SA-CORE-2019-008 Drupal访问绕过漏洞分析

tornado / 2019-07-25 09:26:00 / 浏览数 4707 安全技术 漏洞分析 顶(0) 踩(0)

0x01 概述

7月17日，Drupal官方发布Drupal核心安全更新公告，修复了一个访问绕过漏洞，攻击者可以在未授权的情况下发布/修改/删除文章，CVE编号CVE-2019-6342

公告地址：https://www.drupal.org/sa-core-2019-008

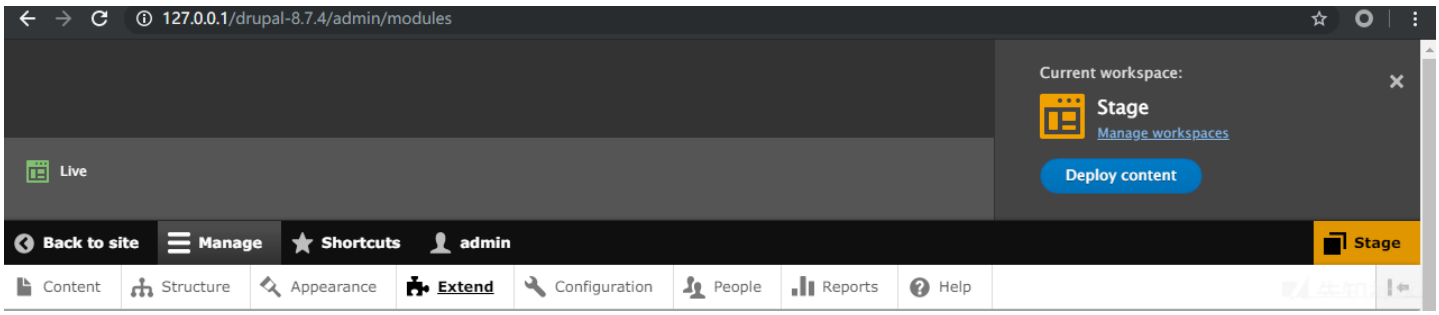
0x02 受影响的版本

- Drupal Version == 8.7.4

0x03 漏洞复现

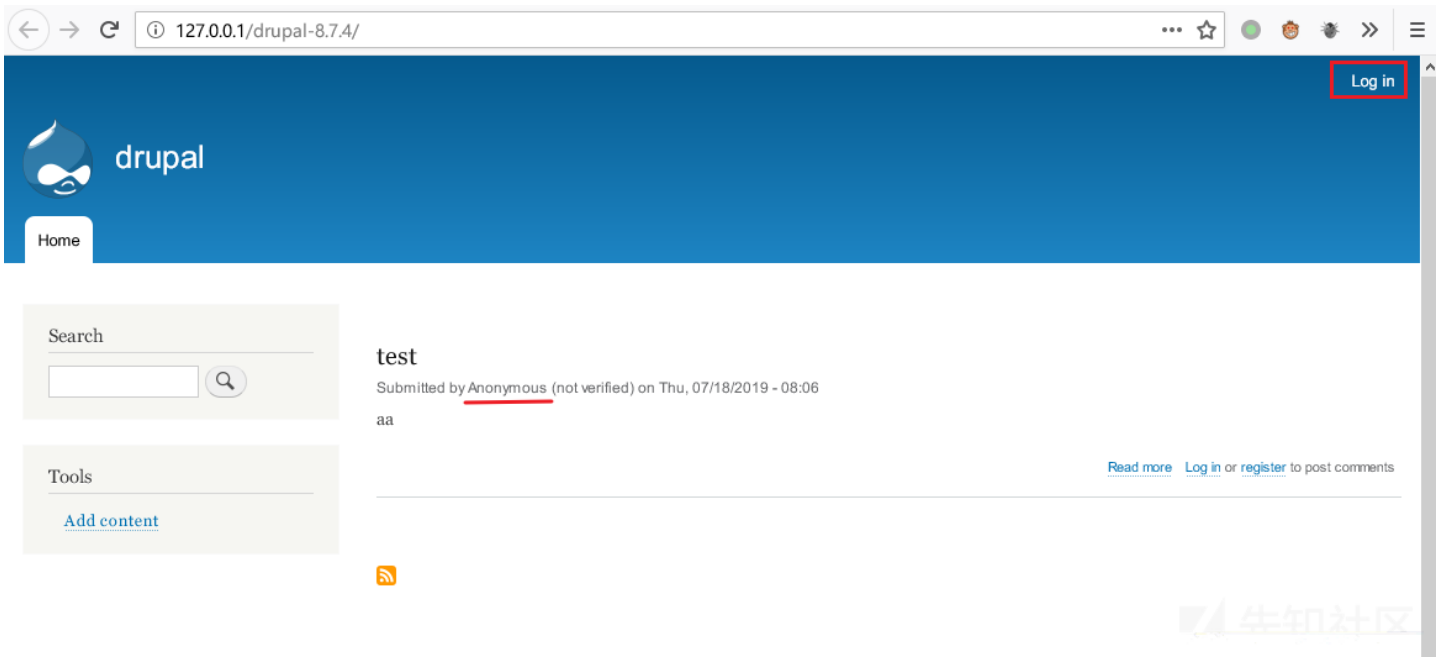
安装Drupal 8.7.4版本，登录管理员账户，进入后台/admin/modules，勾选Workspaces模块并安装

在页面上方出现如下页面则安装成功，管理员可以切换Stage模式或者Live模式



另外开启一个浏览器访问首页（未登录任何账户），访问http://127.0.0.1/drupal-8.7.4/node/add/article

可直接添加文章，无需作者或管理员权限。



受影响操作包括基本文章操作（添加、修改、删除、上传附件等）

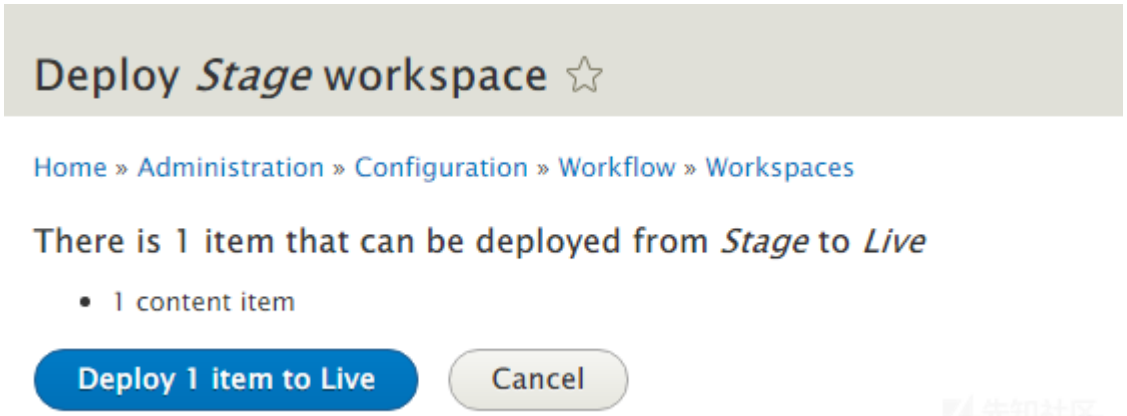
0x04 漏洞分析

Workspaces的功能

Workspaces是Drupal 8.6核心新增的实验模块，主要功能是方便管理员一次性发布/修改多个内容。

Workspaces有两种模式，分别为Stage模式和Live模式，，默认为Live模式，两者的区别在于：

- Stage模式下修改内容不会及时更新，所有文章修改完毕后管理员可以通过Deploy to Live发布到实际环境，相当于一个暂存区；



- Live下更新是即时的，发布后站点内容立即更新。

在这两种模式下，由于编码失误导致存在一个缺陷：匿名用户无需登录即可创建/发布/修改/删除文章，问题点出现在权限鉴定模块EntityAccess下。

漏洞分析

当用户发起请求时，会根据当前操作回调相关权限检查模块对当前用户权限进行检查，请求调用为事件监听器(EventListener)的RouterListener类，在其onKernelRe

```
protected function performCheck($service_id, ArgumentsResolverInterface $arguments_resolver) {
  $callable = $this->checkProvider->loadCheck($service_id);
  $arguments = $arguments_resolver->getArguments($callable);
  /** @var \Drupal\Core\Access\AccessResultInterface $service_access **/
  $service_access = call_user_func_array($callable, $arguments);

  if (!$service_access instanceof AccessResultInterface) {
    throw new AccessException( message: "Access error in $service_id. Access services must return an object that implements
  }

  return $service_access;
}
```

例如发布文章时回调的是access_check.node.add，相关方法在NodeAccessControlHandler控制器中定义，这个控制器继承自EntityAccessControlHandler，

```
$function = module . '_' . $hook;
```

例如此处回调的是workspaces_entity_create_access()方法，进入到Workspaces中。

```
/**
 * Implements hook_entity_create_access().
 *
 * @see \Drupal\workspaces\EntityAccess
 */
function workspaces_entity_create_access(AccountInterface $account, array $context, $entity_bundle) {
  return \Drupal::service( id: 'class_resolver')
    ->getInstanceFromDefinition(EntityAccess::class)
    ->entityCreateAccess($account, $context, $entity_bundle);
}
```

在调用entityCreateAccess()方法时有一个关键操作bypassAccessResult

```
public function entityCreateAccess(AccountInterface $account, array $context, $entity_bundle) {
  // Workspaces themselves are handled by their own access handler and we
  // should not try to do any access checks for entity types that can not
  // belong to a workspace.
  $entity_type = $this->entityTypeManager->getDefinition($context['entity_type_id']);
  if ($entity_type->id() === 'workspace' || !$this->workspaceManager->isEntityTypeSupported($entity_type)) {
    return AccessResult::neutral();
  }

  return $this->bypassAccessResult($account);
}
```

bypassAccessResult()方法是一个检查用户是否有"[redacted](bypass node access)"的操作，是Workspaces中特有的，这个方法决定了"如果用户在各自的激活的工作区中，那么他将拥有所有权限"，这里的所有权限指文章相关的增删改操作。

这个权限虽然奇怪但确实是一个设计好的功能，正常操作应该在后台admin/people/permissions中配置好用户是否拥有这个权限，默认情况下匿名用户和认证用户都没

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
Select method for cancelling account *Warning: Give to trusted roles only; this permission has security implications.*	☐	☐	☑
View user information	☐	☐	☑
Views UI			
Administer views *Warning: Give to trusted roles only; this permission has security implications.*	☐	☐	☑
Workspaces			
Administer workspaces	☐	☐	☑
Bypass content entity access in own workspace *Warning: Give to trusted roles only; this permission has security implications.* Allow all Edit/Update/Delete permissions for all content entities in a workspace owned by the user.	☐	☐	☑
Create a new workspace	☐	☐	☑
Delete any workspace	☐	☐	☑
Delete own workspace	☐	☐	☑
Edit any workspace	☐	☐	☑
Edit own workspace	☐	☐	☑
View any workspace	☐	☐	☑
View own workspace	☐	☐	☑

当开启了Bypass content entity access in own workspace权限后用户才可以在未登录的情况下发布/删除文章，而此次漏洞就绕过了这个配置，默认情况下进行了越权操作。

具体分析一下bypassAccessResult()的实现，整个过程返回的是AccessResultAllowed对象或者AccessResultNeutral对象，所谓"中立"是因为后续还可能会对结

```php
protected function bypassAccessResult(AccountInterface $account)
{
    // This approach assumes that the current "global" active workspace is
    // correct, i.e. if you're "in" a given workspace then you get ALL THE PERMS
    // to ALL THE THINGS! That's why this is a dangerous permission.
    $active_workspace = $this->workspaceManager->getActiveWorkspace();

    $owner_has_access = AccessResult::allowedIf( condition: $active_workspace->getOwnerId() == $account->id())
        ->cachePerUser()->addCacheableDependency($active_workspace);
    $access_bypass = AccessResult::allowedIfHasPermission($account, permission: 'bypass entity access own workspace');
    return $owner_has_access->orIf($access_bypass);
}
```

首先获取了当前激活的工作区，然后通过allowedIf判断当前用户是否有权限，随后这些数据存入缓存，包括缓存内容、缓存标签和过期时间。然后再经过一次allowedIf

接下来就是出现问题的地方

```
$owner_has_access->orIf(access_bypass);
```

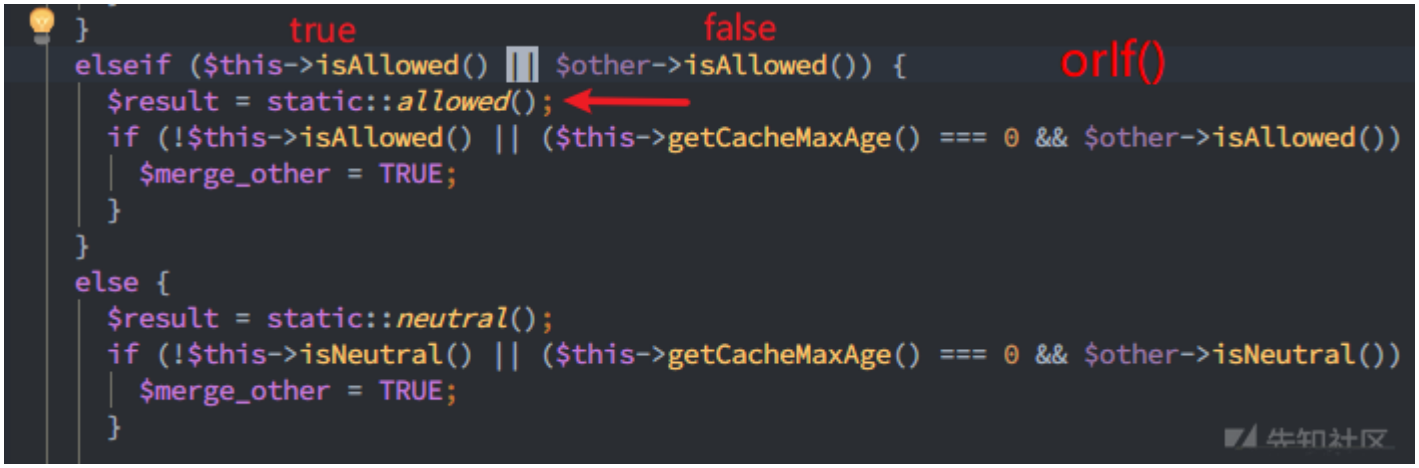
通过补丁可以发现漏洞就修补了这行语句，把orIf换成了andIf

```
6 modules/workspaces/src/EntityAccess.php
@@ -124,10 +124,8 @@ protected function bypassAccessResult(AccountInterface $account) {
        // to ALL THE THINGS! That's why this is a dangerous permission.
        $active_workspace = $this->workspaceManager->getActiveWorkspace();

-       $owner_has_access = AccessResult::allowedIf($active_workspace->getOwnerId() == $account->id())
-         ->cachePerUser()->addCacheableDependency($active_workspace);
-       $access_bypass = AccessResult::allowedIfHasPermission($account, 'bypass entity access own workspace');
-       return $owner_has_access->orIf($access_bypass);
+       return AccessResult::allowedIf($active_workspace->getOwnerId() == $account->id())->cachePerUser()->addCacheableDependency($active_workspac
+         ->andIf(AccessResult::allowedIfHasPermission($account, 'bypass entity access own workspace'));
      }

    }
```

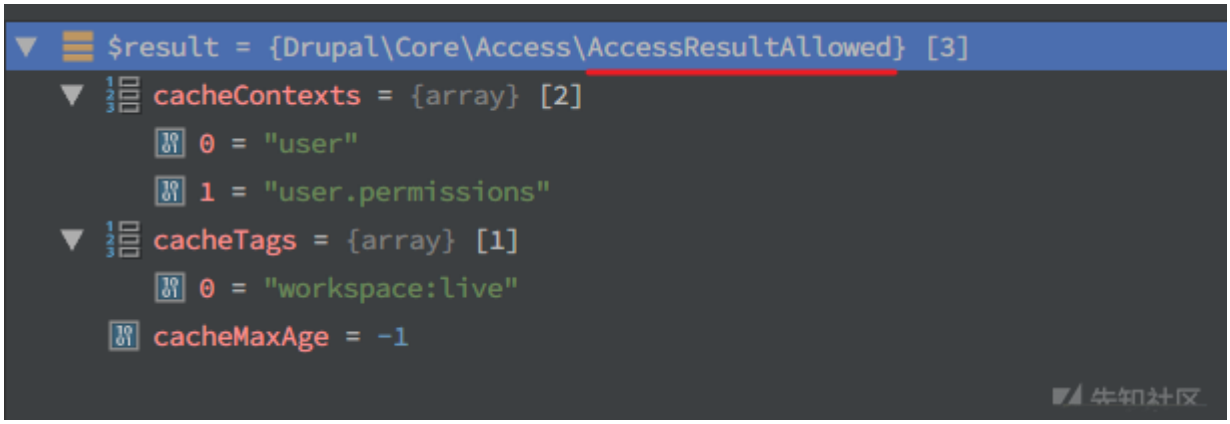
这两个方法的设计逻辑比较复杂，最主要的功能是对一个如果返回为"中立"的结果做后续判断，如果采用orIf方法合并，那么是否允许由调用者决定；如果以andIf方法合并，

具体到此次漏洞上的区别如下方图片所示：

- `orIf()`



返回的是`AccessResultAllowed`对象



- `andIf()`

```
}                    true                false
elseif ($this->isAllowed() && $other->isAllowed()) {        andIf()
  $result = static::allowed();
  $merge_other = TRUE;
}
else {
  $result = static::neutral();
  if (!$this->isNeutral()) {
    $merge_other = TRUE;
    if ($other instanceof AccessResultReasonInterface) {
      $result->setReason($other->getReason());
    }
  }
```

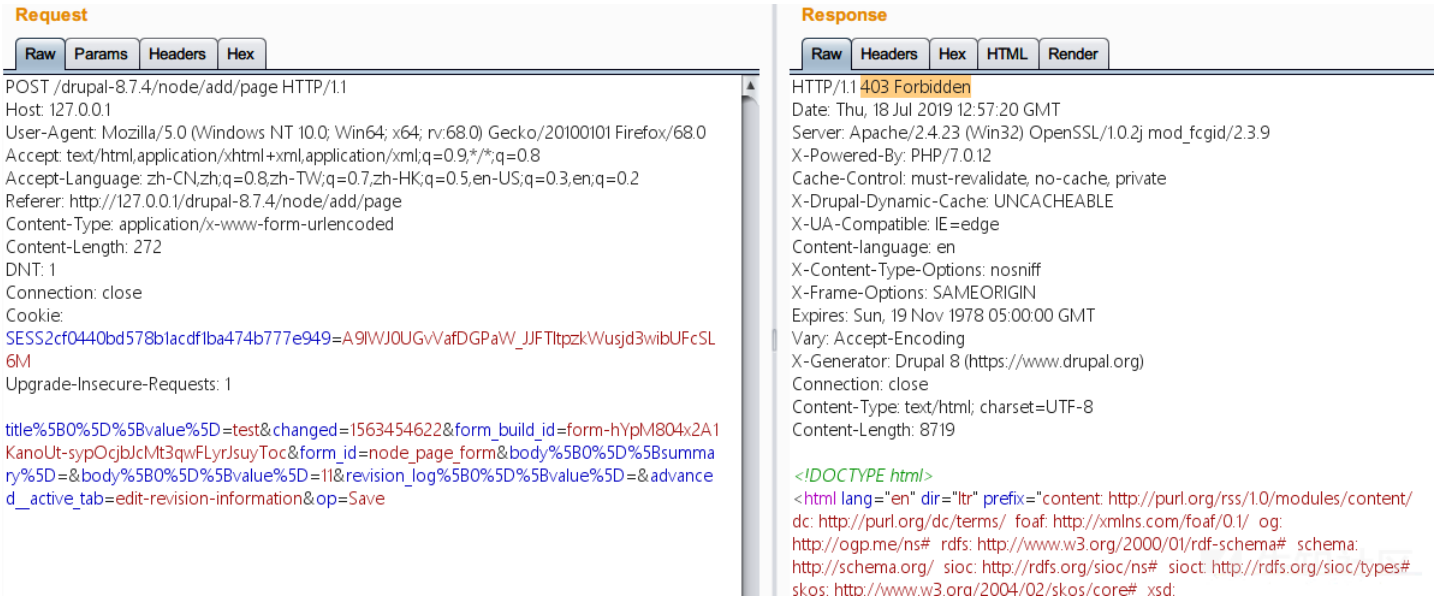
返回的是AccessResultNeutral对象

```
$result = {Drupal\Core\Access\AccessResultNeutral} [4]
  reason = "The 'bypass entity access own workspace' permission is required."
  cacheContexts = {array} [2]
    0 = "user"
    1 = "user.permissions"
  cacheTags = {array} [1]
    0 = "workspace:live"
  cacheMaxAge = -1
```

在检查完毕后会回到AccessAwareRouter->checkAccess()方法，在该方法中对返回结果进行了判断，AccessResultNeutral的isAllowed()返回false，因此会

```
protected function checkAccess(Request $request) {
  // The cacheability (if any) of this request's access check result must be
  // applied to the response.
  $access_result = $this->accessManager->checkRequest($request, $this->account, return_as_object: TRUE);
  // Allow a master request to set the access result for a subrequest: if an
  // access result attribute is already set, don't overwrite it.
  if (!$request->attributes->has( key: AccessAwareRouterInterface::ACCESS_RESULT)) {
    $request->attributes->set(AccessAwareRouterInterface::ACCESS_RESULT, $access_result);
  }
  if (!$access_result->isAllowed()) {
    if ($access_result instanceof CacheableDependencyInterface && $request->isMethodCacheable()) {
      throw new CacheableAccessDeniedHttpException($access_result, message: $access_result instanceof Access
    }
    else {
      throw new AccessDeniedHttpException( message: $access_result instanceof AccessResultReasonInterface ? $
    }
  }
}
```

返回到页面上则是Access denied



更新补丁后只有在开启后台匿名用户权限后才能进行文章操作，该选项默认不开启。

相关调用栈为

```
Drupal\workspaces\EntityAccess->bypassAccessResult()
Drupal\workspaces\EntityAccess->entityCreateAccess()
...
Drupal\Core\Extension\ModuleHandler->invokeAll()
Drupal\node\NodeAccessControlHandler->createAccess()
Drupal\node\Access\NodeAddAccessCheck->access()
Drupal\Core\Access\AccessManager->performCheck()
Drupal\Core\Routing\AccessAwareRouter->checkAccess()
Drupal\Core\Routing\AccessAwareRouter->matchRequest()
Symfony\Component\HttpKernel\EventListener\RouterListener->onKernelRequest()
...
DrupalKernel.php:693, Drupal\Core\DrupalKernel->handle()
index.php:19, {main}()
```

0x05 总结

此次漏洞出现在设计过程的一个疏忽，在默认没有分配权限的情况下用户可以绕过权限检查进行发布/删除/修改文章操作，但由于该漏洞仅影响Drupal 8.7.4版本，并且需要开启Workspaces模块，这又是一个实验功能，默认不启用，因此漏洞影响减弱了不少，用户可以升级Drupal版本或者关闭Workspaces模块以消除漏

上一篇：PHPCMS漏洞分析合集(上) 下一篇：Linux kernel Expl...

1. 0 条回复
   - 动动手指，沙发就是你的了！