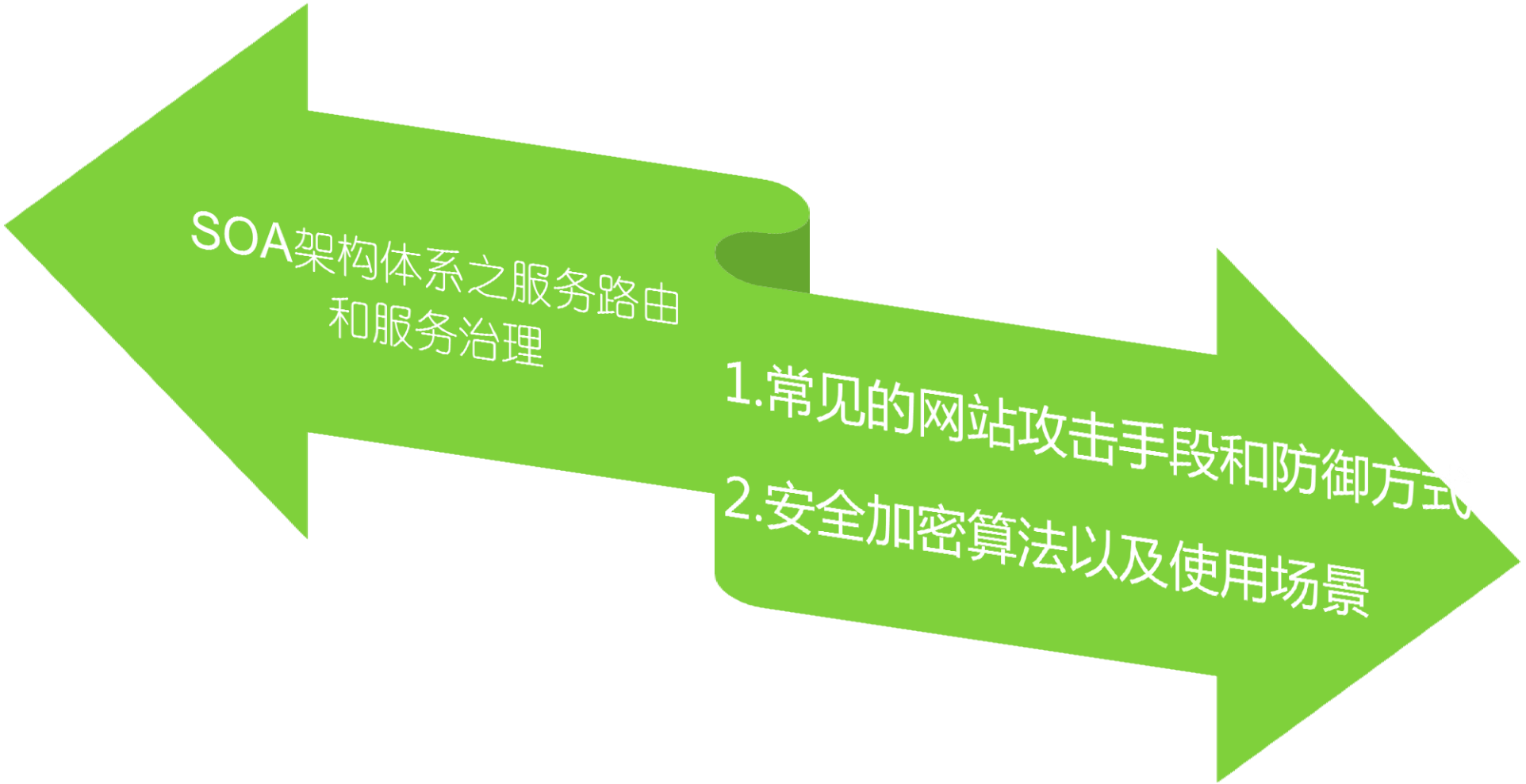
大型电商分布式系统实践 第3周

DATAGURU专业数据分析社区

上节内容回顾



常见的攻击手段--XSS

XSS攻击的全称是跨站脚本攻击(Cross Site Scripting)，为不跟层叠样式表(Cascading Style Sheets,CSS)的缩写混淆，故将跨站脚本攻击缩写为XSS,是WEB应用程序中最常见到的攻击手段之一。跨站脚本攻击指的是攻击者在网页中嵌入恶意脚本程序，当用户打开该网页时，脚本程序便开始在客户端的浏览器上执行，以盗取客户端cookie、盗取用户名密码、下载执行病毒木马程序甚至是获取客户端admin权限等等。

<input type="text" name="nick" value="xiaomao">

<input type="text" name="nick" value=""/><script>alert("haha")</script><!-" />

常见的攻击手段--XSS

还有一种场景，用户在表单上输入一段数据后，提交给服务端进行持久化，其他页面上需要从服务端将数据取出来展示。还是使用之前那个表单nick，用户输入昵称之后，服务端会将nick保存，并在新的页面展现给用户，当普通用户正常输入zhangsan，页面会显示用户的nick为zhangsan：

```
<body>
        zhangsan
</body>
```

但是，如果用户输入的不是一段正常的nick字符串，而是<script>alert("haha")</script>，服务端会将这段脚本保存起来，当有用户查看该页面时，页面会出现如下代码：

```
<body>
        <script>alert("haha")</script>
</body>
```

常见的攻击手段—XSS该如何防御

XSS之所以会发生，是因为用户输入的数据变成了代码。因此，我们需要对用户输入的数据进行HTML转义处理，将其中的“尖括号”、“单引号”、“引号” 之类的特殊字符进行转义编码。

HTML字符	HTML转义后的字符
<	&lt;
>	&gt;
'	&amp;

如今很多开源的开发框架本身默认就提供HTML代码转义的功能，如流行的jstl、Struts等等，不需要开发人员再进行过多的开发。使用jstl标签进行HTML转义，将变量输出，代码如下：

```
<c:out value="${nick}" escapeXml="true"></c:out>
```

只需要将escapeXml设置为true，jstl就会将变量中的HTML代码进行转义输出。

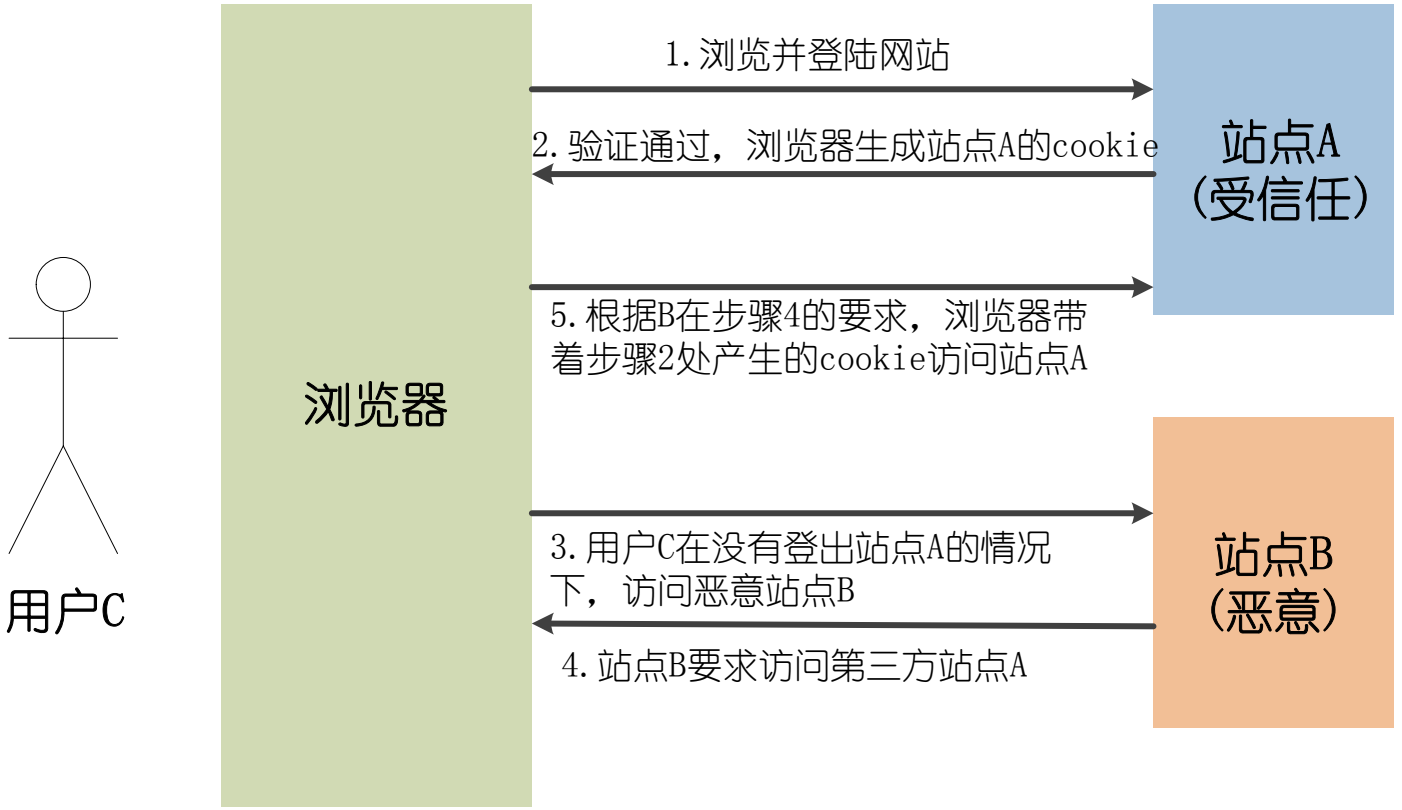
常见的攻击手段--CSRF

CSRF攻击的全称是跨站请求伪造(cross site request forgery)，是一种对网站的恶意利用，尽管听起来跟XSS跨站脚本攻击有点相似，但事实上CSRF与XSS差别很大，XSS利用的是站点内的信任用户，而CSRF则是通过伪装来自受信任用户的请求来利用受信任的网站。你可以这么理解CSRF攻击：攻击者盗用了你的身份，以你的名义向第三方网站发送恶意请求。CRSF能做的事情包括利用你的身份发邮件、发短信、进行交易转账等等，甚至盗取你的账号。

常见的攻击手段--CRSF

站点A：存在CSRF漏洞的站点
站点B：恶意攻击者
用户C：受害者

6.由于浏览器会带上用户C的cookie，站点A不知道步骤5的请求是B发出的，因此站点A会根据用户C的权限处理步骤5的请求，这样B就达到了伪造用户C请求的目的



常见的攻击手段--CRSF

假设某银行网站A，他以GET请求来发起转账操作，转账的地址为www.xxx.com/transfer.do?accountNum=10001&money=10000，accountNum参数表示转账的目的账户，money参数表示转账金额。

而某大型论坛B上，一个恶意用户上传了一张图片，而图片的地址栏中填的并不是图片的地址，而是前面所说的转账地址：

<img src="http://www.xxx.com/transfer.do?accountNum=10001&money=10000">

当你登陆网站A后，没有及时登出，这个时候你访问了论坛B，不幸的事情发生了，你会发现你的账户里面少了10000块……

为什么会这样呢，在你登陆银行A的时候，你的浏览器端会生成银行A的cookie，而当你访问论坛B的时候，页面上的<img>标签需要浏览器发起一个新的HTTP请求，以获得图片资源，当浏览器发起请求的时候，请求的却是银行A的转账地址www.xxx.com/transfer.do?accountNum=10001&money=10000，并且会带上银行A的cookie信息，结果银行的服务器收到这个请求后，会认为是你发起的一次转账操作，因此你的账户里边便少了10000块。

常见的攻击手段—CSRF的防御

1.cookie设置为HttpOnly
CSRF攻击很大程度上是利用了浏览器的cookie，为了防止站内的XSS漏洞盗取cookie，需要在cookie中设置"HttpOnly"属性，这样通过程序(如JavascriptS脚本、Applet等)就无法读取到cookie信息，避免了攻击者伪造cookie的情况出现。

2.增加token
CSRF攻击之所以能够成功，是因为攻击者可以伪造用户的请求，该请求中所有的用户验证信息都存在于cookie中，因此攻击者可以在不知道用户验证信息的情况下直接利用用户的cookie来通过安全验证。由此可知，抵御CSRF攻击的关键在于：在请求中放入攻击者所不能伪造的信息，并且该信息不存在于cookie之中。鉴于此，系统开发人员可以在HTTP请求中以参数的形式加入一个随机产生的token，并在服务端进行token校验，如果请求中没有token或者token内容不正确，则认为是CSRF攻击而拒绝该请求。

常见的攻击手段—CSRF的防御

3.通过Referer识别
根据HTTP协议，在HTTP头中有一个字段叫Referer，它记录了该HTTP请求的来源地址。在通常情况下，访问一个安全受限页面的请求都来自于同一个网站。比如某银行的转账是通过用户访问http://www.xxx.com/transfer.do页面完成，用户必须先登录www.xxx.com，然后通过点击页面上的提交按钮来触发转账事件。当用户提交请求时，该转账请求的Referer值就会是提交按钮所在页面的URL（本例为www.xxx.com/transfer.do）。如果攻击者要对银行网站实施CSRF攻击，他只能在其他的网站构造请求，当用户通过其他网站发送请求到银行时，该请求的Referer的值是其他网站的地址，而不是银行转账页面的地址。
因此，要防御CSRF攻击，银行网站只需要对于每一个转账请求验证其Referer值，如果是以www.xxx.com域名开头的地址，则说明该请求是来自银行网站自己的请求，是合法的。如果Referer是其他网站的话，就有可能是CSRF攻击，则拒绝该请求。

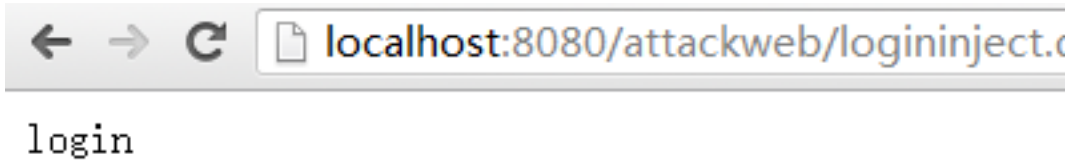
常见的攻击手段—SQL注入攻击

所谓SQL注入，就是通过把SQL命令伪装成正常的HTTP请求参数，传递到服务端，欺骗服务器最终执行恶意的SQL命令，达到入侵目的。攻击者可以利用SQL注入漏洞，查询非授权信息，修改数据库服务器的数据，改变表结构，甚至是获取服务器root权限。总而言之，SQL注入漏洞的危害极大，攻击者采用的SQL指令，决定攻击的威力。当前涉及到大批量数据泄露的攻击事件，大部分都是通过利用SQL注入来实施的。

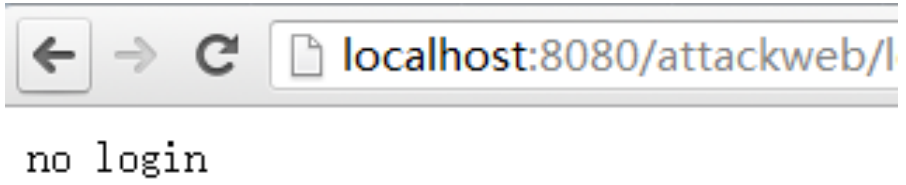
常见的攻击手段—SQL注入攻击原理

假设有个网站的登录页面，如下所示：

假设用户输入nick为zhangsan，密码为password1，则验证通过，显示用户登录：



否则，显示用户没有登录：

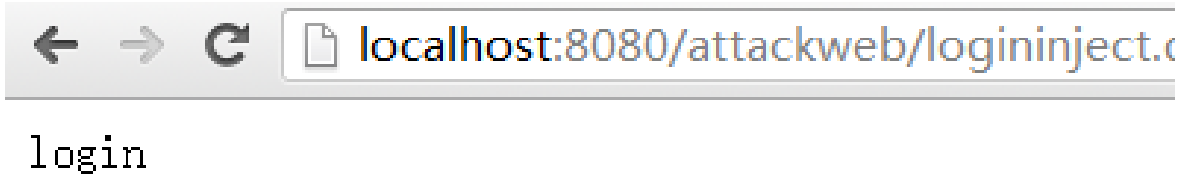


常见的攻击手段—SQL注入攻击原理

```
Connection  conn = getConnection();
String sql = "select * from hhuser where nick = '" + nickname +
                        "'" + " and passwords = '" + password + "'";
Statement st = (Statement) conn.createStatement();
ResultSet rs = st.executeQuery(sql);
List<UserInfo> userInfoList = new ArrayList<UserInfo>();
while (rs.next()) {
        UserInfo userinfo = new UserInfo();
        userinfo.setUserid(rs.getLong("userid"));
        userinfo.setPasswords(rs.getString("passwords"));
        userinfo.setNick(rs.getString("nick"));
        userinfo.setAge(rs.getInt("age"));
        userinfo.setAddress(rs.getString("address"));
        userInfoList.add(userinfo);
}
```

常见的攻击手段—SQL注入攻击原理

当用户输入nick为zhangsan，密码为’ or ’1’=’1的时候，意想不到的事情出现了，页面显示为login状态：



以上便是一次简单的、典型的SQL注入攻击。当然，SQL注入的危害不仅如此，假设用户输入用户名zhangsan，在密码框输入’ ;drop table aaa;--， 会发生什么呢？

常见的攻击手段—SQL注入攻击防御

1. 使用预编译语句

预编译语句PreparedStatement是java.sql中的一个接口，继承自Statement接口。通过Statement对象执行SQL语句时，需要将SQL语句发送给DBMS，由DBMS先进行编译后再执行。而预编译语句和Statement不同，在创建PreparedStatement对象时就指定了SQL语句，该语句立即发送给DBMS进行编译，当该编译语句需要被执行时，DBMS直接运行编译后的SQL语句，而不需要像其他SQL语句那样首先将其编译。

前面介绍过，引发SQL注入的根本原因是恶意用户将SQL指令伪装成参数传递到后端数据库执行，作为一种更为安全的动态字符串的构建方法，预编译语句使用参数占位符来替代需要动态传入的参数，这样攻击者无法改变SQL语句的结构，SQL语句的语义不会发生改变，即便用户传入类似于前面’ or ’1’=’1这样的字符串，数据库也会将其作为普通的字符串来处理。

常见的攻击手段—SQL注入攻击防御

2. 使用ORM框架

由上文可见，防止SQL注入的关键在于对一些关键字符进行转义，而常见的一些ORM框架，如ibatis、hibernate等，都支持对相应的关键字或者特殊符号进行转义，可以通过简单的配置，很好的预防SQL注入漏洞，降低了普通的开发人员进行安全编程的门槛。

Ibatis的insert语句配置：

```
<insert id="insert" parameterClass="userDO">
        insert into
        users(gmt_create,gmt_modified,userid,user_nick,address,age,sex)
        values(now(),now(),#userId#,#userNick#,#address#,#age#,#sex#)
</insert>
```

通过#符号配置的变量，ibatis能够对输入变量的一些关键字进行转义，防止SQL注入攻击。

常见的攻击手段—SQL注入攻击防御

3. 避免密码明文存放
对存储的密码进行单向Hash，如使用MD5对密码进行摘要，而非直接存储明文密码，这样的好处就是万一用户信息泄露，即圈内所说的被“拖库”，黑客无法直接获取用户密码，而只能得到一串跟密码相差十万八千里的Hash码。

4. 处理好相应的异常
后台的系统异常，很可能包含了一些如服务器版本、数据库版本、编程语言等等的信息，甚至是数据库连接的地址及用户名密码，攻击者可以按图索骥，找到对应版本的服务器漏洞或者数据库漏洞进行攻击，因此，必须要处理好后台的系统异常，重定向到相应的错误处理页面，而不是任由其直接输出到页面上。

常见的攻击手段—文件上传漏洞

在上网的过程中，我们经常会将一些如图片、压缩包之类的文件上传到远端服务器进行保存，文件上传攻击指的是恶意攻击者利用一些站点没有对文件的类型做很好的校验这样的漏洞，上传了可执行的文件或者脚本，并且通过脚本获得服务器上相应的权利，或者是通过诱导外部用户访问或者下载上传的病毒或者木马文件，达到攻击目的。

为了防范用户上传恶意的可执行文件和脚本，以及将文件上传服务器当做免费的文件存储服务器使用，需要对上传的文件类型进行白名单(非黑名单，这点非常重要)校验，并且限制上传文件的大小，上传的文件，需要进行重新命名，使攻击者无法猜测到上传文件的访问路径。

对于上传的文件来说，不能简单的通过后缀名称来判断文件的类型，因为恶意攻击可以将可执行文件的后缀名称改成图片或者其他的后缀类型，诱导用户执行。因此，判断文件类型需要使用更安全的方式。

很多类型的文件，起始的几个字节内容是固定的，因此，根据这几个字节的内容，就可以确定文件类型，这几个字节也被称为魔数(magic number)。

常见的攻击手段—DDoS攻击

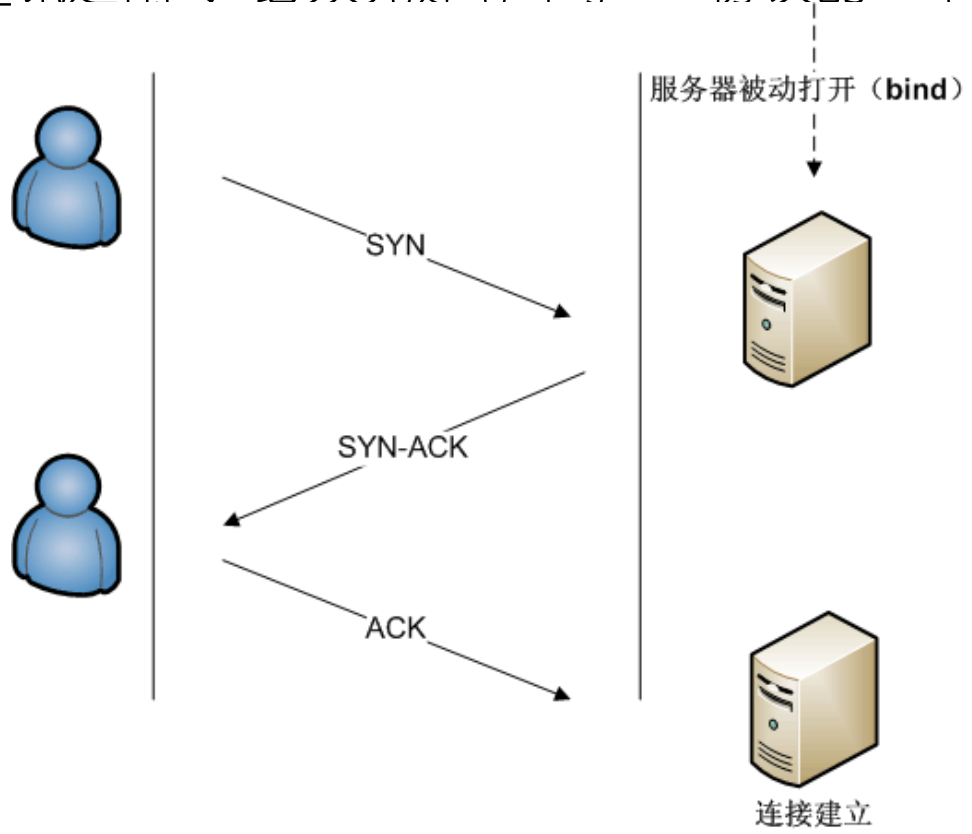
DDoS(Distributed Denial of Service)，即分布式拒绝服务攻击，是目前最为强大、最难以防御的攻击方式之一。要理解DDos，得先从DoS说起。最基本的DoS攻击就是利用合理的客户端请求来占用过多的服务器资源，从而使合法用户无法得到服务器的响应。DDoS攻击手段是在传统的DoS攻击基础之上产生的一类攻击方式，传统的DoS攻击一般是采用一对一方式的，当攻击目标CPU速度、内存或者网络带宽等等各项性能指标不高的情况下，它的效果是明显的，但随着计算机与网络技术的发展，计算机的处理能力显著增加，内存不断增大，同时也出现了千兆级别的网络，这使得DoS攻击逐渐失去效果。这时候分布式拒绝服务攻击手段（DDoS）便应运而生了。你理解了DoS攻击以后，DDoS的原理就非常简单了，它指的是攻击者借助公共网络，将数量庞大的计算机设备联合起来作为攻击平台，对一个或多个目标发动攻击，从而达到瘫痪目标主机的目的。通常，在攻击开始前，攻击者会提前控制大量的用户计算机，称之为“肉鸡”，并通过指令使大量的肉鸡在同一时刻对某个主机进行访问，从而达到瘫痪目标主机的目的。

常见的攻击手段—DDoS攻击

DDoS的攻击有很多种类型，如依赖蛮力的ICMP Flood、UDP Flood等等，随着硬件性能的提升，需要的机器规模越来越大，组织大规模的攻击越来越困难，现在已经不常见，还有就是依赖协议特征以及具体的软件漏洞进行的攻击，如Slowloris攻击，Hash碰撞攻击等等，这类攻击主要利用协议以及软件漏洞发起攻击，需要在特定环境下才会出现，更多的攻击者采用的是前面两种的混合方式，即利用了协议、系统的缺陷，又具备了海量的流量，如SYN Flood、DNS Query Flood等等。

常见的攻击手段—SYN Flood

SYN Flood是互联网最经典的攻击方式之一，要明白它的攻击原理，还得从TCP协议连接建立的过程开始说起，TCP协议与UDP协议不同，TCP是基于连接的协议，也就是说，在进行TCP协议通讯之前，必须先建立基于TCP协议的一个连接，连接建立的过程如下：



常见的攻击手段—DNS Query Flood

DNS Query Flood实际上是UDP Flood攻击的一种变形，由于DNS服务在互联网中不可替代的作用，一旦DNS服务器瘫痪，影响甚大。
DNS Query Flood攻击采用的方法是向被攻击的服务器发送海量的域名解析请求，通常，请求解析的域名是随机生成，大部分根本就不存在，并且通过伪造端口和客户端IP，防止查询请求被ACL过滤。被攻击的DNS 服务器在接收到域名解析请求后，首先会在服务器上查找是否有对应的缓存，由于域名是随机生成的，几乎不可能有相应的缓存信息，当没有缓存，并且该域名无法直接由该DNS服务器进行解析的时候，DNS服务器会向其上层DNS服务器递归查询域名信息，直到全球互联网的13台根DNS服务器。大量不存在的域名解析请求，给服务器带来了很大的负载，当解析请求超过一定量的时候，就会造成DNS服务器解析域名超时，这样攻击者便达成了攻击目的。

常见的攻击手段—CC攻击

CC(Challenge Collapsar)攻击属于DDos的一种，是基于应用层HTTP协议发起的DDos攻击，也被称为HTTP Flood。

CC攻击的原理是这样的，攻击者通过控制的大量“肉鸡”或者利用从互联网上搜寻的大量匿名的HTTP代理，模拟正常用户给网站发起请求直到该网站拒绝服务为止。大部分网站会通过CDN以及分布式缓存来加快服务端响应，提升网站的吞吐量，而这些精心构造的HTTP请求往往有意避开这些缓存，需要进行多次DB查询操作或者是一次请求返回大量的数据，加速系统资源消耗，从而拖垮后端的业务处理系统，甚至连相关存储以及日志收集系统也无法幸免。

常见安全算法—数字摘要

数字摘要也称为消息摘要，它是一个唯一对应一个消息或文本的固定长度的值，它由一个单向Hash函数对消息进行计算而产生。如果消息在传递的途中改变了，接收者通过对收到消息采用相同的Hash重新计算，新产生的摘要与原摘要进行比较，就可知道消息是否被篡改了，因此消息摘要能够验证消息的完整性。消息摘要采用单向Hash函数将需要计算的内容"摘要"成固定长度的串，这个串亦称为数字指纹。这个串有固定的长度，且不同的明文摘要成密文，其结果总是不同的(相对的)，而同样的明文其摘要必定一致。这样这串摘要便可成为验证明文是否是"真身"的"指纹"了。

常见安全算法—常用数字摘要算法

MD5

MD5即Message Digest Algorithm 5(信息摘要算法5)，是数字摘要算法一种实现，用于确保信息传输完整性和一致性，摘要长度为128位。MD5由MD4、MD3、MD2改进而来，主要增强算法复杂度和不可逆性，该算法因其普遍、稳定、快速的特点，在产业界得到了极为广泛的使用，目前主流的编程语言普遍都已有MD5算法实现。

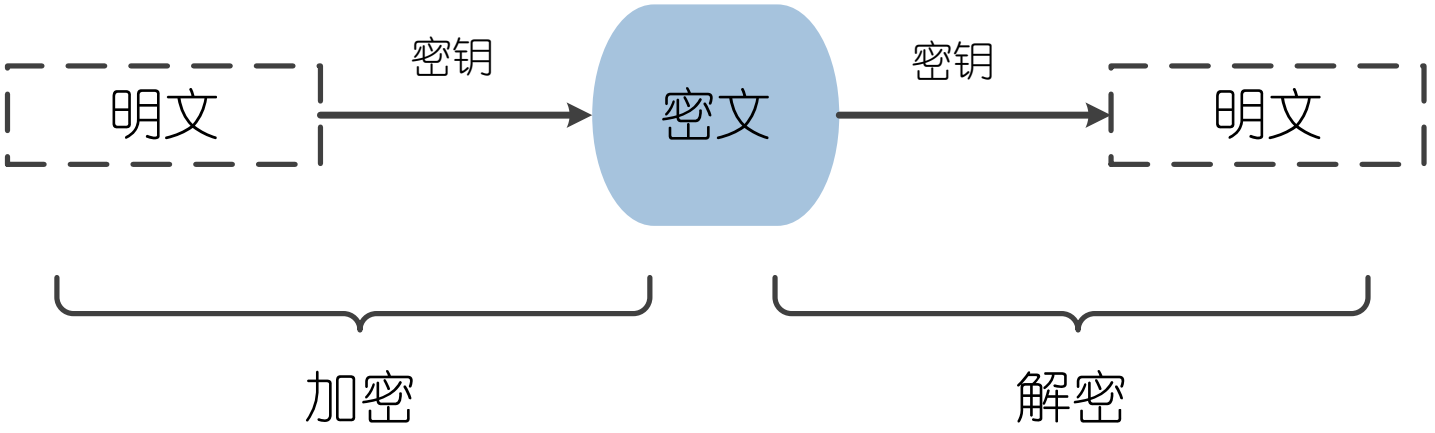
SHA

SHA的全称是Secure Hash Algorithm，即安全散列算法。1993年，安全散列算法(SHA)由美国国家标准和技术协会 (NIST)提出，并作为联邦信息处理标准(FIPS PUB 180)公布，1995年又发布了一个修订版FIPS PUB 180-1，通常称之为SHA-1。SHA-1是基于MD4算法的，现在已成为公认的最安全的散列算法之一，并被广泛使用。

SHA-1算法生成的摘要信息的长度为160位，由于生成的摘要信息更长，运算的过程更加复杂，在相同的硬件上，SHA-1的运行速度比MD5更慢，但是也更为安全。

常见安全算法—对称加密

对称加密算法是应用较早的加密算法，技术成熟。在对称加密算法中，数据发送方将明文(原始数据)和加密密钥一起经过特殊加密算法处理后，生成复杂的加密密文进行发送，数据接收方收到密文后，若想读取原文，则需要使用加密使用的密钥及相同算法的逆算法对加密的密文进行解密，才能使其恢复成可读明文。在对称加密算法中，使用的密钥只有一个，发送和接收双方都使用这个密钥对数据进行加密和解密，这就要求加密和解密方事先都必须知道加密的密钥。



常见安全算法—DES算法

DES算法

1973 年，美国国家标准局(NBS)在认识到建立数据保护标准既明显又急迫的情况下，开始征集联邦数据加密标准的方案。1975 年3月17日，NBS公布了IBM公司提供的密码算法，以标准建议的形式在全国范围内征求意见。经过两年多的公开讨论之后，1977 年7月15日，NBS宣布接受这建议，作为联邦信息处理标准46 号数据加密标准(Data Encryptin Standard)，即DES正式颁布，供商业界和非国防性政府部门使用。

DES算法属于对称加密算法，明文按64位进行分组，密钥长64位，但事实上只有56位参与DES运算(第8、16、24、32、40、48、56、64位是校验位，使得每个密钥都有奇数个1),分组后的明文和56位的密钥按位替代或交换的方法形成密文。

由于计算机运算能力的增强，原版DES密码的密钥长度变得容易被暴力破解，因此演变出了3DES算法。3DES是DES向AES过渡的加密算法，它使用3条56位的密钥对数据进行三次加密，是DES的一个更安全的变形。

常见安全算法—AES算法

AES算法

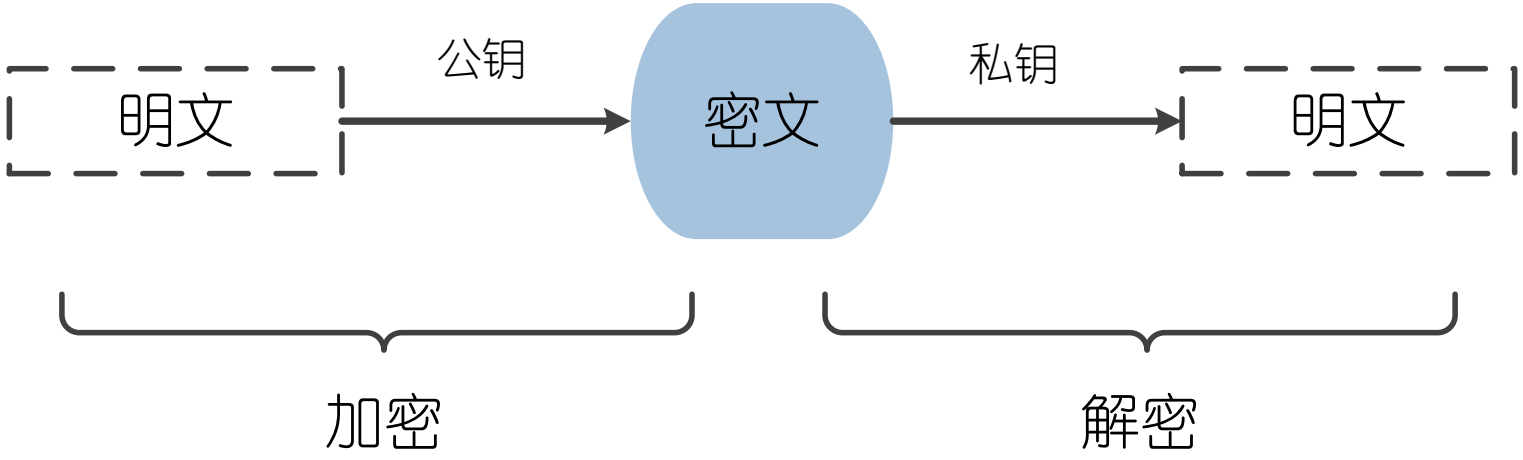
AES的全称是Advanced Encryption Standard，即高级加密标准，该算法由比利时密码学家Joan Daemen和Vincent Rijmen所设计，结合两位作者的名字，又称Rijndael加密算法，是美国联邦政府采用的一种对称加密标准，这个标准用来替代原先的DES算法，已经广为全世界所使用，已然成为对称加密算法中最流行的算法之一。

AES算法作为新一代的数据加密标准汇聚了强安全性、高性能、高效率、易用和灵活等优点，设计有三个密钥长度:128,192,256位，比DES算法的加密强度更高，更为安全。

常见安全算法—非对称加密

非对称加密算法又称为公开密钥加密算法，它需要两个密钥，一个称为公开密钥(public key)，即公钥，另一个称为私有密钥(private key)，即私钥。公钥与私钥需要配对使用，如果用公钥对数据进行加密，只有用对应的私钥才能进行解密，而如果使用私钥对数据进行加密，那么只有用对应的公钥才能进行解密。因为加密和解密使用的是两个不同的密钥，所以这种算法称为非对称加密算法。

非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公钥向其它人公开，得到该公钥的乙方使用该密钥对机密信息进行加密后再发送给甲方，甲方再使用自己保存的另一把专用密钥，即私钥，对加密后的信息进行解密。



常见安全算法—RSA算法

RSA算法

RSA非对称加密算法是1977年由Ron Rivest、Adi Shamirh和LenAdleman开发的，RSA取名来自开发他们三者的名字。RSA是目前最有影响力的非对称加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被ISO推荐为公钥数据加密标准。RSA算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但反过来想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

Thanks

FAQ时间