Defcon 2018 Qualify: Easy Pisy writeup

原创

LiRiu 于 2021-04-26 13:55:37 发布 126 收藏

分类专栏： 非专业知识积累 文章标签： 安全漏洞 密码学 区块链

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/cemao4548/article/details/116153072

版权

非专业知识积累 专栏收录该内容

7 篇文章 0 订阅

订阅专栏

文章目录

Defcon 2018 Qualify: Easy Pisy

1. Source Code

题目给了俩PHP：

- `execute.php`

```php
<?php

include 'common.php';

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
  print highlight_string(file_get_contents("execute.php"), TRUE);
  exit(0);
}

$keys = get_keys();
$privkey = $keys[0];
$pubkey = $keys[1];

$file_info = $_FILES['userfile'];
check_uploaded_file($file_info);

$data = file_get_contents($file_info['tmp_name']);
$signature = hex2bin($_POST['signature']);
if (openssl_verify($data, $signature, $pubkey)) {
  print 'Signature is OK.<br/>';
} else {
  die('Bad signature.');
}

$text = pdf_to_text($file_info['tmp_name']);
print "Text: \"$text\"<br/>";

$execute_query = "EXECUTE ";
$echo_query = "ECHO ";
if (substr($text, 0, strlen($execute_query)) === $execute_query) {
  $payload = substr($text, strlen($execute_query));
  print "About to execute: \"$payload\".<br/>";
  $out = shell_exec($payload);
  print "Output: $out";
} else if (substr($text, 0, strlen($echo_query)) === $echo_query) {
  $payload = substr($text, strlen($echo_query));
  print "About to echo: \"$payload\".<br/>";
  echo $payload;
} else {
  print "I can't recognize the command type. Go away.<br/>";
}

?>
```

- `shellme.php`

```php
<?php

include 'common.php';

if ($_SERVER['REQUEST_METHOD'] === 'GET') {
  print highlight_string(file_get_contents("sign.php"), TRUE);
  exit(0);
}

$keys = get_keys();
$privkey = $keys[0];
$pubkey = $keys[1];

if ($privkey === FALSE || $pubkey === FALSE) {
  die("Could not load keys. Contact admin.<br/>");
}

$file_info = $_FILES['userfile'];
check_uploaded_file($file_info);

$text = pdf_to_text($file_info['tmp_name']);
print "Extracted text: \"$text\"<br/>";

$execute_query = "EXECUTE ";
$echo_query = "ECHO ";
if (substr($text, 0, strlen($execute_query)) === $execute_query) {
  print "I don't sign EXECUTE commands. Go away.<br/>";
} else if (substr($text, 0, strlen($echo_query)) === $echo_query) {
  print "I'm OK with ECHO commands. Here is the signature: <br/>";
  $data = file_get_contents($file_info['tmp_name']);
  openssl_sign($data, $signature, $privkey);
  print bin2hex($signature);
} else {
  print "I can't recognize the command type. Go away.<br/>";
}

?>
```

2. Writeup

shellme.php是入口，用户将首位为`ECHO`的pdf上传，得到`signature`

将`signature`和pdf上传给`execute.php`

- 首先检查`signature`一致性
- 如果pdf首位为`ECHO`则显示PDF内容
- 如果pdf首位为`EXECUTE`则执行PDF内容

这道题的考点在`检查signature一致性`这里，
注意题目中验证签名用的函数是`openssl_verify`
该函数默认使用`sha1`算法进行签名
由于`sha1`不安全，我们可以使用工具碰撞出首位为`EXECUTE`而非`ECHO`,但`signature`一致的pdf内容,从而RCE。

3. Info

challenge source code https://gist.github.com/janmasarik/232381eec3918313b5b4d2c20ca1ed0f

CTF Time writeup https://ctftime.org/task/6094

Writeup provider teams:

- mhackeroni
- EmpireCTF
- tuna$laves

Author：Not Found, But Maybe

- https://gist.github.com/janmasarik/ 就这哥们儿有题目源码
- https://github.com/nneonneo/ 这位哥在题目之前写了对应的工具
- Marc Stevens(CWI Amsterdam) 这位是《Announcing the first SHA1 collision》一作
- Elie Bursztein (Google) 《Announcing the first SHA1 collision》二作

writeup for study: https://github.com/0e85dc6eaf/CTF-Writeups/tree/05ad6a9ecbc19ceb8890f4581dfee36f16d164aa/DEF%20CON%20CTF%20Qualifier%202018/Easy%20Pisy#easy-pisy

tools used:

- https://github.com/nneonneo/sha1collider
- https://github.com/cr-marcstevens/sha1collisiondetection

4. Analysis of Author

由于还不能确定这道题的出题人是谁，就先面向目前可能性比较大的前四位分析。

4.1 janmasarik

- languages
  - python\go\php
- 社交媒体
  - https://twitter.com/s14ve
  - https://masarik.sh/
- Recent
  - resolvers DNS拦截的检查工具？20210416
  - low-hanging 漏洞扫描器
  - bucketsperm 用上面的low-hanging做的 云存储桶 权限 漏洞扫描器？
  - Automating Bug Bounty 论文:自动挖漏洞？这个好像专门挖云存储漏洞，应该和bucketsperm有联动

jan.masarik先生19年至今的动态主要在云存储漏洞的自动挖掘上。

4.2 nneonneo (Robert Xiao)

- languages
  - python
- 社交媒体
  - https://twitter.com/nneonneo
  - https://www.robertxiao.ca
- Recent
  - Ghidra 逆向工具
  - iOS-Torrent-Client
  - ios-sock
  - ntfsrecover

nneonneo 大佬2020年底的github动态都是在搞RE和PWN，做的研究工作都比较偏底层。应该不会出web的题目⑧
而且nneonneo大佬似乎是CTFer而非出题人，但他是怎么在出题之前设计出做题工具的呢
说明他看了Marc的文章…

4.3 Marc Stevens

languages

- C/C++

社交媒体

- Github
- google学术
- 个人博客

Recent

- Non-interactive cryptographic timestamping based on verifiable delay functions 2020
- Real World Cryptanalysis 2019
- The general sieve kernel and new records in lattice reduction
- sha1collisiondetection 但这个似乎还是18年defcon的EasyPisy考点
- snippets Marc个人的脚本收集仓库

学习一下`Non-interactive cryptographic timestamping based on verifiable delay functions`

仔细看一下`sha1collisiondetection`的更新细节，也许更新了新的知识点可以出题？

这位老师还写了一篇区块链的文章 On immutability of blockchains

4.4 Elie Bursztein (Google)

- languages
  - Python/PHP
- 社交媒体
  - 个人博客
  - Github
  - Google Scholar
- Recent
  - Sok: Hate, harassment, and the changing landscape of online abuse 2021
  - Spotlight: Malware Lead Generation at Scale
  - Who is targeted by email-based phishing and malware? Measuring factors that differentiate risk
  - Coinpolice: Detecting hidden cryptojacking attacks with neural networks

尽管是《Announcing the first SHA1 collision》二作，Elie的主要精力还是投入在了Phishing、Malware、Abuse等领域。

> 我比较喜欢他博客里这些没有数学公式的科普文章，像看小说一样欢乐。我女朋友则喜欢有逻辑推导、数学演算的东西，会更优美、更艺术一些…

从领域相匹配度的角度分析,Elie也比较符合EasyPisy出题人的画像。
不过博客里几乎没看到有关CTF的内容。

- A Hacker's guide to reducing side-channel attack surfaces using deep-learning

这个"基于深度学习来防御旁路攻击"的思路宣讲于`Defcon 28`&`Black Hat USA 2020` // TODO
密码学旁路攻击: 基于旁路攻击的AES算法中间变量脆弱点 // TODO

5. 语言中的签名函数

先找一下各语言标准库中的签名函数，再进一步找第三方库中引入的签名函数。

5.1 php

https://www.php.net/manual/zh/function.hash-hmac-file.php
https://segmentfault.com/a/1190000020201103
https://www.php.net/manual/zh/ref.openssl.php

5.1.1 standards library

5.1.1.1 Hash

PHP 5.1.2开始，Hash成为内置拓展。
PHP 7.4.0开始，Hash成为核心拓展，可以直接使用其函数。
Hash 函数

- hash_algos — 返回已注册的哈希算法列表
- hash_copy — 拷贝哈希运算上下文
- hash_equals — 可防止时序攻击的字符串比较
- hash_file — 给指定文件的内容生成哈希值
- hash_final — 结束增量哈希，并且返回摘要结果
- hash_hkdf — Generate a HKDF key derivation of a supplied key input
- hash_hmac_algos — Return a list of registered hashing algorithms suitable for hash_hmac
- hash_hmac_file — 使用 HMAC 方法和给定文件的内容生成带密钥的哈希值
- hash_hmac — 使用 HMAC 方法生成带有密钥的哈希值
- hash_init — 初始化增量哈希运算上下文
- hash_pbkdf2 — 生成所提供密码的 PBKDF2 密钥导出
- hash_update_file — 从文件向活跃的哈希运算上下文中填充数据
- hash_update_stream — 从打开的流向活跃的哈希运算上下文中填充数据
- hash_update — 向活跃的哈希运算上下文中填充数据
- hash — 生成哈希值 （消息摘要）

> Hash系列函数的 `algo` 参数没有默认值。

5.1.1.2 密码散列算法函数

下列函数可以直接调用

> 如果要使用 Argon2 密码哈希，PHP 必须用 --with-password-argon2[=DIR] 配置选项和 libargon2 构建。
> 密码散列算法函数

- password_algos — Get available password hashing algorithm IDs
- password_get_info — 返回指定散列（hash）的相关信息
- password_hash — 创建密码的散列（hash）
- password_needs_rehash — 检测散列值是否匹配指定的选项
- password_verify — 验证密码是否和散列值匹配
  `password_hash` 所有可用的签名算法如下
- PASSWORD_BCRYPT (default) 学习链接
- PASSWORD_ARGON2I 学习链接
- PASSWORD_ARGON2ID
- PASSWORD_ARGON2_DEFAULT_MEMORY_COST
- PASSWORD_ARGON2_DEFAULT_TIME_COST
- PASSWORD_ARGON2_DEFAULT_THREADS
  慢哈希函数学习链接 // 这啥啊，看不懂…等女朋友看完讲一下吧…
  ARGON2学习链接

5.1.2 第三方库

5.1.2.1 OpenSSL

要使用 PHP 的 OpenSSL 模块时，须使用 --with-openssl[=DIR] 参数来编译 PHP。
OpenSSL库所有的函数如下

```
openssl_cipher_iv_length – 获取密码iv长度
openssl_cms_decrypt – Decrypt a CMS message
openssl_cms_encrypt – Encrypt a CMS message
openssl_cms_read – Export the CMS file to an array of PEM certificates
openssl_cms_sign – Sign a file
openssl_cms_verify – Verify a CMS signature
openssl_csr_export_to_file – 将CSR导出到文件
openssl_csr_export – 将CSR作为字符串导出
openssl_csr_get_public_key – 返回CSR的公钥
openssl_csr_get_subject – 返回CSR的主题
openssl_csr_new – 生成一个 CSR
openssl_csr_sign – 用另一个证书签署 CSR (或者本身) 并且生成一个证书
openssl_decrypt – 解密数据
openssl_dh_compute_key – 计算远程DH密钥(公钥)和本地DH密钥的共享密钥
openssl_digest – 计算摘要
openssl_encrypt – 加密数据
openssl_error_string – 返回 openSSL 错误消息
openssl_free_key – 释放密钥资源
openssl_get_cert_locations – 检索可用的证书位置
openssl_get_cipher_methods – 获取可用的加密算法
openssl_get_curve_names – 获得ECC的可用曲线名称列表
openssl_get_md_methods – 获取可用的摘要算法
openssl_get_privatekey – 别名 openssl_pkey_get_private
openssl_get_publickey – 别名 openssl_pkey_get_public
openssl_open – 打开密封的数据
openssl_pbkdf2 – 生成一个 PKCS5 v2 PBKDF2 字符串
openssl_pkcs12_export_to_file – 输出一个 PKCS#12 兼容的证书存储文件
openssl_pkcs12_export – 将 PKCS#12 兼容证书存储文件导出到变量
openssl_pkcs12_read – 将 PKCS#12 证书存储区解析到数组中
openssl_pkcs7_decrypt – 解密一个 S/MIME 加密的消息
openssl_pkcs7_encrypt – 加密一个 S/MIME 消息
openssl_pkcs7_read – 将PKCS7文件导出为PEM格式证书的数组
openssl_pkcs7_sign – 对一个 S/MIME 消息进行签名
openssl_pkcs7_verify – 校验一个已签名的 S/MIME 消息的签名
openssl_pkey_derive – Computes shared secret for public value of remote and local DH or ECDH key
openssl_pkey_export_to_file – 将密钥导出到文件中
openssl_pkey_export – 将一个密钥的可输出表示转换为字符串
openssl_pkey_free – 释放一个私钥
openssl_pkey_get_details – 返回包含密钥详情的数组
openssl_pkey_get_private – 获取私钥
openssl_pkey_get_public – 从证书中解析公钥，以供使用。
openssl_pkey_new – 生成一个新的私钥
openssl_private_decrypt – 使用私钥解密数据
openssl_private_encrypt – 使用私钥加密数据
openssl_public_decrypt – 使用公钥解密数据
openssl_public_encrypt – 使用公钥加密数据
openssl_random_pseudo_bytes – 生成一个伪随机字节串
openssl_seal – 密封 (加密) 数据
openssl_sign – Generate signature
openssl_spki_export_challenge – 导出与签名公钥和挑战相关的挑战字符串
openssl_spki_export – 通过签名公钥和挑战导出一个可用的PEM格式的公钥
openssl_spki_new – 生成一个新的签名公钥和挑战
openssl_spki_verify – 验证签名公钥和挑战。
openssl_verify – 验证签名
openssl_x509_check_private_key – 检查私钥是否对应于证书
```

```
openssl_x509_check_private_key — 检查密钥是否对应于证书
openssl_x509_checkpurpose — 验证是否可以为特定目的使用证书
openssl_x509_export_to_file — 导出证书至文件
openssl_x509_export — 以字符串格式导出证书
openssl_x509_fingerprint — 计算一个给定的x.509证书的指纹或摘要
openssl_x509_free — 释放证书资源
openssl_x509_parse — 解析一个X509证书并作为一个数组返回信息
openssl_x509_read — 解析一个x.509证书并返回一个资源标识符
openssl_x509_verify — Verifies digital signature of x509 certificate against a public key
```

以默认算法对openssl库函数进行分类,如下:
默认加密算法: `OPENSSL_CIPHER_RC2_40` 的函数

- openssl_cms_encrypt
- openssl_pkcs7_encrypt

默认编码方法: `OPENSSL_ENCODING_SMIME` 的函数

- openssl_cms_verify
- openssl_cms_sign

默认加解密方法: `RC4` 的函数

- openssl_open
- openssl_seal

默认签名算法: `SHA1` 的函数

- openssl_pbkdf2
- openssl_sign
- openssl_verify
- openssl_x509_fingerprint

默认签名Flag: `PKCS7_DETACHED` 的函数

- openssl_pkcs7_sign

默认Padding: `OPENSSL_PKCS1_PADDING` 的函数

- openssl_private_decrypt
- openssl_private_encrypt
- openssl_public_decrypt
- openssl_public_encrypt

5.2 python

5.2.1 standards library

5.2.1.1 hashlib

hashlib底层代码为openssl开源库，openssl支持的方法，hashlib都能支持。
所有支持的方法有:

```
DSA, DSA-SHA, MD4, MD5, RIPEMD160, SHA, SHA1, SHA224, SHA256,
SHA384, SHA512, blake2b, blake2s, dsaEncryption, dsaWithSHA,
ecdsa-with-SHA1, md4, md5, ripemd160, sha, sha1, sha224, sha256,
sha384, sha3_224, sha3_256, sha3_384, sha3_512, sha512,
shake_128, shake_256, whirlpool
```

其中，无论执行运行环境是什么操作系统，都必然可用的方法有:

```
blake2b, blake2s, md5, sha1, sha224, sha256, sha384, sha3_224, sha3_256, sha3_384, sha3_512, sha512, shake_128,
shake_256
```

> 不同环境的 `algorithms_guaranteed` 不同
> 这里的环境为 `Python3.7.7 MacOS 10.14.6`

hashlib使用方法如下:

```
import hashlib
h = hashlib.md5()
h.update("hello".encode('utf-8'))
print(h.hexdigest())

h = hashlib.sha1()
h.update("hello".encode('utf-8'))
print(h.hexdigest())
```

hashlib用方法名初始化,代码如下:

```
hash_name = "sha1"
h = hashlib.new(hash_name)
h.update(args.data.encode('utf-8'))
print(h.hexdigest())
```

这种不明说用了什么hash的写法更方便出题。

5.2.1.2 hmac

规范hmac的RFC标准
hmac是一个检查通信过程中消息完整性的标准化实现

使用方法如下

```
import hmac
import hashlib

h = hmac.new(b'secret-shared-key', 'hello', hashlib.sha1)
digest = h.hexdigest()
print(digest)
```

hmac.new函数的参数有三个：秘钥、消息、签名算法
其中，签名算法默认为 `md5`

5.2.2 第三方库

5.2.2.1 cryptography

Cryptography：一个提供了加密算法和原语的 python 包。

5.2.2.2 VoidSpace

VoidSpace 是 适用于IronPython的Hashlib装饰器

5.3 go

5.3.1 standards library

5.3.1.1 crypto

所有可用的Hash方法如下

```
const (
    MD4         Hash = 1 + iota // import golang.org/x/crypto/md4
    MD5                         // import crypto/md5
    SHA1                        // import crypto/sha1
    SHA224                      // import crypto/sha256
    SHA256                      // import crypto/sha256
    SHA384                      // import crypto/sha512
    SHA512                      // import crypto/sha512
    MD5SHA1                     // no implementation; MD5+SHA1 used for TLS RSA
    RIPEMD160                   // import golang.org/x/crypto/ripemd160
    SHA3_224                    // import golang.org/x/crypto/sha3
    SHA3_256                    // import golang.org/x/crypto/sha3
    SHA3_384                    // import golang.org/x/crypto/sha3
    SHA3_512                    // import golang.org/x/crypto/sha3
    SHA512_224                  // import crypto/sha512
    SHA512_256                  // import crypto/sha512
    BLAKE2s_256                 // import golang.org/x/crypto/blake2s
    BLAKE2b_256                 // import golang.org/x/crypto/blake2b
    BLAKE2b_384                 // import golang.org/x/crypto/blake2b
    BLAKE2b_512                 // import golang.org/x/crypto/blake2b
)
```

使用方法，以`MD5举例`

```
func GetMd5String(s string) string {
    h := md5.New()
    h.Write([]byte(s))
    return hex.EncodeToString(h.Sum(nil))
}
```

5.3.1.2 crypto包下其他加密包

```
aes,cipher,des,dsa,ecdsa,elliptic,hmac,md5,rand,rc4,rsa,sha1,sha256,sha512,subtle,tls,x509
```

5.3.2 第三方库

Golang似乎没有特别主流的第三方加密库，标准库已经可以解决大部分需求。

6. 密码学知识点

md5

https://en.wikipedia.org/wiki/MD5
https://zhuanlan.zhihu.com/p/37257569

慢哈希函数

https://tate-young.github.io/2019/05/21/bcrypt.html
对了对抗用字典暴破的攻击者，设计出了这种慢哈希函数。
除了所有哈希函数都具备的"给hash，让反着算，算不出来原文"的特性外，同时“有原文，正着算Hash,特别特别慢”
比如`bcrypt`,`scrypt`,`PBKDF2`等
python的bcrypt写法如下:

```
import bcrypt

passwd = b's$cret12'

salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(passwd, salt)

print(salt)
print(hashed)
```

ARGON2

https://github.com/P-H-C/phc-winner-argon2/raw/master/argon2-specs.pdf
Towards Practical Attacks on Argon2i and Balloon Hashing
也是一种慢哈希函数。
`Memory-hard hash function`：通过提高算法中内存的使用量，来提高算法的Hash的耗时。

RC2_40

http://cryptowiki.net/index.php?title=RC2/40_english
https://github.com/SaschaWessel/rc2-40-cbc

SMIME

https://www.cse.scu.edu/~tschwarz/coen350_03/Lectures/smime.html
https://zh.wikipedia.org/zh-hans/S/MIME
安全的电子邮件加密方式
如何使用SMIME

S盒

https://zh.wikipedia.org/wiki/S%E7%9B%92
通常，S-Box接受特定数量的输入比特m，并将其转换为特定数量的输出比特n，其中n 不一定等于m[1]。一个m×n的S盒可以通过包含2m条目，每条目n比特的查找表实现。
S盒通常是固定的（例如DES和AES加密算法），也有一些加密算法的S盒是基于密钥动态生成的（例如Blowfish和双鱼算法加密算法）。

RC4

https://zh.wikipedia.org/zh-hans/RC4

SHA1

https://zh.wikipedia.org/wiki/SHA-1

PKCS7_DETACHED

https://segmentfault.com/a/1190000019793040

OPENSSL_PKCS1_PADDING

https://blog.csdn.net/liuxianbing119/article/details/7405628

ecdsa

https://zhuanlan.zhihu.com/p/97953640

subtle

> subtle不仅是golang加密包的名字，还是一个男士挎包品牌。
> 看上去是那种明明很奢华，但在我身上就被穿成地摊货的款式…
> 诶，又走神了

https://documentation.help/Golang/crypto_subtle.htm

7. Bucket 配置错误

jan.masarik先生19年至今的动态主要在云存储漏洞的自动挖掘上。
- bucketsperm
- Automating Bug Bounty

先来学习一下Bucket错误配置漏洞是咋回事。
由于配置错误的Amazon S3存储桶导致成千上万的患者数据遭到破坏
使用OSS时须先创建Bucket存储空间。
阿里云如何创建Bucket
在jan.masarik先生的论文中，总结了用户在创建buckets时经常犯的错误：

- **不配置权限限制ACL，所有内容public**导致敏感信息泄露
- 同时很多人会像命名变量那样，**以bucket的用途来命名bucket**。
  使得很多存储敏感信息的Bucket可以通过暴破子域名的方式被访问到。
  jan.masarik先生将这些可能存储敏感信息的bucket name汇总成了 下表中的Group2
  同时给出了，经常被使用 但一般不存储敏感信息的Bucket name Group 1

(a) Group 1

bucket name	occurrences
images	662
media	295
public	214
videos	168
content	149

(b) Group 2

bucket name	occurrences
assets	210
files	174
documents	97
uploads	93
docs	81

static	104	backup	43
video	94	data	41
test	93	user-assets	39
temp	91	backups	30
logos	79	logs	29

https://blog.csdn.net/cemao4548

在论文的最后，jan.masarik建议这些OSS 提供商设置bucket name黑名单。
查阅文档，阿里云的OSS就没有这个限制。

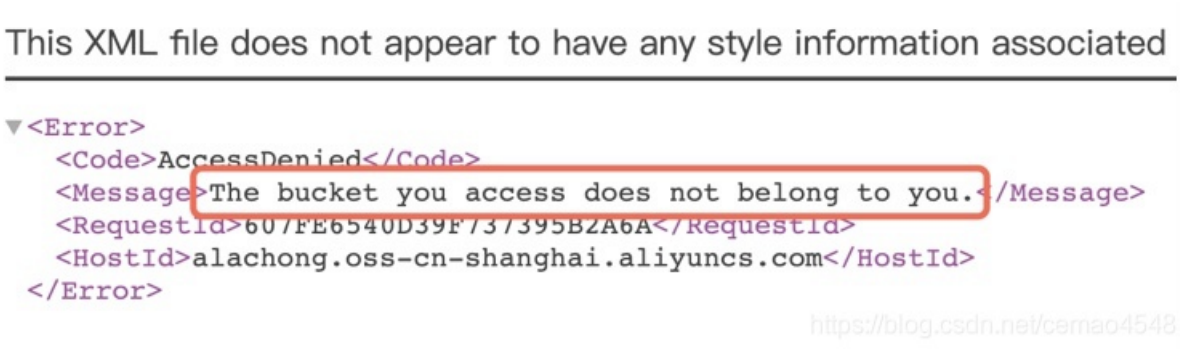
3. 在创建Bucket面板，按如下说明配置必要参数。其他参数均可保持默认配置，也可以在Bucket创建完成后单独配置。

参数	描述
Bucket名称	Bucket的名称。Bucket一旦创建，则无法更改其名称。 命名规则如下： ○ Bucket名称必须全局唯一。 ○ 只能包括小写字母、数字和短划线（-）。 ○ 必须以小写字母或者数字开头和结尾。 ○ 长度必须在3~63字节之间。

https://blog.csdn.net/cemao4548

我在google和github上面随便找了几个oss的教程，
访问了一下教程中的oss链接。

- 当bucket存在，path存在，但ACL private时，报如下错误。



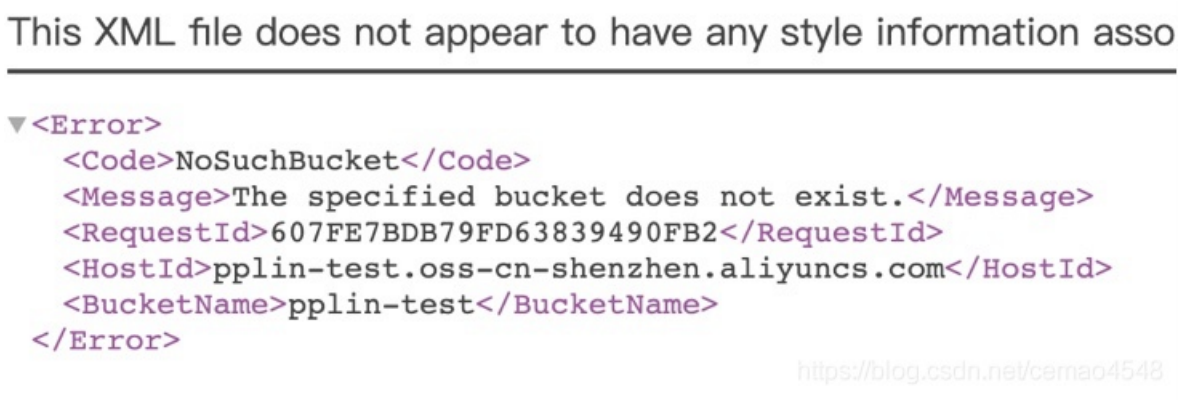
https://blog.csdn.net/cemao4548

- 当bucket存在，但是path不存在时，报如下错误

https://blog.csdn.net/cemao4548

- 当bucket不存在时，报如下错误

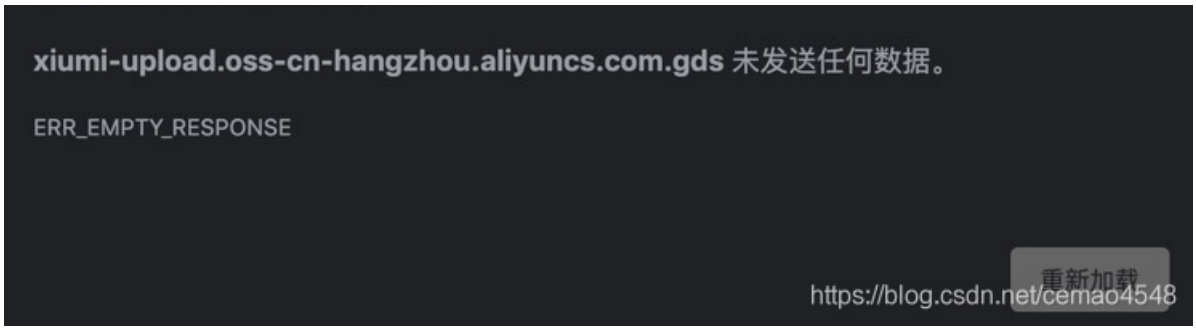
- 当bucket不存在时，报如下错误



- 当bucket存在，path存在，同时还有访问权限时，就可以访问对应的文件或文件夹。

暂时还没有找到ACL错误配置的oss bucket
通过子域名解析的方式搜集了一些oss的链接，确实有Group2中的关键字。看上去的确诱人。

然而访问结果欠佳，应该是bucket对应的子域名注销了oss服务。



暂时没有遇到jan先生论文里说的oss配置错误漏洞实例。

8. 《Non-interactive cryptographic timestamping based on verifiable delay functions》

我们平常在web应用中的时间戳校验，都是依赖于某个权威服务器。
为了实现去中心化，Marc Stevens提出了这种基于VDF的去中心化时间戳方案。

9. sha1collisiondetection

TODO

10. 《On immutability of blockchains》

TODO

11. collect Crypto 2021 Paper

TODO

12. A Hacker's guide to reducing side-channel attack surfaces using deep-learning

这个"基于深度学习来防御旁路攻击"的思路宣讲于`Defcon 28`&`Black Hat USA 2020` // TODO

13. 基于旁路攻击的AES算法中间变量脆弱点

密码学旁路攻击: 基于旁路攻击的AES算法中间变量脆弱点 // TODO

总结

EasyPisy可能的出题人有四位，每个人可能的出题方向如下

- janmasarik: **buckets**未授权访问漏洞(WEB)
- nneonneo (Robert Xiao): **ios**逆向(RE)
- Marc Stevens: 基于**VDF**的去中心化时间戳(BLOCKCHAIN)
- Elie Bursztein (Google): **AES**旁路攻击(CRYPTO)