Wordpress安全架构分析

lorexxar / 2017-10-27 01:11:00 / 浏览数 3606 技术文章 技术文章 顶(0) 踩(0)

作者：LoRexxar'@知道创宇404实验室

0x01 前言

WordPress是一个以PHP和MySQL为平台的自由开源的博客软件和内容管理系统。WordPress具有插件架构和模板系统。Alexa排行前100万的网站中有超过16.7%的网站使

在zoomeye上可以搜索到的wordpress站点超过500万，毫不夸张的说，每时每刻都有数不清楚的人试图从wordpress上挖掘漏洞...

由于前一段时间一直在对wordpress做代码审计，所以今天就对wordpress做一个比较完整的架构安全分析...

0x02 开始

在分析之前，我们可能首先需要熟悉一下wordpress的结构

```
■■wp-admin
■■wp-content
■   ■■languages
■   ■■plugins
■   ■■themes
■■wp-includes
■■index.php
■■wp-login.php
```

- admin目录不用多说了，后台部分的所有代码都在这里。
- content主要是语言、插件、主题等等，也是最容易出问题的部分。
- includes则是一些核心代码，包括前台代码也在这里

除了文件目录结构以外，还有一个比较重要的安全机制，也就是nonce，nonce值是wordpress用于防御csrf攻击的手段，所以在wordpress中，几乎每一个请求都需要带上

0x03 nonce安全机制

出于防御csrf攻击的目的，wordpress引入了nonce安全机制，只有请求中_wpnonce和预期相等，请求才会被处理。

我们一起来从代码里看看

当我们在后台编辑文章的时候，进入`/wp-admin/edit.php line 70`

进入`check_admin_referer`，这里还会传入一个当前行为的属性，跟入`/wp-includes/pluggable.php line 1072`

传入的`_wpnonce`和`action`进入函数`wp_verify_nonce`，跟入`/wp-includes/pluggable.php line 1874`

这里会进行`hash_equals`函数来比对，这个函数不知道是不是wp自己实现的，但是可以肯定的是没办法绕过，我们来看看计算nonce值的几个参数。

```
$expected = substr( wp_hash( $i . '|' . $action . '|' . $uid . '|' . $token, 'nonce'), -12, 10 );
```

- i:忘记是什么了，是个定值
- action：行为属性名，可以被预测，在代码里的不同部分都是固定的
- uid：当前用户的id，由1自增，可以算是可以被预测
- token：最重要的就是这部分

当我们登陆后台时，我们会获得一个cookie，cookie的第一部分是用户名，第三部分就是这里的token值。

我们可以认为这个参数是无法获得的。

当我们试图通过csrf攻击后台，添加管理员等，我们的请求就会被拦截，因为我们没办法通过任何方式获得这个`_wpnonce`值。

但事实上，在wordpress的攻击思路上，很多攻击方式都受限于这个wpnonce，比如后台反射性xss漏洞，但可能是通过编辑文件、提交表单、提交查询等方式触发，那么我

这里举两个例子

Loginizer CSRF漏洞(CVE-2017-12651)

Loginizer是一个wordpress的安全登陆插件，通过多个方面的设置，可以有效的增强wp登陆的安全性，在8月22日，这个插件爆出了一个CSRF漏洞。

我们来看看代码

```
/loginizer/tags/1.3.6/init.php line 1198
```

这里有一个删除黑名单ip和白名单ip的请求，当后台登陆的时候，我们可以通过这个功能来删除黑名单ip。

但是这里并没有做任何的请求来源判断，如果我们构造CSRF请求，就可以删除黑名单中的ip。

这里的修复方式也就是用了刚才提到的_wpnonce机制。

这种方式有效的防止了纯CSRF漏洞的发生。

UpdraftPlus插件的SSRF漏洞

UpdraftPlus是一个wordpress里管理员用于备份网站的插件，在UpdraftPlus插件中存在一个CURL的接口，一般是用来判断网站是否存活的，但是UpdraftPlus本身没有对

当请求形似

```
wp-admin/admin-ajax.php?action=updraft_ajax&subaction=httpget&nonce=2f2f07ce90&uri=http://127.0.0.1&curl=1
```

服务器就会向http://127.0.0.1发起请求。

正常意义上来说，我们可以通过构造敏感链接，使管理员点击来触发。但我们注意到请求中带有nonce参数，这样一来，我们就没办法通过欺骗点击的方式来触发漏洞了。

wordpress的nonce机制从另一个角度防止了这个漏洞的利用。

0x04 Wordpress的过滤机制

除了Wordpress特有的nonce机制以外，Wordpress还有一些和普通cms相同的的基础过滤机制。

和一些cms不同的是，Wordpress并没有对全局变量做任何的处理，而是根据不同的需求封装了多个函数用于处理不同情况下的转义。

对于防止xss的转义

wordpress对于输出点都有着较为严格的输出方式过滤。

```
/wp-includes/formatting.php
```

这个文件定义了所有关于转义部分的函数，其中和xss相关的较多。

```
esc_url()
■■■■url■■■■■■■■■■■■■■■■■■■■■url■■■■■■■■■■■$_context■db■■

esc_js()
■■■■■■■■■js■■■■■■■■" < > &■■■■■■■■■■■■■■

esc_html()
■■■■■■■■■html■■■■■■■■■■■

esc_attr()
■■■■■■■■■■■■■■■■■■■■■■■■■

esc_textarea()
■■■■■■■■■textarea■■■■■■■■■■■■■■

tag_escape()
■■■■■■HTML■■■■■■■■■■■■■■
```

在wordpress主站的所有源码中，所有会输出的地方都会经过这几个函数，有效的避免了xss漏洞出现。

举个例子，当我们编辑文章的时候，页面会返回文章的相关信息，不同位置的信息就会经过不同的转义。

对于sql注入的转义

在Wordpress中，关于sql注入的防御逻辑比较特别。

我们先从代码中找到一个例子来看看

```
/wp-admin/edit.php line 86

$post_ids = $wpdb->get_col( $wpdb->prepare( "SELECT ID FROM $wpdb->posts WHERE post_type=%s AND post_status = %s", $post_type,
```

这里是一个比较典型的从数据存储数据，wordpress自建了一个prepare来拼接sql语句，并且拼接上相应的引号，做部分转义。

当我们传入

```
$post_type = "post";
$post_status = "test'";
```

进入语句

```
$wpdb->prepare( "SELECT ID FROM $wpdb->posts WHERE post_type=%s AND post_status = %s", $post_type, $post_status )
```

进入prepare函数

```
/wp-includes/wp-db.php line 1291

public function prepare( $query, $args ) {
if ( is_null( $query ) )
return;

// This is not meant to be foolproof -- but it will catch obviously incorrect usage.
if ( strpos( $query, '%' ) === false ) {
_doing_it_wrong( 'wpdb::prepare', sprintf( __( 'The query argument of %s must have a placeholder.' ), 'wpdb::prepare()' ), '3.
}

$args = func_get_args();
array_shift( $args );
// If args were passed as an array (as in vsprintf), move them up
if ( isset( $args[0] ) && is_array($args[0]) )
$args = $args[0];
$query = str_replace( "'%s'", '%s', $query ); // in case someone mistakenly already singlequoted it
$query = str_replace( '"%s"', '%s', $query ); // doublequote unquoting
$query = preg_replace( '|(?<!%)%f|' , '%F', $query ); // Force floats to be locale unaware
$query = preg_replace( '|(?<!%)%s|', "'%s'", $query ); // quote the strings, avoiding escaped strings like %%s
array_walk( $args, array( $this, 'escape_by_ref' ) );
return @vsprintf( $query, $args );
}
```

这个函数会读取参数值，然后会在字符串处加上相应的单引号或者双引号，并且在拼接之前，调用escape_by_ref转义参数。

```
public function escape_by_ref( &$string ) {
if ( ! is_float( $string ) )
$string = $this->_real_escape( $string );
}
```

这里的_real_escape函数，就是一些转义函数的封装。

```
    function _real_escape( $string ) {
if ( $this->dbh ) {
if ( $this->use_mysqli ) {
return mysqli_real_escape_string( $this->dbh, $string );
} else {
return mysql_real_escape_string( $string, $this->dbh );
}
}

$class = get_class( $this );
if ( function_exists( '__' ) ) {
/* translators: %s: database access abstraction class, usually wpdb or a class extending wpdb */
_doing_it_wrong( $class, sprintf( __( '%s must set a database connection for use with escaping.' ), $class ), '3.6.0' );
} else {
_doing_it_wrong( $class, sprintf( '%s must set a database connection for use with escaping.', $class ), '3.6.0' );
}
return addslashes( $string );
}
```

这样在返回前，调用vsprintf的时候，post_status的值中的单引号就已经被转义过了。

当然，在代码中经常会不可避免的拼接语句，举个例子。

```
/wp-includes/class-wp-query.php line 2246~2282
```

面对这种大批量的拼接问题，一般会使用esc_sql函数来过滤

这里esc_sql最终也是会调用上面提到的escape函数来转义语句

```
function esc_sql( $data ) {
global $wpdb;
return $wpdb->_escape( $data );
}
```

其实一般意义上来说，只要拼接进入语句的可控参数进入esc_sql函数，就可以认为这里不包含注入点。

但事实就是，总会有一些错误发生。

Wordpress Sqli漏洞

这是一个很精巧的漏洞，具体的漏洞分析可以看文章

https://paper.seebug.org/386/

这里不讨论这个，直接跳过前面的步骤到漏洞核心原理的部分

`wp-includes/meta.php line 365`■

这里我们可以找到漏洞代码

我们可以注意到，当满足条件的时候，字符串会两次进入prepare函数。

当我们输入`22 %1$%s hello`的时候，第一次语句中的占位符%s会被替换为`&#39;%s&#39;`，第二次我们传入的%s又会被替换为`&#39;%s&#39;`,这样输出结果就是`meta_value = &#39;22 %1$&#39;%s&#39; hello&#39;`

紧接着`%1$&#39;%s`会被格式化为`$_thumbnail_id`，这样就会有一个单引号成功的逃逸出来了。

这样，在wordpress的严防死守下，一个sql注入漏洞仍然发生了。

0x05 Wordpress插件安全

其实Wordpress的插件安全一直都是Wordpress的安全体系中最最薄弱的一环，再加上Wordpress本身的超级管理员信任问题，可以说90%的Wordpress安全问题都是出在

我们可以先了解一下Wordpress给api开放的接口，在wordpress的文档中，它推荐wordpress的插件作者通过hook函数来把自定义的接口hook进入原有的功能，甚至重写

也就是说，如果你愿意，你可以通过插件来做任何事情。

从几年前，就不断的有wordpress的插件主题爆出存在后门。

http://www.freebuf.com/articles/web/97990.html
https://paper.seebug.org/140/

事实上，在wordpress插件目录中，wordpress本身并没有做任何的处理，当你的用户权限为超级管理员时，wordpress默认你可以对自己的网站负责，你可以修改插件文件

也正是由于这个原因，一个后台的反射性xss就可以对整个站进行利用。

而Wordpress的插件问题也多数出现在开发者水平的参差不齐上，对很多接口都用了有问题的过滤方式甚至没做任何过滤，这里举个例子。

Wordpress Statistics注入漏洞

Wordpress Statistics在v12.0.7版本的时候，爆出了一个注入漏洞，当一个编辑权限的账户在编辑文章中加入短代码，服务端在处理的时候就会代入sql语句中。

短代码是一个比较特殊的东西，这是Wordpress给出的一个特殊接口，当文章加入短代码时，后台可以通过处理短代码返回部分数据到文章中，就比如文章阅读数等...

当我们传入

```
[wpstatistics stat="searches" time="today" provider="sss' union select 1,sleep(5),3,4,5,6#" format="1111" id="1"]
```

跟入代码`/includes/functions/funstions.php 725`■

然后进入 /includes/functions/funstions.php 622行

这里直接拼接，后面也没有做任何处理。

这个漏洞最后的修复方式就是通过调用esc_sql来转义参数，可见漏洞的产生原因完全是插件开发者的问题。

0x06 总结

上面稀里哗啦的讲了一大堆东西，但其实可以说Wordpress的安全架构还是非常安全的，对于Wordpress主站来说，最近爆出的漏洞大部分都是信任链的问题，在wordpre

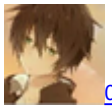
https://www.seebug.org/vuldb/ssvid-92845

而在实际生活中，wordpress的漏洞重点集中在插件上面...在wordpress的插件上多做注意可能最重要的一点。

原文地址：https://paper.seebug.org/422/

上一篇：Typecho 事件始末 下一篇：前段防御从入门到弃坑--CSP变迁

1. 4 条回复


0r3ak 2017-10-27 02:11:12

辛苦了

0 回复Ta


hades 2017-10-27 02:22:11

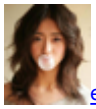
文章还是挺不错的~

0 回复Ta


steven1881 2017-10-27 06:52:47

这文章值得借鉴参考，谢谢

0 回复Ta


evil77 2017-10-27 08:06:50

比较全面的文了 赞

0 回复Ta

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板