Summary of PHP code audit experience

wisdomtree / 2018-04-04 20:50:36 / 浏览数 1350 安全技术 技术讨论 顶(4) 踩(0)

从上学期开始正式接触代码审计。从一开始的一脸懵逼，到后来的soga，再到现在的mmp。这段时间也是从代码审计的过程中学到了许多。刚写完漏洞报告，不想看jav web，就顺便把这篇想写很久的博客写出来吧。

why

代码审计的价值要从两个方面来阐述：技术价值和其余价值

技术价值

一项技术的存在必然有其存在的合理意义。否则它早就被世人所淘汰。
我认为对于一个web安全研究人员来说，代码审计是必须掌握的基本技能。
熟悉掌握代码审计，有助在进行黑盒渗透测试时合理猜测后端的执行流程，还能了解各个cms的大致设计理念，从而在对相关网站进行渗透时达到事半功倍的效果。

此外熟练掌握代码审计也有助更好的巩固一门后端语言。以我自己为例，当时学习php时也就学完基本语法，说是有学习反序列化，可是合上书以后什么也不会。真正开始熟

其余价值

其余价值就有意思了。
首先，阿里先知在悬赏通用漏洞。金额还是很诱人的。我自己也通过这一技能实现了现阶段的经济独立。果然如古人说：金钱是第一驱动力——我也不知道谁说的:)
其次，就是各大公司的src了。假设挖到一个"dz
x全版本前台无条件getshell"，国内有那么多大互联网公司都在用dz，就可以开开心心的在各大互联网公司的SRC下留名。

what

代码审计是什么？古人云：工欲善其事必先利其器。在开始代码审计之前，首先需要知道这是个什么玩意。
我不喜欢用别人的总结。我就以我自己对于代码审计的理解，尝试对其进行一个描述：

```
■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■....
■■■■■tricks■■■■■■■■■■■■

■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■tricks■
■■■■php■■■■■■■PHP■■■■■■■■■■■■■■■■

■■■■■
■■php■■■■■■■■■■■■■■■■■■■■■■
```

三个层次难度依次递增，而且给我的感觉还是指数型递增。。。目前我也就勉强符合第一层次。

how

由于自己的审计水平也不是很高，这里提到的审计方法仅仅适用于第一层次。

在拿到一套cms时，不急着看代码。先利用搜索引擎寻找已公开的漏洞，尝试复现。
这样做的好处诸多，首先能避免和别人撞洞。其次，在复现漏洞的过程中，我们可以学习前辈在挖掘该漏洞时的思维方式。最后一个，也是我认为最重要的：我们在利用漏洞

在对已有的漏洞复现完毕之后，可以正式开始进行代码审计。到这里需要对审计方法进行一个选择：
是从代码下手还是从网站已有的功能点入手？
这个选择因人而异，就我个人而言。我更喜欢直接看代码。定位危险函数，逐步往外跟，直至跟踪到用户可接触到的页面。
这一行为就是我审计原则的一种映射：

```
■■■■■■■■■■
```

根据该原则，我们可推出如下公式:

```
if(■■■■■■) ■■■■;
■■■■if(■■eval■■■■■■■■■■) ■■■■■■;
```

事实上，根据该公式，我的确挖到dedecms后台代码执行漏洞。

但是有时候处于种种原因，我们在代码中无法进行代码追踪，或者无处下手的时候。我一般会选择从网站功能下手。

当从网站功能下手时，又面临着一个选择。是从前台开始测试还是后台开始测试？
从危害性来说，毫无疑问，前台漏洞的危害性远大于后台漏洞。（同一漏洞类型）相对应的，前台漏洞的挖掘难度要远远大于后台漏洞。
就新手而言的话，我比较建议放眼全局——能挖到啥算啥。
有的cms的设计理念就是，管理员即上帝，给管理员极高的权限。可操纵web根目录下的任意文件，还可任意使用数据库。遇到这种cms，后台漏洞挖掘的难度极低。有的cm
所以说，要针对每个cms的设计理念进行合理地漏洞挖掘。这样才能避免无处下手的窘境。

丰富的网站功能漏洞测试经验也有助于快速定位漏洞。就比如，我在对cms进行后台测试时，发现每个cms在后台模板功能上，都会赋予模板执行PHP代码的功能。这就是一

details

代码执行
这里贴出几个可以导致代码执行的危险函数：

```
eval(), assert(), preg_replace(), call_user_func(), call_user_func_array(), array_map()
```

具体这几个函数怎么产生代码执行，网上一找一大堆。
需要注意的点是，遇到assert(1==$code)这种形式的还是放弃跟踪吧。。。
每个cms基本都会内置一个php代码执行的功能，若没有做好身份验证，这里也有可能出问题。一般来说，一个cms最少都会有一个代码执行（哪怕是后台），除非不用e

文件包含
像文件包含漏洞，其本质就是包含了非预期文件。也就是说，我们在include，include_once，require，require_once这四个函数中的包含参数可控，即可导致文件包含
这里我给出一条正则，用于快速定位cms中所有的文件包含语句：

```
(include.+|include_.+|require.+|require_.+).+\$.+
```

如果你有足够的耐性，可以跟踪完所有的文件包含语句，我相信一定能挖到不少漏洞。
就我个人感觉，现在cms对于这四个函数的过滤可以说是很严格了。百分之八十的函数里的参数都是写死的。很难去包含一个不同文件夹下的文件。因此，文件包含我一
可以去寻找在包含文件之前对文件执行写入操作的页面。有些cms包含文件前会执行对目标文件进行写入操作。这就很有意思了。如果写入的内容我们可控，这分分钟又
文件包含还有个点就是伪协议的利用。比如，用zip://伪协议包含压缩包中的文件，或者运气好一点，可以使用data://、php://input伪协议，直接搞个代码执行也是很开

注入
注入是web安全老生长谈的话题。近几年的owasp top
1，其危害总所周知。可直接获取数据库中的核心数据。要是数据库的权限稍微高点，还能从数据库里导出个webshell。
正因为其危害巨大，近几年来，常见的sql注入几乎绝迹。很难再从一个百万级的cms中挖掘到前台注入漏洞。
我在挖掘sql注入时，首先会看看该cms是否存在全局转义（一般来说都有），再一个就是看连接数据库时的编码。（若以gbk连接，很有可能导致宽字节注入）很不幸的
这样一来，我们就要转移关注点。可以多关注order by注入，以及limit注入。对于这两个点，有时候过滤的并没有特别严格。
再一个，万一真的遇到对输入的数据进行完美过滤。那就要考虑二次注入。在审计时，很明显能感受到对于数据库中获取的数据，各大cms基本完全信任。不经任何过滤
这里我再给出一条筛选可能出现sql注入的正则：

```
(select.+\$.+)|(update.+\$.+)|(insert.+\$.+)|(delete from.+\$.+)
```

ssrf(Server-Side Request Forgery:服务器端伪造请求)
挖掘ssrf主要关注以下几个函数：

```
file_get_contents■curl_exec■fsockopen■fopen■copy■rename...
```

其中copy和rename是我在无意中发现了。根据大佬们的说法，貌似是PHP底层的问题，我还没看过。。在此不做分析，只提出以上几个函数可导致ssrf。
ssrf的利用不仅仅只是实现内网漫游，在一定条件下还可导致命令执行，getshell，任意文件读取等等。
如ssrf配合redis（root），利用gopher://伪协议便可实现反弹shell。
ssrf利用和伪协议灵活结合使用往往会产生奇效。

XXE注入
XXE注入漏洞往往是由于simplexml_load_file函数的参数过滤不严，导致引入外部实体。产生任意文件读取。具体漏洞分析可自行查询。
若要挖掘XXE注入，跟踪simplexml_load_file函数中的参数即可。

任意文件操作
这个漏洞的范围就很广了。可以说任何操作文件的函数都能导致相应的漏洞。如:unlink()导致任意文件删除，fwrite()导致任意文件写入.....
虽然说有很多函数都可以利用，但是开发人员也不傻，他们对这些函数都有特殊照顾。这些函数里的目标文件路径都是写死的，还限制了外部输入的数据。这就很尴尬了
但是关于写文件的函数还是有点特殊的。相对而言，写入的数据我们还是比较容易控制的。如果直接写入php文件，那我们可以考虑闭合前面的语法逻辑，直接写入web
关于写入文件这个点，如果对输入数据未过滤，还能配合伪协议进行使用，可实现文件读取。

反序列化漏洞
查看类中的魔术方法是否可利用。熟练掌握常见魔术方法的意义及调用场景。还有就是要学会利用PHP的内置类。像常见的__wakeup()方法的绕过这也是必须掌握的。
关于反序列化这块，我还是比较建议去读读PHP反序列化的源码。我也正准备这么做:)

关于漏洞审计的思路及利用方式我所了解的基本就这么多。
由于自身水平有限，不再进行进一步分析。若文中有错误欢迎大佬指出。

End

挖洞时的心态也是很重要的。心态要放平和，挖洞这种事。不是说一定要挖到s2-045那种洞才算挖洞。哪怕只是一个很鸡肋的变量覆盖，也能算是一个漏洞。要学会在挖洞的
在挖到一定层次之后，再去找那些毫无过滤的变量是没有意义的（除了能拿钱）。要尝试去bypass一些东西，学习目标cms的过滤策略，尝试打破它的防御。这个打破防御的
漏洞的危害性和挖掘难度未必存在关系。因此，没必要因为一个未经过滤的注入而沾沾自喜，也无需因为利用各种tricks bypass
waf之后得到鸡肋ssrf而沮丧。只要能在这个过程中学习到新知识，积累tricks，这就够了。这些自己发掘的tricks能在日后审计过程中带给我们极大的方便。
我也不知道有多少人能读到这篇文章。只希望能给花费时间读完文章的人一点点的思维启发。表示没有浪费时间读文章。
想写这篇文章很久了。今天总算写完了。很开心！

上一篇：Coding art in she... 下一篇：利用PowerShell诊断脚本执...

1. 0 条回复
   - 动动手指，沙发就是你的了！