交互式方式操作

```

2 案例2 : 加固常见服务的安全

2.1 问题

本案例要求优化提升常见网络服务的安全性, 主要完成以下任务操作:

1. 优化Nginx服务的安全配置
2. 优化MySQL数据库的安全配置
3. 优化Tomcat的安全配置

2.2 方案

[Top](#)

Nginx安全优化包括: 删除不要的模块、修改版本信息、限制并发、拒绝非法请求、防止buffer溢出。

MySQL安全优化包括：初始化安全脚本、密码安全、备份与还原、数据安全。

Tomcat安全优化包括：隐藏版本信息、降权启动、删除默认测试页面。

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：优化Nginx服务的安全配置

1) 删除不需要的模块

Nginx是模块化设计的软件，需要什么功能与模块以及不需要哪些模块，都可以在编译安装软件时自定义，使用--with参数可以开启某些模块，使用--without可以禁用某些模块。最小化安装永远都是对的方案！

下面是禁用某些模块的案例：

```
01. [ root@proxy ~] # tar -xvf nginx-1.12.tar.gz
02. [ root@proxy ~] # cd nginx-1.12
03. [ root@proxy nginx-1.12] # ./configure \
04. > -without-http_autoindex_module \           //禁用自动索引文件目录模块
05. > -without-http_ssi_module
06. [ root@proxy nginx-1.12] # make
07. [ root@proxy nginx-1.12] # make install
```

2) 修改版本信息，并隐藏具体的版本号

默认Nginx会显示版本信息以及具体的版本号，这些信息给攻击者带来了便利性，便于他们找到具体版本的漏洞。

如果需要屏蔽版本号信息，执行如下操作，可以隐藏版本号。

```
01. [ root@proxy ~] # vim /usr/local/nginx/conf/nginx.conf
02. ... ..
03. http{
04.     server_tokens off;           //在http下面手动添加这么一行
05.     ... ..
06. }
07. [ root@proxy ~] # nginx -s reload
08.
09. [ root@proxy ~] # curl -I http://192.168.4.5 //查看服务器响应的头部信息
```

但服务器还是显示了使用的软件为nginx，通过如下方法可以修改该信息。

[Top](#)

```
01. [ root@proxy nginx-1.12] # vim +48 src/http/ngx_http_header_filter_module.c
```

```

02. //注意：vim这条命令必须在nginx-1.12源码包目录下执行！！！！！！
03. //该文件修改前效果如下：
04. static u_char ngx_http_server_string[] = "Server: nginx" CRLF;
05. static u_char ngx_http_server_full_string[] = "Server: " NGINX_VER CRLF;
06. static u_char ngx_http_server_build_string[] = "Server: " NGINX_VER_BUILD CRLF;
07. //下面是我们修改后的效果：
08. static u_char ngx_http_server_string[] = "Server: Jacob" CRLF;
09. static u_char ngx_http_server_full_string[] = "Server: Jacob" CRLF;
10. static u_char ngx_http_server_build_string[] = "Server: Jacob" CRLF;
11.
12. //修改完成后，再去编译安装Nginx，版本信息将不再显示为Nginx，而是Jacob
13. [ root@proxy nginx-1.12] # ./configure
14. [ root@proxy nginx-1.12] # make && make install
15. [ root@proxy nginx-1.12] # killall nginx
16. [ root@proxy nginx-1.12] # /usr/local/nginx/sbin/nginx //启动服务
17.
18. [ root@proxy nginx-1.12] # curl -I http://192.168.4.5 //查看版本信息验证

```

3) 限制并发量

DDOS攻击者会发送大量的并发连接，占用服务器资源（包括连接数、带宽等），这样会导致正常用户处于等待或无法访问服务器的状态。

Nginx提供了一个ngx_http_limit_req_module模块，可以有效降低DDOS攻击的风险，操作方法如下：

```

01. [ root@proxy ~] # vim /usr/local/nginx/conf/nginx.conf
02. ... ..
03. http{
04. ... ..
05. limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
06.     server {
07.         listen 80;
08.         server_name localhost;
09.         limit_req zone=one burst=5;
10.     }
11. }
12.
13. //备注说明：
14. //limit_req_zone语法格式如下：
15. //limit_req_zone key zone=name: size rate=rate;
16. //上面案例中是将客户端IP信息存储名称为one的共享内存，内存空间为10M

```

[Top](#)

17. //1M可以存储8千个IP信息，10M可以存储8万个主机连接的状态，容量可以根据需要任意
18. //每秒中仅接受1个请求，多余的放入漏斗
19. //漏斗超过5个则报错
20. [root@proxy ~] # /usr/local/nginx/sbin/nginx -s reload

客户端使用ab测试软件测试效果：

```
01. [ root@client ~] # ab -c 100 -n 100 http://192.168.4.5/
```

4) 拒绝非法的请求

网站使用的是HTTP协议，该协议中定义了很多方法，可以让用户连接服务器，获得需要的资源。但实际应用中一般仅需要get和post。

具体HTTP请求方法的含义如表-1所示。

表-1 HTTP请求方法及含义

请求方法	功能描述
GET	请求指定的页面信息，并返回实体主体
HEAD	类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头
POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）
DELETE	请求服务器删除指定的页面
PUT	向服务器特定位置上传资料
...	其他

未修改服务器配置前，客户端使用不同请求方法测试：

- ```
01. [root@client ~] # curl -i -X GET http://192.168.4.5 //正常
02. [root@client ~] # curl -i -X HEAD http://192.168.4.5 //正常
03. //curl命令选项说明：
04. //- 选项：访问服务器页面时，显示HTTP的头部信息
05. //- X选项：指定请求服务器的方法
```

通过如下设置可以让Nginx拒绝非法的请求方法：

- ```
01. [ root@proxy ~] # vim /usr/local/nginx/conf/nginx.conf
02. http{
03.     server {
04.         listen 80;
05.         #这里，!符号表示对正则取反，~符号是正则匹配符号
```

[Top](#)

```

06.    #如果用户使用非GET或POST方法访问网站，则return返回444的错误信息
07.        if ( $request_method !~ ^( GET| POST) $ ) {
08.            return 444;
09.        }
10.    }
11. }
12. [ root@proxy ~] # /usr/local/nginx/sbin/nginx -s reload

```

修改服务器配置后，客户端使用不同请求方法测试：

```

01. [ root@client ~] # curl -i -X GET http://192.168.4.5 //正常
02. [ root@client ~] # curl -i -X HEAD http://192.168.4.5 //报错

```

4) 防止buffer溢出

当客户端连接服务器时，服务器会启用各种缓存，用来存放连接的状态信息。

如果攻击者发送大量的连接请求，而服务器不对缓存做限制的话，内存数据就有可能溢出（空间不足）。

修改Nginx配置文件，调整各种buffer参数，可以有效降低溢出风险。

```

01. [ root@proxy ~] # vim /usr/local/nginx/conf/nginx.conf
02. http{
03.     client_body_buffer_size 1k;
04.     client_header_buffer_size 1k;
05.     client_max_body_size 1k;
06.     large_client_header_buffers 2 1k;
07.     ... ..
08. }
09. [ root@proxy ~] # /usr/local/nginx/sbin/nginx -s reload

```

步骤二：数据库安全

1) 初始化安全脚本

安装完MariaDB或MySQL后，默认root没有密码，并且提供了一个任何人都可以操作的test测试数据库。有一个名称为mysql_secure_installation的脚本，该脚本可以帮助我们为root设置密码，并禁止root从远程其他主机登陆数据库，并删除测试性数据库test。

```

01. [ root@proxy ~] # systemctl status mariadb
02. //确保服务已启动

```

[Top](#)

03. [root@proxy ~] # mysql_secure_installation
04. //执行初始化安全脚本

2) 密码安全

手动修改MariaDB或MySQL数据库密码的方法：

01. [root@proxy ~] # mysqladmin - uroot - predhat password 'mysql'
02. //修改密码，旧密码为redhat，新密码为mysql
03. [root@proxy ~] # mysql - uroot - pmysql
04. MariaDB [(none)] > set password for root@'localhost' = password('redhat');
05. //使用账户登录数据库，修改密码
06. MariaDB [(none)] > select user, host, password from mysql.user;
07. +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
08. | user | host | password |
09. +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
10. | root | localhost | *84BB5DF4823DA 319BBF86C99624479A 198E6EEE9 |
11. | root | 127.0.0.1 | *84BB5DF4823DA 319BBF86C99624479A 198E6EEE9 |
12. | root | ::1 | *84BB5DF4823DA 319BBF86C99624479A 198E6EEE9 |
13. +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

修改密码成功，而且密码在数据库中是加密的，有什么问题吗？问题是你的密码被明文记录了，下面来看看明文密码：

01. [root@proxy ~] # cat .bash_history
02. mysqladmin - uroot - pxxx password 'redhat'
03. //通过命令行修改的密码，bash会自动记录历史，历史记录中记录了明文密码
- 04.
05. [root@proxy ~] # cat .mysql_history
06. set password for root@'localhost' = password('redhat');
07. select user, host, password from mysql.user;
08. flush privileges;
09. //通过mysql命令修改的密码，mysql也会有所有操作指令的记录，这里也记录了明文密码

另外数据库还有一个binlog日志里也有明文密码（5.6版本后修复了）。

怎么解决？

管理好自己的历史，不使用明文登录，选择合适的版本5.6以后的版本，

[Top](#)

日志，行为审计（找到行为人），使用防火墙从TCP层设置ACL（禁止外网接触数据库）。

3) 数据备份与还原

首先，备份数据库（注意用户名为root，密码为redhat）：

01. [root@proxy ~] # mysqldump -uroot -predhat my db table > table.sql
02. //备份数据库中的某个数据表
03. [root@proxy ~] # mysqldump -uroot -predhat my db > my db.sql
04. //备份某个数据库
05. [root@proxy ~] # mysqldump -uroot -predhat -- all- databases > all.sql
06. //备份所有数据库

接下来，还原数据库（注意用户名为root，密码为redhat）：

01. [root@proxy ~] # mysql -uroot -predhat my db < table.sql //还原数据表
02. [root@proxy ~] # mysql -uroot -predhat my db < my db.sql //还原数据库
03. [root@proxy ~] # mysql -uroot -predhat < all.sql //还原所有数据库

4) 数据安全

在服务器上（192.168.4.5），创建一个数据库账户：

01. [root@proxy ~] # mysql -uroot -predhat
02. //使用管理员，登陆数据库
03. MariaDB [(none)]> grant all on *.* to tom@'%' identified by '123';
04. //创建一个新账户tom

使用tcpdump抓包（192.168.4.5）

01. [root@proxy ~] # tcpdump -w log -i any src or dst port 3306
02. //抓取源或目标端口是3306的数据包，保存到log文件中

客户端（192.168.4.100）从远程登陆数据库服务器（192.168.4.5）

01. [root@client ~] # mysql -utom -p123 -h 192.168.4.5
02. //在192.168.4.100这台主机使用mysql命令登陆远程数据库服务器（192.168.4.5）
03. //用户名为tom，密码为123
04. MariaDB [(none)]> select * from mysql.user;

[Top](#)

05. //登陆数据库后，任意执行一条查询语句

回到服务器查看抓取的数据包

01. [root@proxy ~] # tcpdump -A -r log

02. //使用tcpdump查看之前抓取的数据包，很多数据库的数据都明文显示出来

如何解决？

可以使用SSH远程连接服务器后，再从本地登陆数据库（避免在网络中传输数据，因为网络环境中不知道有没有抓包者）。

或者也可以使用SSL对MySQL服务器进行加密，类似与HTTP+SSL一样，MySQL也支持SSL加密（确保网络中传输的数据是被加密的）。

步骤三：Tomcat安全性

1）隐藏版本信息、修改tomcat主配置文件（隐藏版本信息）

未修改版本信息前，使用命令查看服务器的版本信息

注意：proxy有192.168.2.5的IP地址，这里使用proxy作为客户端访问192.168.2.100服务器。

01. [root@proxy ~] # curl -I http://192.168.2.100:8080/xx

02. //访问不存在的页面文件，查看头部信息

03. [root@proxy ~] # curl -I http://192.168.2.100:8080

04. //访问存在的页面文件，查看头部信息

05. [root@proxy ~] # curl http://192.168.2.100:8080/xx

06. //访问不存在的页面文件，查看错误信息

修改tomcat配置文件，修改版本信息(在192.168.2.100操作)：

01. [root@web1 tomcat] # yum -y install java-1.8.0-openjdk-devel

02. [root@web1 tomcat] # cd /usr/local/tomcat/lib/

03. [root@web1 lib] # jar -xf catalina.jar

04. [root@web1 lib] # vim org/apache/catalina/util/ServerInfo.properties

05. //根据自己的需要，修改版本信息的内容

06. [root@web1 lib] # /usr/local/tomcat/bin/shutdown.sh //关闭服务

07. [root@web1 lib] # /usr/local/tomcat/bin/startup.sh //启动服务

[Top](#)

修改后，客户端再次查看版本信息（在192.168.2.5操作）：

01. [root@proxy ~] # curl -I http://192.168.2.100:8080/xx
02. //访问不存在的页面文件，查看头部信息
03. [root@proxy ~] # curl -I http://192.168.2.100:8080
04. //访问存在的页面文件，查看头部信息
05. [root@proxy ~] # curl http://192.168.2.100:8080/xx
06. //访问不存在的页面文件，查看错误信息

再次修改tomcat服务器配置文件，修改版本信息，手动添加server参数（在192.168.2.100操作）：

01. [root@web1 lib] # vim /usr/local/tomcat/conf/server.xml
02. <Connector port="8080" protocol="HTTP/1.1"
03. connectionTimeout="20000" redirectPort="8443" server="jacob" />
04. [root@web1 lib] # /usr/local/tomcat/bin/shutdown.sh //关闭服务
05. [root@web1 lib] # /usr/local/tomcat/bin/startup.sh //启动服务

修改后,客户端再次查看版本信息（在192.168.2.5操作）：

01. [root@proxy ~] # curl -I http://192.168.2.100:8080/xx
02. //访问不存在的页面文件，查看头部信息
03. [root@proxy ~] # curl -I http://192.168.2.100:8080
04. //访问存在的页面文件，查看头部信息
05. [root@proxy ~] # curl http://192.168.2.100:8080/xx
06. //访问不存在的页面文件，查看错误信息

2) 降级启动

默认tomcat使用系统高级管理员账户root启动服务，启动服务尽量使用普通用户。

01. [root@web1 ~] # useradd tomcat
02. [root@web1 ~] # chown -R tomcat:tomcat /usr/local/tomcat/
03. //修改tomcat目录的权限，让tomcat账户对该目录有操作权限
04. [root@web1 ~] # su -c /usr/local/tomcat/bin/startup.sh tomcat
05. //使用su命令切换为tomcat账户，以tomcat账户的身份启动tomcat服务
06. [root@web1 ~] # chmod +x /etc/rc.local //该文件为开机启动文件
07. [root@web1 ~] # vim /etc/rc.local //修改文件，添加如下内容[Top](#)
08. su -c /usr/local/tomcat/bin/startup.sh tomcat

3) 删除默认测试页面

```
01. [root@web1 ~] # rm -rf /usr/local/tomcat/webapps/*
```

3 案例3：使用diff和patch工具打补丁

3.1 问题

本案例要求优化提升常见网络服务的安全性，主要完成以下任务操作：

1. 使用diff对比文件差异
2. 使用diff生成补丁文件
3. 使用patch命令为旧版本打补丁

3.2 方案

程序是人设计出来的，总是会有这样那样的问题与漏洞，目前的主流解决方法就是为有问题的程序打补丁，升级新版本。

在Linux系统中diff命令可以为我们生成补丁文件，然后使用patch命令为有问题的程序代码打补丁。

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：对比单个文件差异

- 1) 编写两个版本的脚本，一个为v1版本，一个为v2版本。

```
01. [root@proxy ~] # cat test1.sh //v 1版本脚本
02. #!/bin/bash
03. echo "hello wrld"
04.
05. [root@proxy ~] # cat test2.sh //v 2版本脚本
06. #!/bin/bash
07. echo "hello the world"
08. echo "test file"
```

- 2) 使用diff命令语法

使用diff命令查看不同版本文件的差异。

```
01. [root@proxy ~] # diff test1.sh test2.sh //查看文件差异
02. @@ -1,3 +1,3 @@
03. #!/bin/bash
```

[Top](#)

```

04. - echo "hello world"
05. - echo "test"
06. +echo "hello the world"
07. +echo "test file"
08.
09. [ root@proxy ~] # diff -u test1.sh test2.sh //查看差异，包含头部信息
10. --- test1.sh 2018-02-07 22:20:02.723971251 +0800
11. +++ test2.sh 2018-02-07 22:20:13.358760687 +0800
12. @@ -1,3 +1,3 @@
13.  #! /bin/bash
14. - echo "hello world"
15. - echo "test"
16. +echo "hello the world"
17. +echo "test file"

```

diff制作补丁文件的原理：告诉我们怎么修改第一个文件后能得到第二个文件。

这样如果第一个版本的脚本有漏洞，我们不需要将整个脚本都替换，仅需要修改有问题的一小部分代码即可，diff刚好可以满足这个需求！

像Linux内核这样的大块头，一旦发现有一个小漏洞，我们不可能把整个内核都重新下载，全部替换一遍，而仅需要更新有问题的那一小部分代码即可！

diff命令常用选项：

-u 输出统一内容的头部信息（打补丁使用），计算机知道是哪个文件需要修改

-r 递归对比目录中的所有资源（可以对比目录）

-a 所有文件视为文本（包括二进制程序）

-N 无文件视为空文件（空文件怎么变成第二个文件）

-N选项备注说明：

A目录下没有txt文件，B目录下有txt文件

diff比较两个目录时，默认会提示txt仅在B目录有（无法对比差异，修复文件）

diff比较时使用N选项，则diff会拿B下的txt与A下的空文件对比，补丁信息会明确说明如何从空文件修改后变成txt文件，打补丁即可成功！

步骤二：使用patch命令对单文件代码打补丁

1) 准备实验环境

```

01. [ root@proxy ~] # cd demo
02. [ root@proxy demo] # vim test1.sh
03.  #! /bin/bash
04.  echo "hello world"
05.  echo "test"
06. [ root@proxy demo] # vim test2.sh

```

[Top](#)

```

07.  #! /bin/bash
08.  echo "hello the world"
09.  echo "test file"

```

2) 生成补丁文件

```
01. [root@proxy demo] # diff -u test1.sh test2.sh > test.patch
```

3) 使用patch命令打补丁

在代码相同目录下为代码打补丁

```

01. [root@proxy demo] # yum -y install patch
02. [root@proxy demo] # patch -p0 < test.patch           //打补丁
03. patching file test1.sh
04. //patch -pnum (其中num为数字,指定删除补丁文件中多少层路径前缀)
05. //如原始路径为/u/howard/src/blurfl/blurfl.c
06. //- p0则整个路径不变
07. //- p1则修改路径为u/howard/src/blurfl/blurfl.c
08. //- p4则修改路径为blurfl/blurfl.c
09. //- R(reverse) 反向修复, -E修复后如果文件为空,则删除该文件
10. [root@proxy demo] # patch -RE < test.patch          //还原旧版本,反向修复

```

步骤三：对比目录中所有文件的差异

1) 准备实验环境

```

01. [root@proxy ~] # mkdir demo
02. [root@proxy ~] # cd demo
03. [root@proxy demo] # mkdir {source1,source2}
04.
05. [root@proxy demo] # echo "hello world" > source1/test.sh
06. [root@proxy demo] # cp /bin/find source1/
07. [root@proxy demo] # tree source1/                  //source1目录下2个文件
08. |-- find
09. `-- test.sh
10.
11. [root@proxy demo] # echo "hello the world" > source2/test.sh
12. [root@proxy demo] # echo "test" > source2/tmp.txt

```

[Top](#)

```

13. [root@proxy demo] # cp /bin/find source2/
14. [root@proxy demo] # echo "1" >> source2/find
15. [root@proxy demo] # tree source2/           //source1目录下3个文件
16. |-- find
17. |-- test.sh
18. `-- tmp.txt
19. //注意：两个目录下find和test.sh文件内容不同，source2有tmp.txt而source1没有该文件

```

2) 制作补丁文件

```

01. [root@proxy demo] # diff -u source1/ source2/
02. //仅对比了文本文件test.sh；二进制文件、tmp都没有对比差异，仅提示，因为没有-a和
03. [root@proxy demo] # diff -Nu source1/ source2/
04. //对比了test.sh，并且使用source2目录的tmp.txt与source1的空文件对比差异。
05. [root@proxy demo] # diff -Nua source1/ source2/
06. //对比了test.sh、tmp.txt、find(程序)。

```

步骤四：使用patch命令对目录下的所有代码打补丁

1) 使用前面创建的source1和source2目录下的代码为素材，生成补丁文件

```

01. [root@proxy ~] # cd demo
02. [root@proxy demo] # diff -Nua source1/ source2/ > source.patch

```

2) 使用patch命令为代码打补丁

```

01. [root@proxy demo] # ls
02. source1 source2 source.patch
03. [root@proxy demo] # cat source.patch           //对比的文件有路径信息
04. --- source1/test.sh 2018-02-07 22:51:33.034879417 +0800
05. +++ source2/test.sh 2018-02-07 22:47:32.531754268 +0800
06. @@ -1+1 @@
07. - hello world
08. +hello the world
09. [root@proxy demo] # cd source1
10. [root@proxy source1] # patch -p1 < ../source.patch

```

[Top](#)

[Top](#)