

漏洞 Vulnerability

@Zedd

漏洞是什么

漏洞是什么

漏洞是在[硬件](#)、[软件](#)、协议的具体实现或系统安全策略上存在的缺陷，从而可以使攻击者能够在未授权的[情况](#)下访问或破坏系统。

具体举例来说，比如在Intel Pentium芯片中存在的逻辑错误，在Sendmail早期版本中的[编程](#)错误，在NFS协议中认证方式上的弱点，在Unix[系统管理员](#)设置匿名Ftp服务时配置不当的问题都可能被攻击者使用，威胁到系统的安全。因而这些都可以认为是系统中存在的[安全漏洞](#)。

漏洞从哪来？是天生的吗？

漏洞从哪来？是天生的吗？

为什么设计者不去找需要安全工作者去找？

在计算机领域，漏洞特指系统的安全方面存在缺陷，一般被定义为信息系统设计、编码和运行当中引起的、可能被外部利用用于影响信息系统机密性、完整性、可用性的缺陷。

统计表明，程序员每写 1000 行代码，就会有 1 个缺陷，一个大型的应用系统，代码行数动辄几十万行，甚至更多。

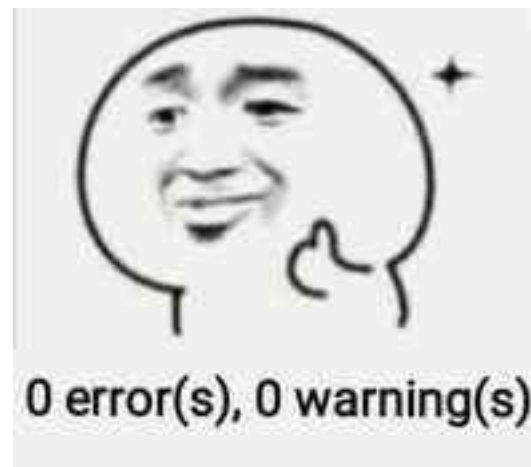
可以说，从世界上第一个操作系统或应用软件诞生的那天开始，缺陷就存在于 IT 系统的各个环节，而且始终会存在。

漏洞与BUG

漏洞与BUG

你家里的窗可以从外面打开，那叫漏洞。

你家里的窗打开时非常费劲，那叫bug。



漏洞与BUG

被利用的叫漏洞，没被利用的能叫漏洞吗？

是不是感到很哲学？



漏洞来源——操作缺陷

程序员编程序时的疏忽、运维人员设置安全配置时的不当操作、用户设置的简单口令和泄露.....

这些人为的、无意的失误就是操作缺陷。



漏洞来源——认知缺陷

2000年的“千足虫”危机，过去为了节省空间，存储年份用两位十进制数表示，例如1980就是80，1998年出生就是98-80=18，但是在2000就变成了负数，就会引起各种系统紊乱甚至崩溃。

2018年1月的 Meltdown & Spectre 。为了提升 CPU 处理性能，芯片企业用乱序执行和预测执行。通俗来说，CPU 并不完全严格按照指令的顺序来执行，而是会自己预测可能要执行的内容，以及为了更好地利用 CPU 资源将指令顺序打乱，以便能同时执行一些指令。

但设计者没有考虑到，或者没有人为这个问题时重要的，即：由于 CPU 缓存内容没有同步恢复到原始状态，导致缓存中存储的重要信息可以被漏洞利用者获取，可能会造成受保护的密码和敏感信息泄露。

漏洞来源——知识缺陷

很突出的一个例子就是工控安全。

原本的工业控制系统，大多以系统功能作为第一要素，多数系统在设计之初时封闭的“单机系统”，连联网需求都没有考虑过，就更不要提在设计、研发和集成阶段考虑网络安全问题了。

物联网时代到来以后，这些工控系统都开始在互联网上“裸奔”，黑客可以轻而易举地利用系统漏洞进行攻击，造成严重后果。

以及身边的类“工控”系统——学校内网。

漏洞与BUG

并不是所有的缺陷都是漏洞，只有可以被外部利用的缺陷才被称为漏洞。

这句话可以换一个角度来理解，当利用缺陷的方法出现时，漏洞导致的现实威胁就出现了。

就像“心脏滴血”漏洞，引发这个漏洞的缺陷在爆发前两年的版本中就已经静悄悄地存在，当黑客利用这个缺陷获取服务器里用户的敏感信息，影响了数据的机密性，就构成了漏洞。(OpenSSL)

漏洞与BUG

心脏出血（英语：Heartbleed），也简称为心血漏洞，是一个出现在加密程序库OpenSSL的安全漏洞

该程序库广泛用于实现互联网的传输层安全（TLS）协议

它于2012年被引入了软件中，2014年4月首次向公众披露。

只要使用的是存在缺陷的OpenSSL实例，无论是服务器还是客户端，都可能因此而受到攻击。

此问题的原因是在实现TLS的心跳扩展时没有对输入进行适当验证（缺少边界检查），因此漏洞的名称来源于“心跳”（heartbeat）。该程序错误属于缓冲区过读，即可以读取的数据比应该允许读取的还多。



Asuri

```
msf auxiliary(scanner/ssl/openssl_heartbleed) > exploit

[*] 192.168.224.131:8443 - Sending Client Hello...
[*] 192.168.224.131:8443 - SSL record #1:
[*] 192.168.224.131:8443 -   Type: 22
[*] 192.168.224.131:8443 -   Version: 0x0301
[*] 192.168.224.131:8443 -   Length: 86
[*] 192.168.224.131:8443 -   Handshake #1:
[*] 192.168.224.131:8443 -     Length: 82
[*] 192.168.224.131:8443 -     Type: Server Hello (2)
[*] 192.168.224.131:8443 -     Server Hello Version: 0x0301
[*] 192.168.224.131:8443 -     Server Hello random data: 5a43715421d06bd473b76a1669efc691114ec00eed78f64
[*] 192.168.224.131:8443 -     Server Hello Session ID length: 32
[*] 192.168.224.131:8443 -     Server Hello Session ID: e5e7d102eaefef68b82560b81ac90edb4478aac4a0d485e
[*] 192.168.224.131:8443 - SSL record #2:
[*] 192.168.224.131:8443 -   Type: 22
[*] 192.168.224.131:8443 -   Version: 0x0301
[*] 192.168.224.131:8443 -   Length: 675
[*] 192.168.224.131:8443 -   Handshake #1:
[*] 192.168.224.131:8443 -     Length: 671
[*] 192.168.224.131:8443 -     Type: Certificate Data (11)
[*] 192.168.224.131:8443 -     Certificates length: 668
[*] 192.168.224.131:8443 -     Data length: 671
[*] 192.168.224.131:8443 -     Certificate #1:
[*] 192.168.224.131:8443 -       Certificate #1: Length: 665
[*] 192.168.224.131:8443 -       Certificate #1: #<OpenSSL::X509::Certificate: subject=#<OpenSSL::X509::
SSL:::BN:0x005651169ac258>, not_before=2013-04-14 18:11:32 UTC, not_after=2018-04-13 18:11:32 UTC>
[*] 192.168.224.131:8443 - SSL record #3:
[*] 192.168.224.131:8443 -   Type: 22
[*] 192.168.224.131:8443 -   Version: 0x0301
[*] 192.168.224.131:8443 -   Length: 203
[*] 192.168.224.131:8443 -   Handshake #1:
[*] 192.168.224.131:8443 -     Length: 199
[*] 192.168.224.131:8443 -     Type: Server Key Exchange (12)
[*] 192.168.224.131:8443 - SSL record #4:
[*] 192.168.224.131:8443 -   Type: 22
[*] 192.168.224.131:8443 -   Version: 0x0301
[*] 192.168.224.131:8443 -   Length: 4
[*] 192.168.224.131:8443 -   Handshake #1:
[*] 192.168.224.131:8443 -     Length: 0
[*] 192.168.224.131:8443 -     Type: Server Hello Done (14)
[*] 192.168.224.131:8443 - Sending Heartbeat...
[*] 192.168.224.131:8443 - Heartbeat response, 13027 bytes
[+] 192.168.224.131:8443 - Heartbeat response with leak
[*] 192.168.224.131:8443 - Printable info leaked:
.....ZBr.....3{.....8.....NSs.....f.....!.9.8.....5.....3.2.....E.D...../...A.....
Kit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36..Accept: text/html,application/xhtml+xml,application/
/login.php..Accept-Encoding: gzip, deflate, br..Accept-Language: zh-CN,zh;q=0.9..Cookie: security_level=0; PHPSESSID=87
_level=0; PHPSESSID=29ba07aa8b6b3cd27b0163305fca92dd...Login=bee&password=bug&security_level=0&form=submit).....Y..b.
.....repeated 12027 times .....
..
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

漏洞与BUG

脏牛（Dirty Cow）是Linux内核的一个提权漏洞，攻击者可以利用这个漏洞获取root权限。

之所以叫Dirty Cow，因为这个漏洞利用了Linux的copy-on-write机制。脏牛的CVE编号是CVE-2016-5195。

脏牛漏洞是公开后影响范围最广和最深的漏洞之一，这十年来的每一个Linux版本，包括Android、桌面版和服务器版都受到其影响。

尽管已针对该漏洞进行了补丁修复，但国外安全公司Bindecy对该补丁和内容做出深入研究后发现，脏牛漏洞的修复补丁仍存在缺陷，由此产生了“大脏牛”漏洞。

“脏牛”漏洞是Linux内核之父Linus亲自修复的，他提交的补丁单独针对“脏牛”而言并没有问题。从我们的分析过程中发现，内核的开发者希望将“脏牛”的修复方法引用到PMD的逻辑中，但是由于PMD的逻辑和PTE并不完全一致才最终导致了“大脏牛”漏洞。连内核的开发者都会犯错，更何况普通的开发者。

“大脏牛”漏洞再一次提示我们，即便是官方修复过的漏洞仍有可能由修复补丁引入新的严重漏洞，漏洞在修复后需要我们像对待原始漏洞一样继续跟踪其修复补丁。

漏洞杀伤力

不同的漏洞造成的杀伤力不同，也跟利用漏洞的方式有关系。

2016-2017年，NSA 数字武器库遭到泄露，相关人士为了证明自己成功攻入 NSA 开发网络武器的“方程式组织”的系统，在网络披露了几批工具。2017年肆虐全球的 WannaCry 病毒就是从这几批泄露武器中泄露出去的。

希拉里邮件门等。



漏洞的大致分类

对于目前我所见过的漏洞大致归纳为以下几类：

- 内存破坏类
- 逻辑错误类
- 输入验证类
- 设计错误类
- 配置错误类

Web 漏洞

Web Vulnerability

严重

1. 直接获取重要服务器（客户端）权限的漏洞。包括但不限于远程任意命令执行、上传 webshell、可利用远程缓冲区溢出、可利用的 ActiveX 堆栈溢出、可利用浏览器 use after free 漏洞、可利用远程内核代码执行漏洞以及其它因逻辑问题导致的可利用的远程代码执行漏洞；
2. 直接导致严重的信息泄漏漏洞。包括但不限于重要系统中能获取大量信息的SQL注入漏洞；
3. 能直接获取目标单位核心机密的漏洞；

高危

1. 直接获取普通系统权限的漏洞。包括但不限于远程命令执行、代码执行、上传webshell、缓冲区溢出等；
2. 严重的逻辑设计缺陷和流程缺陷。包括但不限于任意账号密码修改、重要业务配置修改、泄露；
3. 可直接批量盗取用户身份权限的漏洞。包括但不限于普通系统的SQL注入、用户订单遍历；
4. 严重的权限绕过类漏洞。包括但不限于绕过认证直接访问管理后台、cookie欺骗。
5. 运维相关的未授权访问漏洞。包括但不限于后台管理员弱口令、服务未授权访问。

中危

1. 需要在一定条件限制下，能获取服务器权限、网站权限与核心数据库数据的操作。包括但不限于交互性代码执行、一定条件下的注入、特定系统版本下的getshell等；
2. 任意文件操作漏洞。包括但不限于任意文件写、删除、下载，敏感文件读取等操作；
3. 水平权限绕过。包括但不限于绕过限制修改用户资料、执行用户操作。

低危

1. 能够获取一些数据，但不属于核心数据的操作；
2. 在条件严苛的环境下能够获取核心数据或者控制核心业务的操作；
3. 需要用户交互才可以触发的漏洞。包括但不限于XSS漏洞、CSRF漏洞、点击劫持；

忽略

- 虚假漏洞
- 互联网上已经被公开的漏洞
- 提交到本平台后又提交到其他平台的漏洞
- 没有链接、截图、利用方法等漏洞详情不详细的漏洞
- 需要登录管理员后台才能触发的漏洞
- 需要中间人攻击的漏洞
- Self-XSS
- 无敏感操作的CSRF漏洞
- 钓鱼漏洞
- 无敏感信息的 JSON Hijacking
- 扫描器取得结果，但白帽子无法提供利用方法的漏洞
- 无意义的源码泄露、内网IP、域名泄露
- 拒绝服务漏洞(DDos)

2017 OWASP Top 10

2013年版《OWASP Top 10》	→	2017年版《OWASP Top 10》
A1 – 注入	→	A1:2017 – 注入
A2 – 失效的身份认证和会话管理	→	A2:2017 – 失效的身份认证
A3 – 跨站脚本 (XSS)	↘	A3:2017 – 敏感信息泄漏
A4 – 不安全的直接对象引用 [与A7合并]	U	A4:2017 – XML外部实体 (XXE) [新]
A5 – 安全配置错误	↘	A5:2017 – 失效的访问控制 [合并]
A6 – 敏感信息泄漏	↗	A6:2017 – 安全配置错误
A7 – 功能级访问控制缺失 [与A4合并]	U	A7:2017 – 跨站脚本 (XSS)
A8 – 跨站请求伪造 (CSRF)	⊗	A8:2017 – 不安全的反序列化 [新, 来自于社区]
A9 – 使用含有已知漏洞的组件	→	A9:2017 – 使用含有已知漏洞的组件
A10 – 未验证的重定向和转发	⊗	A10:2017 – 不足的日志记录和监控 [新, 来自于社区]

Top 1 Injection Flaws 注入

将不受信任的数据作为命令或查询的一部分发送到解析器时，会产生诸如SQL注入、NoSQL注入、OS注入和LDAP注入的注入缺陷。攻击者的恶意数据可以诱使解析器在没有适当授权的情况下执行非预期命令或访问数据。

一些常见的注入，包括：SQL、OS命令、ORM、LDAP和表达式语言（EL）或OGNL注入。所有解释器的概念都是相同的。代码评审是最有效的检测应用程序的注入风险的办法之一，紧随其后的是对所有参数、字段、头、cookie、JSON和XML数据输入的彻底的 DAST 扫描，即 Dynamic Application Security Testing。

Top 2 Broken Authentication and Session Management 失效的身份认证

通常，通过错误使用应用程序的身份认证和会话管理功能

攻击者能够破译密码、密钥或会话令牌，或者利用其它开发缺陷来暂时性或永久性冒充其他用户的身份。

神奇密码：ji32k7au4a83

25 WORST PASSWORDS OF 2018 REVEALED

1. 123456	14. 666666
2. PASSWORD	15. ABC123
3. 123456789	16. FOOTBALL
4. 12345678	17. 123123
5. 12345	18. MONKEY
6. 111111	19. 654321
7. 1234567	20. !@#%~^&*
8. SUNSHINE	21. CHARLIE
9. QWERTY	22. AA123456
10. ILOVEYOU	23. DONALD
11. PRINCESS	24. PASSWORD1
12. ADMIN	25. QWERTY123
13. WELCOME	



Top 3 Sensitive Data Exposure 敏感数据泄露

许多Web应用程序和API都无法正确保护敏感数据，例如：财务数据、医疗数据和PII数据。

攻击者可以通过窃取或修改未加密的数据来实施信用卡诈骗、身份盗窃或其他犯罪行为。

未加密的敏感数据容易受到破坏，因此，我们需要对敏感数据加密，这些数据包括：传输过程中的数据、存储的数据以及浏览器的交互数据。

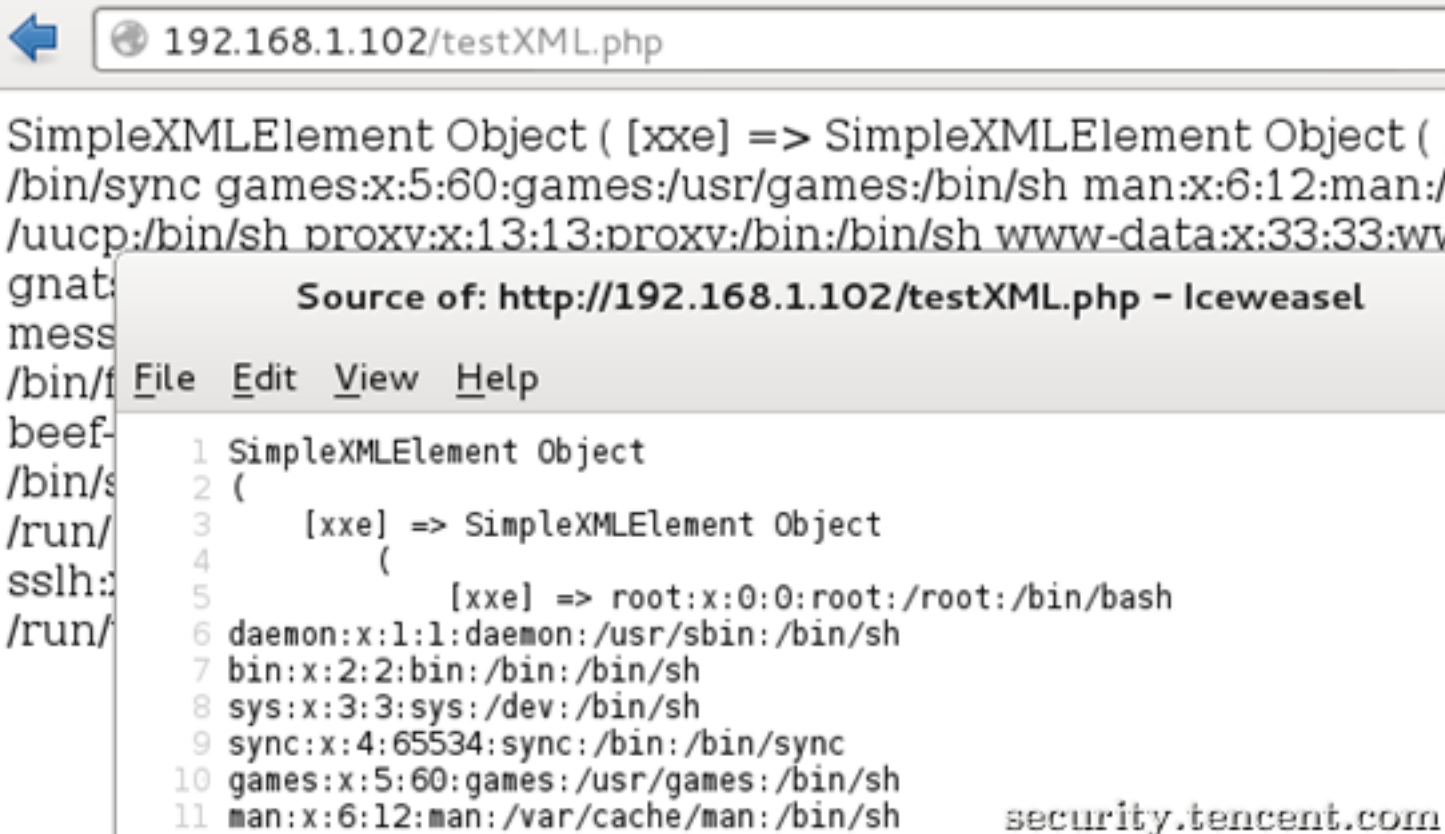
密码明文存储

Top 4 XML External Entity XML 外部实体

许多较早的Web配置器错误的XML处理器组件了XML文件中的外部实体引用

攻击者可
拒绝服务

Content



```

192.168.1.102/testXML.php

SimpleXMLElement Object ( [xxe] => SimpleXMLElement Object (
  /bin/sync games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/
  /uucp:/bin/sh proxv:x:13:13:proxv:/bin:/bin/sh www-data:x:33:33:ww
  gnats:
  mess
  /bin/f
  beef-
  /bin/s
  /run/
  sslh:
  /run/

Source of: http://192.168.1.102/testXML.php - Iceweasel

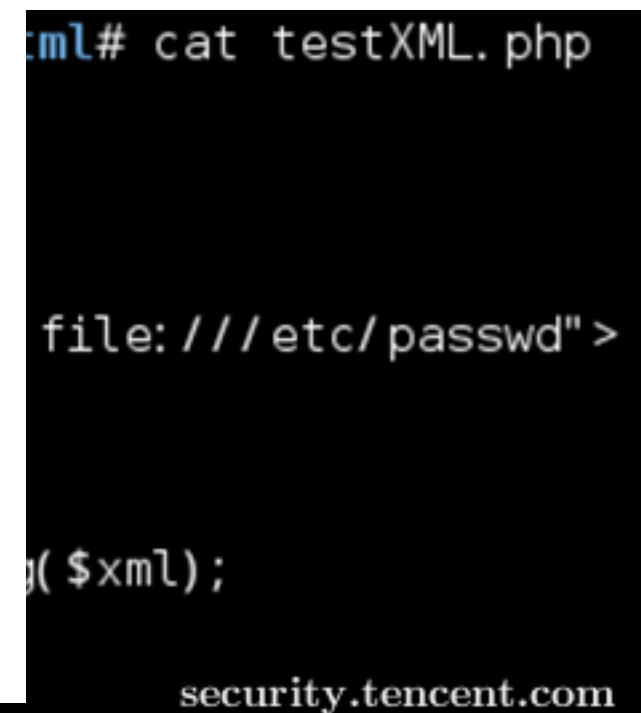
File Edit View Help

1 SimpleXMLElement Object
2 (
3   [xxe] => SimpleXMLElement Object
4   (
5     [xxe] => root:x:0:0:root:/root:/bin/bash
6   daemon:x:1:1:daemon:/usr/sbin:/bin/sh
7   bin:x:2:2:bin:/bin:/bin/sh
8   sys:x:3:3:sys:/dev:/bin/sh
9   sync:x:4:65534:sync:/bin:/bin/sync
10  games:x:5:60:games:/usr/games:/bin/sh
11  man:x:6:12:man:/var/cache/man:/bin/sh

```

security.tencent.com

端口、执行远程代码和实施



```

cat testXML.php

<?xml version="1.0" standalone="no"?>
<!DOCTYPE root [
  <![CDATA[
    file:///etc/passwd
  ]]>
]

```

security.tencent.com

Top 5 Broken Access Control 失效的访问控制

未对通过身份验证的用户实施恰当的访问控制。

攻击者可以利用这些缺陷访问未经授权的功能或数据，例如：访问其他用户的帐户、查看敏感文件、修改其他用户的数据、更改访问权限等。

越权与鉴权

Top 6 Security Misconfiguration 安全配置错误

安全配置错误是最常见的安全问题，这通常是由于不安全的默认配置、不完整的临时配置、开源云存储、错误的HTTP标头配置以及包含敏感信息的详细错误信息所造成的。

因此，我们不仅需要对所有的操作系统、框架、库和应用程序进行安全配置，而且必须及时修补和升级它们。

列举目录

Top 7 Cross-Site Scripting 跨站脚本

当应用程序的新网页中包含不受信任的、未经恰当验证或转义的数据时，或者使用可以创建HTML或JavaScript的浏览器API更新现有的网页时，就会出现XSS缺陷。

XSS让攻击者能够在受害者的浏览器中执行脚本，并劫持用户会话、破坏网站或将用户重定向到恶意站点。

Top 8 Insecure deserialization 不安全的反序列化

不安全的反序列化会导致远程代码执行。

即使反序列化缺陷不会导致远程代码执行，攻击者也可以利用它们来执行攻击，包括：重播攻击、注入攻击和特权升级攻击。

Top 9 Using Components With Known Vulnerabilities

使用含有已知漏洞的组件

组件（例如：库、框架和其他软件模块）拥有和应用程序相同的权限。

如果应用程序中含有已知漏洞的组件被攻击者利用，可能会造成严重的数据丢失或服务器接管。同时，使用含有已知漏洞的组件的应用程序和API可能会破坏应用程序防御、造成各种攻击并产生严重影响。

Top 10 Insufficient Logging and Monitoring

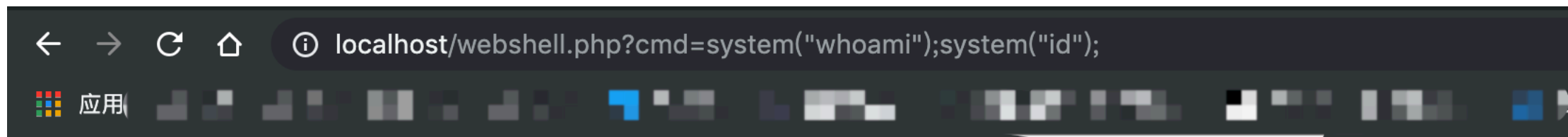
不足的日志记录和监控

不足的日志记录和监控，以及事件响应缺失或无效的集成，使攻击者能够进一步攻击系统、保持持续性或转向更多系统，以及篡改、提取或销毁数据。

大多数缺陷研究显示，缺陷被检测出的时间超过200天，且通常通过外部检测方检测，而不是通过内部流程或监控检测。

Top x 文件上传

文件上传配合服务器解析，轻轻松松拿到 shell



zedd uid=501(zedd) gid=20(staff)

groups=20(staff),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),5

不要相信用户的任何输入!

Top x plus 一次注入

什么过滤都没有，什么防御都没有，使用字符串拼接的方式

```
$sql = 'SELECT * from users where username = '. GET['username'] . ' and password = '. $GET['password'];
```

直接被插入脏数据，例如 username = admin' # & password = 1

Sql 语句变成了

```
SELECT * from users where username = 'admin' # and password = '1';
```

Top x plus 二次注入

首先是在登录注册处进行完美的过滤防注入

```
$sql = "SELECT * FROM users WHERE username='$username' and password='$password'";
```

但是在查询的时候并没有检查数据，导致直接用了脏数据进行查询。

```
$sql = "SELECT * FROM users WHERE username='$username'";
```

这时我们就可以注册一个用户名为 admin ‘ # 的用户

这样第一条因为有完美的防注入，将原本的字符串插入到了数据库里面，但是在第二条语句查询的时候，sql 语句就变成了

```
SELECT * from users where username='admin' #';
```

这样就查询到了 admin 用户的数据了。

Top x plus++

为什么 web 站点更容易受攻击?

低成本, 易上手, Metasploit, Kali Linux



等劳资看完这两本书马上黑你电脑

知识面，决定看到的攻击面有多广。
知识链，决定发动的杀伤链有多深。

Know it, hack it.
Thanks for listening.
Q&A