SQL 注入攻击实验

1 概述

SQL 注入是一种利用 Web 应用程序与数据库交互的漏洞而进行的代码注入技术。这种漏洞是由于用户的输入在发送到后端数据库之前没有被正确检查导致的。

许多 Web 应用程序都会从用户那里获取输入，并使用这些输入来构造 SQL 查询，以便从数据库中获取信息。Web 应用程序也使用 SQL 将信息存储在数据库中。这些是 Web 应用程序开发中的常见做法。当 SQL 的查询构建时不小心的话，可能会出现 SQL 注入漏洞。SQL 注入是针对 Web 应用程序最常见的攻击之一。

在这个实验中，我们创建了一个易受 SQL 注入攻击的 Web 应用程序。学生的目标是利用程序中的漏洞来做 SQL 注入攻击，同时学习防御此类攻击的技术。本实验涵盖了以下主题：

- SQL 语句：SELECT 和 UPDATE 语句
- SQL 注入
- Prepared 语句 (防御方法)

阅读材料 关于 SQL 注入的详细覆盖可以在以下章节中找到：

- 《SEED 书籍》中的第 12 章，*Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.

实验环境。 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。既然我们使用容器来建立实验环境，本实验不太依赖 SEED VM。您可以在其他 VM、物理机器以及云端 VM 上进行此实验。

2 实验环境

我们为这个实验开发了一个 Web 应用程序，并且使用容器来搭建实验环境。实验中包含两个容器，一个用于托管 Web 应用程序，另一个用于托管 Web 应用程序的数据库。Web 应用程序容器的 IP 地址是 10.9.0.5，它的 URL 如下：

```
http://www.seed-server.com
```

我们需要将此主机名映射到容器的 IP 地址。请在 `/etc/hosts` 文件中添加以下条目。您需要使用 root 权限来更改此文件（使用 `sudo`）。需要注意的是，由于某些其他实验的原因，这个名称可能已经添加到了文件中。如果它被映射到不同的 IP 地址，则必须删除旧的条目。

```
10.9.0.5        www.seed-server.com
```

2.1 容器设置与命令

请从实验的网站下载 Labsetup.zip 文件到你的 VM 中，解压它，进入 Labsetup 文件夹，然后用 docker-compose.yml 文件安装实验环境。对这个文件及其包含的所有 Dockerfile 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的，那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

MySQL 数据库。 容器通常是一次性的，因此一旦销毁，容器内的所有数据都会丢失。对于本实验，我们确实希望将数据保留在 MySQL 数据库中，这样在关闭容器时就不会丢失数据。为了实现这一点，我们将主机上的 `mysql_data` 文件夹挂载到 MySQL 容器内的 `/var/lib/mysql` 文件夹（此文件夹是 MySQL 存储数据库的地方）。因此，即使容器被销毁，数据库中的数据仍会保留。这个 `mysql_data` 文件夹在 `Labsetup` 内，它将在 MySQL 容器第一次运行后创建。如果您确实想从空的数据库开始，可以删除此文件夹：

```
$ sudo rm -rf mysql_data
```

2.2 关于 Web 应用程序

我们创建了一个简单的员工管理应用程序。通过此 Web 应用程序，员工可以查看和更新其个人资料信息。在该 Web 应用程序中主要有两种角色：管理员是特权角色，可管理每个别员工的个人资料；员工是普通角色，只能查看或更新自己的个人资料信息。所有员工的信息描述见表 1。

表 1: 数据库

名称	员工 ID	密码	工资	出生日期	社会保障号 (SSN)	昵称	邮箱	地址	电话
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

3 实验任务

3.1 任务 1：熟悉 SQL 语句

本任务的目标是熟悉 SQL 命令。我们的 Web 应用程序使用的数据存储在名为 MySQL 的数据库中，该数据库托管在我们的 MySQL 容器上。我们创建了一个名为 sqllab_users 的数据库，其中包含一个名为 credential 的表。此表格存储了每个员工个人资料信息（例如 eid、密码、工资等）。在这个任务中，你需要与数据库互动以熟悉 SQL 命令。

请在 MySQL 容器中获取一个 shell（参见容器手册中的说明，手册的链接在实验网站）。然后使用 `mysql` 客户端程序来与数据库进行交互。用户名是 root，密码是 dees。

```
// 在MySQL容器内
# mysql -u root -pdees
```

登录后，可以创建新的数据库或加载已有的。因为我们已经创建了 sqllab_users 数据库，你只需要使用 `use` 命令来加载它。要查看 sqllab_users 数据库中的表，请使用 `show tables` 命令打印所选数据库中的所有表。

```
mysql> use sqllab_users;
Database changed
mysql> show tables;
+-----------------------+
| Tables_in_sqllab_users |
+-----------------------+
| credential             |
+-----------------------+
```

运行上述命令后，你需要使用 SQL 命令来打印员工 Alice 的所有个人资料信息。请提供你的结果截图。

3.2 任务 2：在 SELECT 语句上执行 SQL 注入攻击

SQL 注入是一种代码注入技术，通过该技术，攻击者可以执行自己的恶意 SQL 语句（通常称为恶意负载）。通过恶意 SQL 语句，攻击者可以从受害者数据库中窃取信息，甚至更改数据库。我们的员工管理 Web 应用程序存在 SQL 注入漏洞，这些漏洞模仿了开发人员经常犯的错误。

我们将使用 `www.seed-server.com` 中的登录页面来完成此任务。登录页面如图 1 所示。它要求用户提供用户名和密码。Web 应用程序根据这两条数据对用户进行身份验证，因此只有知道密码的员工才允许登录。作为攻击者，您的工作是在不知道任何员工密码的情况下登录 Web 应用程序。

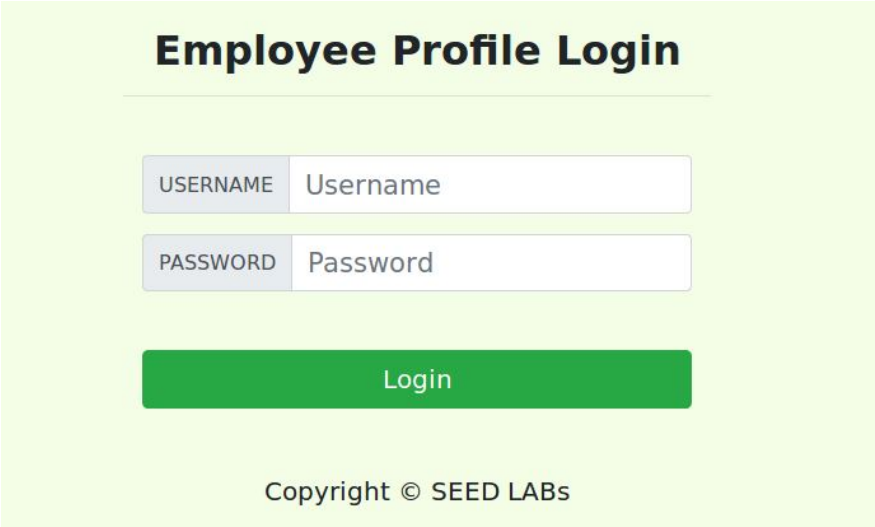

图 1: 登录页面

我们先解释一下这个 Web 应用程序是如何实现身份验证的。位于 `/var/www/SQL_Injection` 目录中的 PHP 代码 `unsafe_home.php` 是用于进行用户身份验证的。以下代码片段显示了如何对用户进行身份验证。

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
```

```
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
                nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);

// 以下是伪代码
if(id != NULL) {
  if(name=='admin') {
     return All employees information;
  } else if (name !=NULL){
    return employee information;
  }
} else {
  Authentication Fails;
}
```

上述 SQL 语句从 `credential` 表中选择员工个人信息，例如 id、姓名、薪水、ssn 等。该 SQL 语句使用了两个变量 `input_uname` 和 `hashed_pwd`，其中 `input_uname` 保存着用户在登录页面的用户名字段中输入的信息，而 `hashed_pwd` 保存着用户输入的密码的 `sha1` 哈希值。程序检查数据库中是否有记录与提供的用户名和密码匹配，如果是的话，那么用户身份验证就成功了，相应的员工信息会提供给用户。如果没有记录匹配，则身份验证失败。

任务 2.1：网页上的 SQL 注入攻击。 您的任务是从登录页面以管理员身份登录 Web 应用程序，以便查看所有员工的信息。我们假设您知道管理员的帐户名称，即 `admin`，但不知道密码。您需要决定在*用户名*和*密码*字段中输入什么才能成功登录。

任务 2.2：从命令行进行 SQL 注入攻击。 您的任务是重复任务 2.1，但您需要在不使用网页的情况下执行此操作。您可以使用命令行工具，例如 `curl`，它可以发送 HTTP 请求。值得一提的是，如果您想在 HTTP 请求中包含多个参数，则需要将 URL 和参数放在一对单引号之间，否则，用于分隔参数的特殊字符（例如 `&`）将被 shell 程序解释，从而改变命令的含义。以下示例显示如何向我们的 Web 应用程序发送 HTTP GET 请求，并附加两个参数（`username` 和 `Password`）：

```
$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
```

如果你需要在用户名或密码字段中包含特殊字符，你需要对它们进行编码，否则，这些特殊字符可以改变你的请求的意义。如果你想在这些字段中包含单引号，你应该使用%27 代替；如果要包含空格，则应使用%20。在这个任务中，在发送请求时你需要处理好 HTTP 编码。

任务 2.3：附加一个新的 SQL 语句 在这两个攻击中，我们只能从数据库中窃取信息，如果有办法可以利用这个登录页面的漏洞修改数据库将会更好。一个想法是使用 SQL 注入攻击注入两条 SQL 语句，第二条可以是更新或删除语句。在 SQL 中，分号（;）用于分隔两条 SQL 语句，请尝试通过登录页面

运行两条 SQL 语句。

系统里有一个防范措施会防止你运行两条 SQL 语句，所以你的攻击不会成功。请查阅网络资源找出这种防范措施是什么，并将你的发现记录在实验报告中。

3.3 任务 3：在 UPDATE 语句上执行 SQL 注入攻击

如果一个 SQL 注入漏洞出现在 UPDATE 语句上，损害将会更大，因为攻击者可以利用这个漏洞修改数据库。我们的员工管理应用程序有一个允许员工更新其个人资料信息（包括昵称、电子邮件、地址、电话号码和密码）的页面（参见附图 2）。要访问此页面，员工需要先登录。

当员工通过这个编辑资料页面来更新他们的信息时，以下 SQL UPDATE 查询将被执行，这个代码在 `unsafe_edit_backend.php` 文件中，位于 /var/www/SQLInjection 目录下。

```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
   nickname='$input_nickname',
   email='$input_email',
   address='$input_address',
   Password='$hashed_pwd',
   PhoneNumber='$input_phonenumber'
   WHERE ID=$id;";
$conn->query($sql);
```

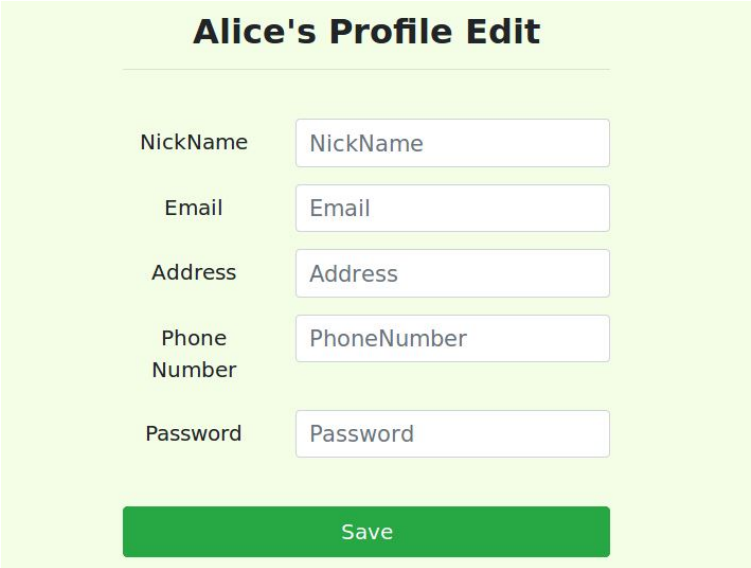

图 2: 编辑资料页面

任务 3.1：修改自己的工资 如图所示，员工只能更新他们的昵称、电子邮件、地址、电话号码和密码，他们没有资格更改自己的工资。假设你是 Alice，你不满你的老板 Boby 今年没有给你加工资。你想利

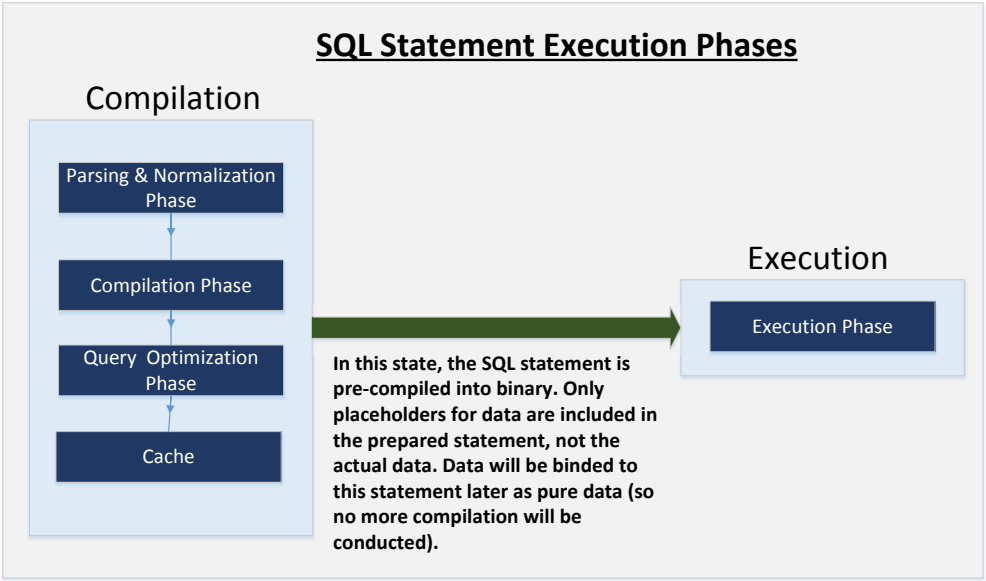

图 3: Prepared 语句的工作流程

用编辑资料的 SQL 注入漏洞来增加自己的工资，请演示如何实现这一点。我们假定你知道工资在数据库表格中的列名是 salary。

任务 3.2：修改其他人的工资 在增加了自己的工资之后，你决定惩罚你的老板 Boby。你想把他的工资减少到 1 美元。请演示如何实现这一点。

任务 3.3：修改其他人的密码 在更改完 Boby 的工资后，你依然不满，所以想把 Boby 的密码改成你知道的内容，然后你就可以登录到他的账号。请演示如何实现这一点。需要注意的是，数据库中存储的是密码的哈希值而不是明文密码字符串。你可以再次查看 `unsafe_edit_backend.php` 代码以了解密码是如何被存储的，它使用 SHA1 哈希函数生成密码的哈希值。

3.4 任务 4：防范措施 — Prepared 语句

SQL 注入漏洞的根本问题是未能将代码与数据区分开来。当在 Web 应用服务器端构建 SQL 语句时，程序员是知道哪一部分是数据哪一部分是代码，但在将 SQL 语句发送到数据库后，原来的界限已经消失，SQL 解释器看到的数据和代码的界限可能与开发人员设置的原始界限不同。为了解决这个问题，确保服务器端和数据库端看到的界限是一致至关重要。最安全的方法是使用 *Prepared 语句*。

为了了解 Prepared 语句如何防止 SQL 注入，我们需要理解当 SQL 服务器收到 SQL 语句时会发生什么。图 3 展示了 SQL 语句执行的工作流程。在编译步骤中，语句首先通过解析和规范化阶段，检查其语法和语义。接下来是编译阶段，在这个阶段，关键字（如 SELECT、FROM、UPDATE 等）被转换成机器可理解的格式。基本上，在此阶段，语句就被解释了。在优化阶段会考虑执行语句的不同方案，并从中选择最佳的优化方案。所选的计划被缓存存储起来，当下一个语句进来时，它将与缓存中的内容进行比较，如果已经在缓存中，则可以跳过解析、编译和查询优化阶段，直接将已编译的语句交给

执行阶段运行。

Prepared 语句不是一个完整的 SQL 语句，它的一些数据部分是空着的，需要在后面来填充。Prepared 语句会经过编译步骤转换成预编译的语句。要运行这个预编译语句，需要填充缺的数据，但是这些数据不会再经历编译步骤，它们将被直接插入到预编译语句中，交给执行引擎。因此，即使数据中包含 SQL 代码，这些代码也不会经过编译，它们会被当做数据，没有任何特殊意义。这就是 Prepared 语句为什么能防止 SQL 注入攻击的原理。

下面是一个中使用 Prepared 语句的例子。我们使用 SELECT 语句作为示例，并展示如何使用 Prepared 语句来重写 SQL 代码。

```
$sql = "SELECT name, local, gender
        FROM USER_TABLE
        WHERE id = $id AND password ='$pwd' ";
$result = $conn->query($sql)
```

上述代码存在 SQL 注入漏洞。可以将其重写为以下形式

```
$stmt = $conn->prepare("SELECT name, local, gender
                        FROM USER_TABLE
                        WHERE id = ? and password = ? ");
// 绑定参数
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

使用 Prepared 语句，我们把发送 SQL 语句到数据库的过程分为两步。第一步是仅发送代码部分，即不包含实际数据。这是编译步骤。如上例所示，实际的数据被占位符（?）替换。在该步骤之后，我们再通过 `bind_param()` 发送数据给数据库。数据库把此步骤中发送的一切仅视为数据而不再将其视为代码。它将这些数据绑定到编译好的 Prepared 语句中的相应占位符上。在 `bind_param()` 方法中，第一个参数`"is"` 指示了参数的类型：`"i"` 表明 `$id` 中的数据为整数类型，而 `"s"` 则表示 `$pwd` 中的数据为字符串类型。

任务。 在这个任务中，我们将使用 Prepared 语句来修复 SQL 注入漏洞。为了简化起见，在 defense 文件夹中我们创建了一个简化的程序。你需要对这个文件夹中的文件进行修改。如果你将浏览器指向以下 URL，则会看到类似于 Web 应用程序登录页面的页面。该页面允许你查询员工的信息，但需要提供正确的用户名和密码。

```
URL: http://www.seed-server.com/defense/
```

此页中输入的数据将被发送到名为 getinfo.php 的服务器程序，该程序会调用名为 unsafe.php 的程序。这个 PHP 程序中的 SQL 查询存在 SQL 注入漏洞。你的任务是用 Prepared 语句修改 unsafe.php 中的 SQL 语句，从而使程序能够防范 SQL 注入攻击。在实验设置文件夹中，unsafe.php 程序位于 image_www/Code/defense 文件夹内。你可以直接在那里修改该程序。完成后，你需要重建并重启容器，否则更改将不会生效。你也可以通过 `"docker cp"` 命令将文件复制到正在运行的容器中。

你还可以在正在运行的容器内部修改此文件。在这个容器内，unsafe.php 程序位于 `/var/www/SQL_Injection/defense` 文件夹下。为了保持 Docker 镜像较小，我们在容器里仅安装了一个非常简单的文本编辑器 nano。对于简单的编辑来说应该足够了。如果你不喜欢这个编辑器，你也可以通过执行以下命令来安装其它命令行编辑器，例如 vim：

```
# apt install -y vim
```

此安装将在容器关闭并销毁后被丢弃。如果你想让其永久化，则可以将安装命令添加到 image_www 文件夹内的 Dockerfile 中。

4 指南

测试 SQL 注入字符串。 在实际应用中，很难检查你的 SQL 注入攻击是否包含任何语法错误，因为通常服务器不会返回此类错误消息。为了进行调查，你可以将 PHP 源代码中的 SQL 语句复制到 MySQL 控制台。假设你有以下 SQL 语句，并且注入字符串是`' or 1=1;#`。

```
SELECT * from credential
WHERE name='$name' and password='$pwd';
```

你可以将 `$name` 的值替换为注入字符串，然后在 MySQL 控制台中测试它。这种方法可以帮助你在实际攻击之前构建一个没有语法错误的注入字符串。

5 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。