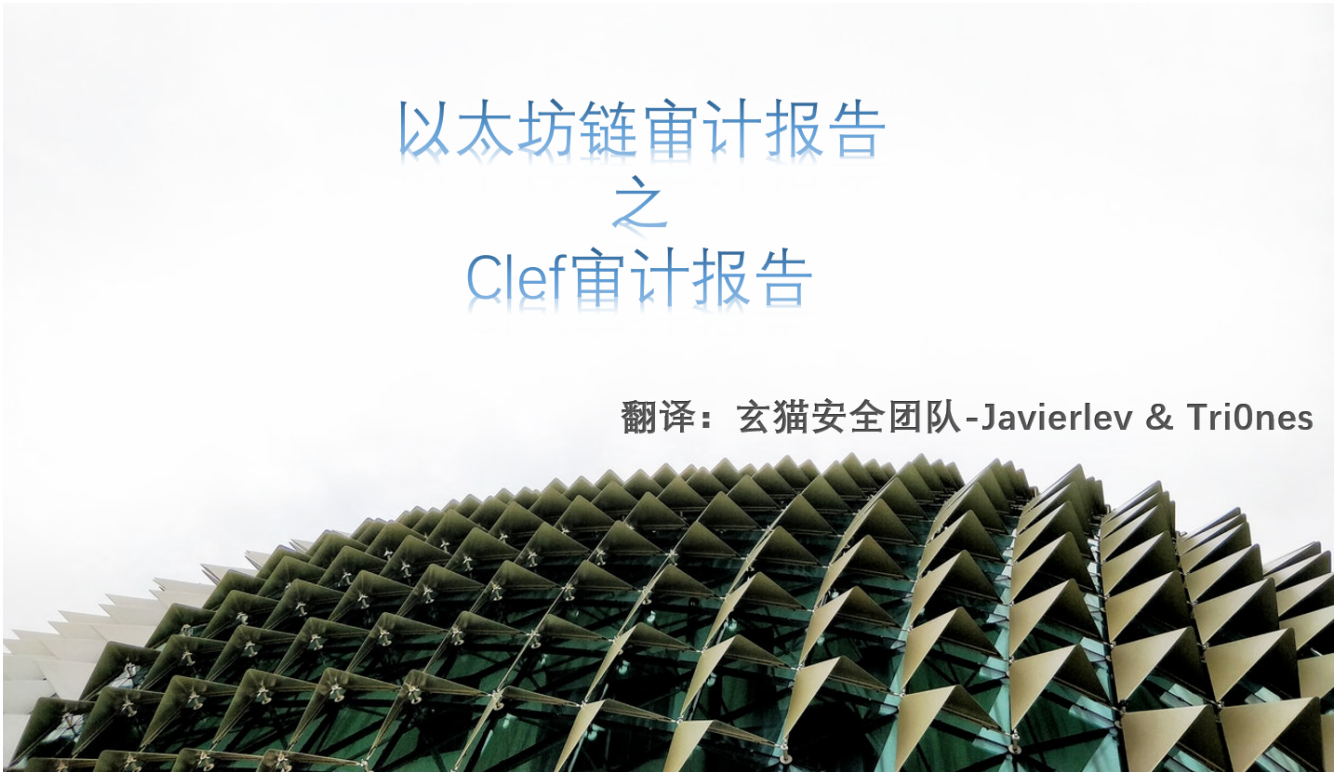
[翻译]以太坊链审计报告之Clef审计报告



翻译：玄猫安全团队-Javierlev & Tri0nes

原文：`https://github.com/ethereum/go-ethereum/blob/master/docs/audits/2018-09-14_Clef-audit_NCC.pdf`

近期，以太坊`go-ethereum`公开了两份审计报告，玄猫安全团队第一时间对其进行了翻译工作。此为第二篇《Ethereum Clef Review》即`2018-09-14_Clef-audit_NCC`，此审计报告完成时间为2018年9月14日。

如果您正在使用的是较旧版本的`go-ethereum`，强烈建议升级至最新版本，以免遭受不必要的损失。

1.1. 概述

在2018年9月初，以太坊基金会委托`NCC Group`对Clef命令行界面进行代码审计。代码位于`https://github.com/holiman/go-ethereum/tree/70cfedc9d7bd64f1f112eb2099a5c36266863f40/cmd/clef`。以下为审计详情。

标题	ID	风险
Clef备份加密不完全	002	中危
缺少密码强度检查	005	中危
交易数据字段验证失败	007	中危
由不正确的方法Selector导致的拒绝服务	010	中危
secrets.dat权限不当	001	低危
键值对加密存储的加密值可交换	003	低危
暴露的Clef API 缺少指引	004	低危
ECRecover不会对恢复的公钥进行身份验证	009	低危
过时的依赖关系	011	低危
规则依赖于时间和状态存在危险性	012	低危
通过格式错误的导入密钥拒绝服务	013	低危
加密密钥库完整性检查不完整	014	低危
UI混合了无关和特定允许的数据	006	建议

1.2. Clef备份加密不完全

1.2.1. 风险: 中危

影响性：高，可利用性： 低

1.2.2. 标识

NCC-EF-Clef-002

1.2.3. 分类

认证

1.2.4. 位置

- cmd/clef/main.go:initializeSecrets()

1.2.5. 影响

一个能够入侵用户设备的攻击者可以任意访问Clef的加密备份

1.2.6. 描述

`Clef`在硬盘中存储了大量文件，如包含账户密码的`credentials.json`，包含`JavaScript`规则文件哈希的`config.json`等文件。通过这些文件可以恢复私钥。这些文件均由一个随机数种子派生的密钥来加密，且这个随机数种子是明文保存在硬盘里的`secrets.dat`文件中。

一个能够访问`secrets.dat`和`credentials.json`的攻击者可以任意访问账户及其资产。因此，一个没有硬盘加密保护的设备如果被物理入侵，将会泄露所有存储在`Clef`加密备份中的信息。当运行中的设备被远程控制时，将会使威胁到休眠的Clef应用，无论此时是否进行了硬盘加密。此外，将这些文件备份或拷贝到另一个位置也会使危及用户的账号安全。

1.2.7. 修复建议

强制使用密码来启动Clef命令行接口。并将此机制用在`go-ethereum/accounts/keystore`来保护`secret.dat`文件。

1.3. 缺少密码强度检查

1.3.1. 风险: 中危

影响性：中，可利用性：低

1.3.2. 标识

NCC-EF-Clef-005

1.3.3. 分类

认证

1.3.4. 位置

- signer/core/api.go:New()

1.3.5. 影响

一个攻击者能够猜解不安全的用户密码，或者对用户弱密码进行爆破。这会导致攻击者获取用户以太坊账户的私钥。

1.3.6. 描述

`Clef CLI`能够被用于创建和管理以太坊外部拥有的账号。账户的创建过程可以通过发送下面的RPC请求给Clef来实现。(如果运行时加上--rpc选项)

```
curl -H "Content-Type: application/json" -X POST --data '{"jsonrpc":"2.0", \
"method":"account_new","params":["test"],"id":67}' localhost:8550
```

运行`Clef`的用户将会被要求输入用于保护账户私钥的密码。在这时，用户能够输入一个任意长度的密码(空密码也将会被接受)。对于可以物理入侵或者远程入侵的攻击者来说，他们将更容易地恢复用户私钥。

1.3.7. 修复建议

强制规定最小密码长度。`NIST`组织已经出版了相关主题的文档，推荐设置最小8个字符长度的密码。除此之外，检查已知的不符合规范的密码。关于不符合规范的密码参考NIST Bad passwords

1.3.8. 客户反馈

Clef现在已经强制使用最小10位的密码： `github.com/holiman/go-ethereum/commit/193f7049719a2da9018027853d0c2237cdad602b`

1.3.9. 参考资料

`https://pages.nist.gov/800-63-3/`

NIST Bad Passwords: `https://cry.github.io/nbp/`

1.4. 交易数据字段验证不当

1.4.1. 风险: 中危

影响性：中，可利用性：低

1.4.2. 标识

NCC-EF-Clef-007

1.4.3. 分类

数据验证

1.4.4. 位置

- signer/core/{abihelper,validation}.go

1.4.5. 影响

恶意构造的数据字段能够允许攻击者欺骗签名者的目的

1.4.6. 描述

当接收交易签名的请求时，在将请求传递给用户之前，`Clef`会先进行一些验证检查。如果一个特殊的参数被传递给一个请求(方法的签名)，程序会试图匹配数据字段。数据字段必须由4个字节的标志和32倍数的字节组成，且该4个字节的标志用于函数调用(方法签名的哈希值会被截断成4个字节):

```
func parseCallData(calldata []byte, abidata string) (*decodedCallData, error) {
    if len(calldata) < 4 {
        return nil, fmt.Errorf("Invalid ABI-data, incomplete method signature of (%d
bytes)", \
        len(calldata))
    }
    sigdata, argdata := calldata[:4], calldata[4:] // 进行截断
    if len(argdata)%32 != 0 { // 32的倍数
        return nil, fmt.Errorf("Not ABI-encoded data; length should be a multiple of 32
(was %d)", \
        len(argdata))
    }
```

如果在请求中，没有方法签名作为参数被传递给Clef，这里的检查就不会进行。这是因为方法签名不是以太坊虚拟机的特性，它是`Solidity-specfic`特性。如果检查失败，`Clef`会中止传递请求，并且发出警告:

```
info, err = testSelector(*methodSelector, data)
if err != nil {
    msgs.warn(fmt.Sprintf("Tx contains data, but provided ABI signature
    could not be matched: %v", err))
}
```

由于`Clef CLI`的用户不会总是传递一个方法签名，或者不能理解关于`ABI`签名检查相关的警告，`Clef`的行为可能使用户快速关闭警告(这被称为警告疲劳)，由于这个原因，恶意的DAPP可以通过短地址攻击或者构造畸形的calldata数据来实施攻击。

1.4.7. 修复建议

当一个方法签名被传递，数据字段格式应该总是指定长度为`4+k×32`，其中`k≥0`。如果不是方法签名被传递，`Clef`不应该把请求传递给用户。当一个方法签名不被传递且数据字段不为空时，它的格式也应该根据之前讨论的编码来进行检查。如果验证失败，`Clef`应该拒绝该交易(用于简化验证过程或列入白名单的合约地址不在此列)。如果非标准的交易需要默认支持，用户应该收到警告，提示该交易不是标准格式。

1.4.8. 客户反馈

现在验证过程中，当拒绝交易时，默认返回警告信息，且增加一个特殊的模式用来绕过这个行为:
`github.com/holiman/go-ethereum/commit/193f7049719a2da9018027853d0c2237cdad602b`

1.4.9. 参考资料

`https://www.dasp.co/#item-9`

1.5. 由不正确的方法Selector导致的拒绝服务

1.5.1. 风险: 中危

影响性：低，可利用性：低

1.5.2. 标识

NCC-EF-Clef-010

1.5.3. 分类

数据验证

1.5.4. 位置

- signer/core/abihelper.go:parseCallData()
- accounts/abi/abi.go:JSON()

1.5.5. 影响

一个可以访问`Clef API`的攻击者能够使应用崩溃

1.5.6. 描述

在一些用例中，Clef持续地运行，并且根据用户编写的规则来接收请求。在一些场景中，应用崩溃会使得在应用重启之前，合法的交易不被处理。

`account_signTransaction`API处理签名请求的交易。为了提供有用的信息给用户，发起请求的终端能够提供被调用的函数方法的签名(防止交易导致合约执行)。如果方法签名是畸形的，Clef将会崩溃。下面这个正则匹配用于验证用户的输入:

```
// MethodSelectorToAbi converts a method selector into
// an ABI struct. The returned data is a valid json string
// which can be consumed by the standard abi package.
func MethodSelectorToAbi(selector string) ([]byte, error) {
    re := regexp.MustCompile(`^( [^\)] +) \( ([a-z0-9,\[\]] *)\)`)
    groups := re.FindStringSubmatch(selector)
```

该正则表达式期望方法签名为如下格式:

```
functionName(uint256, string, address)
```

使用黑名单而不是白名单过于自由。这样的过度接受策略将会使得用户可以输入如下内容:

- 函数名可以是除`\`和`)`的任意字符
- 参数可以是字母数字的或包含`[`与`]`的字符串，但是不需要强制是语法上的正确方括号
- 参数列表能够以`,`结束和开始
- 函数签名的结尾可以包含任何字符

这意味着根据当前的检查规则，下面的函数签名是有效的:

```
call(a,a],bbbb932[,)
#@#((@$!(uint256) anything
```

1.5.7. 复现步骤

在终端运行下面的命令，调用本地RPC接口`localhost:8550`，并且能够观察到`Clef`程序崩溃。

```
curl -i -H "Content-Type: application/json" -X POST --data '{"jsonrpc":"2.0", \
"method":"account_signTransaction","params":[ \
{"from":"0x82A2A876D39022B3019932D30Cd9c97ad5616813", \
"gas":"0x333","gasPrice":"0x123","nonce":"0x0", \
"to":"0x07a565b7ed7d7a678680a4c162885bedbb695fe0", \
"value":"0x10",
"data":"0x4401a6e40000000000000000000000000000000000000000000000000000000000000012"},
"func(uint256,uint256,[]uint256)"],"id":67}' http://localhost:8550/
```

以下的方法签名均会使应用崩溃:

```
func(uint256,uint256,[]uint256)
func(uint256,uint256,uint256,)
func(,uint256,uint256,uint256)
```

1.5.8. 修复建议

为了解决这个问题:

1. 研究`abi`的`JSON`解码器并且找到错误的根源。
2. 在进行操作之前，进一步验证接收到的方法签名

1.5.9. 客户反馈

引入一个PR来修复这个bug: `github.com/ethereum/go-ethereum/pull/17653`

1.6. secrets.dat权限不当

1.6.1. 风险: 低危

影响性：高，可利用性：低

1.6.2. 标识

NCC-EF-Clef-001

1.6.3. 分类

配置不当

1.6.4. 位置

- 权限设置initializeSecrets()，`cmd/clef/main.go`第228行
- 权限检查`checkFile()`,`cmd/clef/main.go`第550行

1.6.5. 影响

`master`种子可能被删除或者覆盖，导致不能访问账户密码和`JavaScript`的规则文件哈希

1.6.6. 描述

`secrets.dat`文件包含了`master`种子，在存储和查询账户密码和`JavaScript`规则文件哈希值的时候需要用到这个种子。在实际中，这个文件只写一次，包含了重要的根密码，需要得到最大限度的保护。

`Master`种子生成和存储是`initializeSecrets()`的主要目的，该函数位于`cmd/clef/main.go`中。在代码228行，当权限设置为700时，可以进行写入硬盘的操作。权限700对应着拥有者的所有权限——读，写，执行。这样做的结果是，拥有者能够轻易地删除或者覆盖这个文件，从而导致不能访问存储的内容。原则上，这个拥有者还能够进行执行操作。

位于`cmd/clef/main.go`的`checkFile()`函数的主要目的是检查文件`secrets.dat`的权限。在代码550行，文件权限是读。将权限模式与077进行与运算，并且检查结果是否为0——任何非0结果将会报错。这与上面描述的`initializeSecrets()`确认存储权限设置是一致的。

对于`secrets.dat`，写权限和执行权限不应该被设置。对于`secrets.dat`文件的处理可以类比处理`SSH keys`的方法。

分开来看，账户密码存储在`credentials.json`中，`JavaScript`规则文件哈希存储在`config.json`中。通过位于`signer/storageaes_gcm_storage.go`的`writeEncryptedStorage`函数，将它们的文件权限设置为600。这被认为合适的，这是因为读/写的本质，键值对的存储，实际上内容应该总是由`root secret`来加密。

1.6.7. 修复建议

在`initializeSecrets()`中，`secrets.dat`的文件权限应该被设置为400(而不是700)。为了保持一致性，在`checkFile()`中，`secrets.dat`的文件权限应该是和377(而不是077)进行与运算。

1.6.8. 参考资料

`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#troublesho ot-unprotected-key`

1.7. 键值对加密存储的加密值可交换

1.7.1. 风险: 低危

影响性：低，可利用性：低

1.7.2. 标识

NCC-EF-Clef-003

1.7.3. 分类

加密

1.7.4. 位置

- signer/storage/aes_gcm_storage.go

1.7.5. 影响

一个能够加密备份文件的攻击者能够交换用户的`keystore`的密文。这将会导致弱攻击(比如确认不同的私钥是否被同一个密码保护)，或者其他未知的依赖于Clef规则的复杂攻击

1.7.6. 描述

Clef命令行接口以加密的形式存储这下面的数据，目的是为了加快应用重启后的恢复速度:

- `keystores`的密码(用于规则引擎)
- `javascript`规则的存储
- 规则文件的哈希

存储和加密是通过一个键值对存储的，只有值是通过`AES-GCM`来进行加密:

```
// Put stores a value by key. 0-length keys results in no-op
func (s *AESEncryptedStorage) Put(key, value string) {
    // ...
    ciphertext, iv, err := encrypt(s.key, []byte(value))
    // ...
    encrypted := storedCredential{Iv: iv, CipherText: ciphertext}
    data[key] = encrypted
    // ...
}
```

键值对接下来被编码成`JSON`格式，并且保存在硬盘中，示例如下:

```
{
    "key1": {
        "iv": "IQZYrnHOYjbcLmBD",
        "c": "oP2S7Li+YYPt2vQcfDgUlc/QaIk="
    },
    "key2": {
        "iv": "OVilp+zm+OvgH7Vm",
        "c": "DP7kmTyJR89nTMb1mfRPokIYRpg="
    }
}
```

一个拥有恰当权限访问这些文件的攻击者能够篡改内容，交换`key1`和`key2`的内容，这样会使得读取`key1`值的时候，获取到`key2`的值

1.7.7. 修复建议

导入在`Seal()`和`Open()`函数的`additionalData`字段，主要部分键值对。详情查看ciper package

1.7.8. 客户反馈

代码中添加了键值对的主要部分作为`AES-GCM`额外的数据:`https://github.com/holiman/go-ethereum/commit/913f77ca8c5c08749b9d668adeb1ab02bbc30663`

1.7.9. 参考资料

`https://golang.org/pkg/crypto/cipher/`

1.8. 暴露的 Clef API 缺乏指引

1.8.1. 风险：低危

影响性：低，可利用性：不确定

1.8.2. 标识

NCC-EF-Clef-004

1.8.3. 分类

拒绝服务

1.8.4. 位置

- cmd/clef/main.go

1.8.5. 影响

有权访问 `Clef API` 的攻击者不断发送大量垃圾请求到接口上使其失效，导致用户强制重新启动应用程序才能处理合法请求。

1.8.6. 描述

有权访问 Clef 的公共 API 的攻击者（例如通过暴露在互联网的RPC接口）可以通过快速发送垃圾请求，导致产生大量需要用户按顺序手动处理进程。

如果执行此类攻击，最终用户将无法继续正常操作，除非重启程序。

此外，通过以太坊的RPC协议完成的请求未加密。虽然大多数API请求和响应最终可能都会发布在以太坊网络上，但“account_sign”方法（旨在为不同目的签署任意数据）还是应该被保密。

1.8.7. 复现步骤

使用 Clef 程序运行下面的`bash`脚本，将 RPC 接口暴露在 `localhost:8550`，然后可以观察到Clef 用户需要一个一个地接收请求。

```
for i in {1..100}
do curl --no-buffer -H "Content-Type: application/json" -X POST --data \
'{"jsonrpc":"2.0","method":"account_new","params":["test"],"id":67}' localhost:8550 &
done
kill $(jobs -p)
```

1.8.8. 修复建议

对连接进行加密（通过TLS）并向Clef API验证客户端。或者，将这些任务委派给较低层协议或前端代理，或者添加文档用于警告用户不要将Clef的API暴露。

1.9. ECRecover不会对恢复的公钥进行身份验证

1.9.1. 风险：低危

影响性：不确定，可利用性：不确定

1.9.2. 标识

NCC-EF-Clef-009

1.9.3. 分类

加密

1.9.4. 位置

- signer/core/api.go

1.9.5. 影响

根据此请求的使用情况，签名可能会被篡改从而恢复错误的公钥。

1.9.6. 描述

`Clef API`公开了一种`EcRecover`方法，该方法允许从签名消息中恢复以太坊公钥。该方法实现了椭圆曲线密码学的高效密码组标准文档中的算法4.1.6（公钥恢复操作）。

如算法规范所述，可以从签名中恢复几个公钥。

这是由于`ECDSA`签名算法从签名的`r`值中删除了一些信息：只保留了x坐标（对于y坐标可以恢复2个解）并且它以椭圆曲线的顺序进一步减小。 而以太坊使用了`secp256k1`，其曲线顺序略低于场模量，因此确实丢失了信息。曲线使用辅助因子1，则算法的可能解决方案的数量是2×（1 + 1）= 4。

为了使恢复算法恢复正确的结果，在每个以太坊签名的末尾都添加一个v字节。其最低有效位包含r值的y坐标的符号，其余位包含用于重新计算r值的x坐标的信息。

由于这些位可能被篡改，攻击者在某些情况下可能会通过导入错误的公钥来欺骗算法。

1.9.7. 修复建议

为了验证恢复的密钥，Clef需要：

1. 验证密钥是否可用于验证请求中传递的签名。 这是SEC算法的步骤1.6.2，Clef没有实现。
2. 将恢复的公钥与以太坊地址或其他身份验证机制进行匹配。

为了防止这些攻击，需要更改Clef的API以接受额外的“身份验证”参数。

1.9.8. 客户反馈

ECRecover 方法已经从Clef API 移除：github.com/holiman/go-ethereum/commit/cf3bf1724e58cc85ec87cb39a0aee0cb246c472e

SEC 1: Elliptic Curve Cryptography version 2.0

1.10. 过时的依赖关系

1.10.1. 风险：低危

影响性：不确定，可利用性：不确定

1.10.2. 标识

NCC-EF-Clef-011

1.10.3. 分类

修补

1.10.4. 位置

- signer/rules/deps/bignumber.js found at
- `https://github.com/holiman/go-ethereum/blob/clefchanges_2/signer/rules/deps`
- `https://github.com/ethereum/go-ethereum/blob/master/signer/rules/deps`
- vendor/vendor.json found at
- `https://github.com/holiman/go-ethereum/blob/clefchanges_2/vendor`
- `https://github.com/ethereum/go-ethereum/blob/master/vendor`

Impact Outdated dependencies may expose the application to publicly discovered vu

1.10.5. 影响

过时的依赖项可能会将应用程序暴露给公开发现的漏洞。

1.10.6. 描述

迄今为止，许多最大的漏洞都依赖于利用过时组件中的已知漏洞。`Clef`和`Go-ethereum`代码库从许多过时的组件中提取，尽管目前没有众所周知的漏洞。风险与组件功能和数据敏感性，开发活动和质量，流行度以及项目依赖性更新之间的时间长度成正比。由于此问题普遍存在，OWASP项目在《十大最关键Web应用程序安全风险》中列出了此风险。

`signer/rules/rules.go`代码利用`signer/rules/deps/bindata.go`实质上加载`bignumber.js`库，用于任意精度的十进制和非十进制算术。 该库的源代码是`signer/rules/deps/bignumber.js`，版本为2.0.3。 此项目的更改日志表示此版本于2015年2月发布，而最新版本为7.2.1。NCC Group 不知道此库中存在任何公开已知的漏洞。

`vendor/vendor.json`文件列出了大约154个Golang依赖项，其修订时间戳从2015年初到2018年8月。其中大部分已过期且可以更新。 例如，来自`https://github.com/huin/goupnp`的Go存储库的UPnP客户端库的七个组件，其提交哈希值为`679507af18f3c7ba2bcc7905392ce23e148661c3`，于2016年12月提交，即11个提交已过期。

1.10.7. 修复建议

将项目依赖项更新为建议用于生产部署的最新且稳定的版本。作为开发过程的一部分，包括定期审查依赖性更新情况。

1.10.8. 参考资料

`https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf`

`https://github.com/MikeMcl/bignumber.js/blob/master/CHANGELOG.md`

1.11. 规则依赖于时间和状态存在危险性

1.11.1. 风险：低危

影响性：中，可利用性：低

1.11.2. 标识

NCC-EF-Clef-012

1.11.3. 分类

配置

1.11.4. 位置

- cmd/clef/rules.md

1.11.5. 影响

存在改变`Clef`状态和时间访问的攻击。如果成功安装，它们将允许攻击者恢复`Clef`规则使用的状态或改变`Clef`所看到和使用的时间。这最终可能允许有权访问`Clef`接口的攻击者移除规则上的某些限制。

```
var windowstart = new Date().getTime() - window;
```

攻击者存在不同的方式来影响设备运行时间而不是系统上的root用户：

1. 如果在日期程序中设置了`CAP_SYS_TIME`功能，则任何用户都可以更改时间。
2. 如果攻击者在网络中具有特权中间位置，她可以攻击NTP协议以改变设备的时间。

此外，为了在执行规则之间保持状态，Clef保留加密的键值存储（jsStorage）。特定攻击可能允许攻击者改变此状态并删除一些限制（例如，如果布尔值设置为阻止进一步的交易事务，则恢复状态将允许交易事务再次流动）：

1. 如果攻击者具有对计算机的物理访问权限，则可以将其重新设置为以前的快照。
2. 如果攻击者拥有`jsStorage`的文件权限，她可以记录更改并将文件还原到以前的某个时间点。

这些攻击可能允许攻击者阻止某些规则正常工作，或者更糟糕的是，攻击者直接访问Clef的API，让钱包中的金钱流失。

1.11.6. 修复建议

这一发现强调了运行Clef的系统的安全性至关重要。应根据涉及的程度，为不同类型的用户提供不同的建议。可以编写不同的威胁模型，记录Clef防范和不防范的内容。最终，很难防御来自强大对手的这些类型的高度针对性的攻击，它们应该超出Clef的威胁模型。

1.11.7. 参考资料

cmd/clef/rules.md

`http://man7.org/linux/man-pages/man7/capabilities.7.html`

`https://www.cs.bu.edu/~goldbe/NTPattack.html`

1.12. 通过格式错误的导入密钥拒绝服务

1.12.1. 风险：低危

影响性：中，可利用性：低

1.12.2. 标识

NCC-EF-Clef-013

1.12.3. 分类

数据验证

1.12.4. 位置

- accounts/keystore/keystore_passphrase.go

1.12.5. 影响

一个可以访问API或拥有特权中间人位置的攻击者，可能会制造恶意导入请求或篡改它们，以便在不提醒用户的情况下使应用程序崩溃或更改导入的密钥。

1.12.6. 描述

`Clef` 的API公开了一种“导入”方法，允许请求导入已有的帐户。此导入方法接受一个加密密钥作为参数，必须以特定格式格式化。`Clef` 接受两种不同的加密方法格式：版本1和版本3。处理密钥导入的大多数代码都假定请求传递的参数是可信的，这可能是因为在实践中，无法编写规则来处理此方法——用户必须手动接受导入请求。以下代码路径都存在问题：

1）导入私钥。

导入流程最终调用`crypto.ToECDSAUnsafe()`方法，如所示，“如果盲目地将二进制blob转换为私钥。它们几乎不会被使用，除非您确定输入的有效性并且避免由于错误的原编码而导致错误（0前缀会被截断）”。

2）JSON解析。

JSON对象中的几个字段，在没有事先检查它们是否存在的情况下进行恢复操作。用于恢复KDF参数的`getKDFKey()`函数不会将空内容映射视为`cryptoJSON.KDFParams`，如果给定一个，则会崩溃。此外，一般以整数作为KDF对象的字段，但即使给定字符串，它也没有进行判断。

3）KDF参数。

可以通过向密钥导出函数提供极大的参数导致拒绝服务，这将迫使程序计算无法进行的加密操作。

4）经过身份验证的加密。

在尝试对导入的密钥进行解密之前，密钥库将验证密文的完整性，以便检测来自中间攻击者的任何篡改。此完整性检查不包括IV，并且不会在恒定时间内完成。这可能允许攻击者篡改IV，使得用户解密错误的私钥，并且没有Clef给出的任何警报（即使恢复的“地址”不同于在请求中作为参数传递的“地址”字段）。

5）解密。

导入器的版本1将使用`AES-CBC`来解密接收到的密钥，特别是低级`CryptBlocks`函数，将多个`blocksize`作为参数。否则，该功能将会出现`panic`报错。

1.12.7. 复现步骤

将 Clef 程序的 RPC 接口暴露在`localhost:8550`，然后在命令行窗口执行下面的命令，可以观察到 Clef 程序奔溃。

```
curl -i -H "Content-Type: application/json" -X POST \
--data '{"jsonrpc":"2.0","method":"account_import", \
"params":[{"version":"1","address":"string", \
"id":"string","crypto":{"cipher":"","ciphertext":"", \
"cipherparams":{"iv":""},"kdf":"", \
"kdfparams":{},"mac":""}}],"id":67}' http://localhost:8550/
```

下面的 payloads 也能够导致程序奔溃。

```
{"version":"1","address":"string","id":"string","crypto": \
{"cipher":"","ciphertext":"","cipherparams":{"iv":""},"kdf":"", \
"kdfparams":{"salt":"","dklen":"","n":"","r":"","p":"","c":"", \
"prf":""},"mac":""}}
{"version":"1","address":"string","id":"string","crypto": \
{"cipher":"","ciphertext":"01","cipherparams":{"iv":""}, \
"kdf":"pbkdf2","kdfparams": \
{"salt":"","dklen":5,"n":5,"r":5,"p":5,"c":5, \
"prf":"hmac-sha256"}, \
"mac":"32f2f344a0bdf0434df8d3c3fd2afd043c1a26b969bb7c448abd67a4af27ae03"}}
```

1.12.8. 修复建议

在文档中详细说明导入API必须只接收可信任的和通过验证的输入。 此外，还要解决这一发现中强调的问题。

此外，请考虑从Clef的API中删除Import方法。

1.13. 加密的密钥库完整性检查不完整

1.13.1. 风险：低危

影响性：低，可利用性：低

1.13.2. 标识

NCC-EF-Clef-014

1.13.3. 分类

密钥存储

1.13.4. 位置

- accounts/keystore/keystore_passphrase.go

1.13.5. 影响

攻击者可以篡改钱包备份，而不会提醒用户，在尝试使用钱包密钥之前，用户不会意识到攻击。

1.13.6. 描述

`Go-Ethereum`的密钥库包具有导出的`EncryptKey()`方法，能够以加密形式存储钱包。此加密使用从用户已知的密码短语派生的密钥。作为完整性检查的一种方法，为了确保加密密钥的备份没有被篡改，密钥库通过密文计算消息认证码（MAC），如下所示：

```
// EncryptKey encrypts a key using the specified scrypt parameters into a json
// blob that can be decrypted later on.
func EncryptKey(key *Key, auth string, scryptN, scryptP int) ([]byte, error) {
    // ...
    derivedKey, err := scrypt.Key(authArray, salt, scryptN, scryptR, scryptP, scryptDKLen)
    // ...
    iv := make([]byte, aes.BlockSize) // 16
    if _, err := io.ReadFull(rand.Reader, iv); err != nil {
        panic("reading from crypto/rand failed: " + err.Error())
    }
    cipherText, err := aesCTRXOR(encryptKey, keyBytes, iv)
    // ...
    mac := crypto.Keccak256(derivedKey[16:32], cipherText)
```

此完整性检查不包括加密/解密过程的重要部分，允许攻击者在不必修改密文的情况下篡改它。由于加密的内容具有高熵，因此不能执行强攻击。

1.13.7. 修复建议

使用经过身份验证的密码，如`AES-GCM`，它将加密与完整性检查编译为单个算法。

1.14. UI混合了无关和特定允许的数据

1.14.1. 风险：建议

影响性：不确定，可利用性：不确定

1.14.2. 标识

NCC-EF-Clef-006

1.14.3. 分类

数据有效性

1.14.4. 位置

- signer/core/cliui.go

1.14.5. 影响

攻击者可以通过在 Clef UI 中显示攻击者控制的信息来对用户进行网络攻击。

1.14.6. 描述

当`Clef`通过其公开的API接收请求时，元数据将显示给负责处理它的用户。此元数据包括的各种字段均与IP地址，用户代理，来源等签名内容无关。在`signer/core/cliui.go`中有6个通过调用`showMetadata()`来调用此功能。其中一些字段可能会被轻微篡改，并可能提供错误的理解，因为用户可能过分依赖它们而不是重要的字段。

Clef将接受以下“恶意”请求（使用Go代码示例注释编造的JSON）。

```
curl http://localhost:8550/ \
    -i -H "Content-Type: application/json" \
    -X POST --data '{...}' \
    -A "indicates INVALID CHECKSUM IS EXPECTED" \
    -H "Origin: NCC Group requires IMMEDIATE APPROVAL per direction of J Smith"
```

Clef 将向用户显示以下信息

```
--------- Transaction request-------------
to: 0x07a565b7ed7d7a678680a4c162885bedbb695fe0

WARNING: Invalid checksum on to-address!

from: 0x82A2A876D39022B3019932D30Cd9c97ad5616813 [chksum ok]
value: 16 wei
gas: 0x333 (819)
gasprice: 291 wei
nonce: 0x0 (0)
data: 0x4401a6e40000000000000000000000000000000000000000000000000000000000000...012

Transaction validation:
  * WARNING : Invalid checksum on to-address
  * Info : safeSend(address: 0x0000000000000000000000000000000000000012)

Request context:
        127.0.0.1:40802 -> HTTP/1.1 -> localhost:8550
        User-Agent: indicates INVALID CHECKSUM IS EXPECTED
        Origin: NCC Group requires IMMEDIATE APPROVAL per direction of J Smith
-------------------------------------------
Approve? [y/N]:
>
```

如目前所示，元数据对合法请求几乎没有好处，却可能促进非法请求。天真的用户可能会将无关的请求数据视为取代上述真实警告并错误地批准此交易。

1.14.7. 复现步骤

如果没有明确的描述和警告，请不要在特定于审批的数据旁边显示请求元数据。 要么明确标记所显示的类别，并警告不能依赖请求数据，要么只删除所有请求数据。

1.14.8. 修复建议

如果没有明确的描述和警告，请不要在特定允许的数据旁边显示请求元数据。要么明确标记所显示的类别，并警告不能依赖请求数据，要么只删除所有请求数据。

1.14.9. 客户反馈

在显示由API的外部调用者提供的元数据之前添加了另一条消息：github.com/holiman/go-ethereum/commit/c6d7644e5a5bd0fe23c7f060a390112115515cab

1.4. 附录

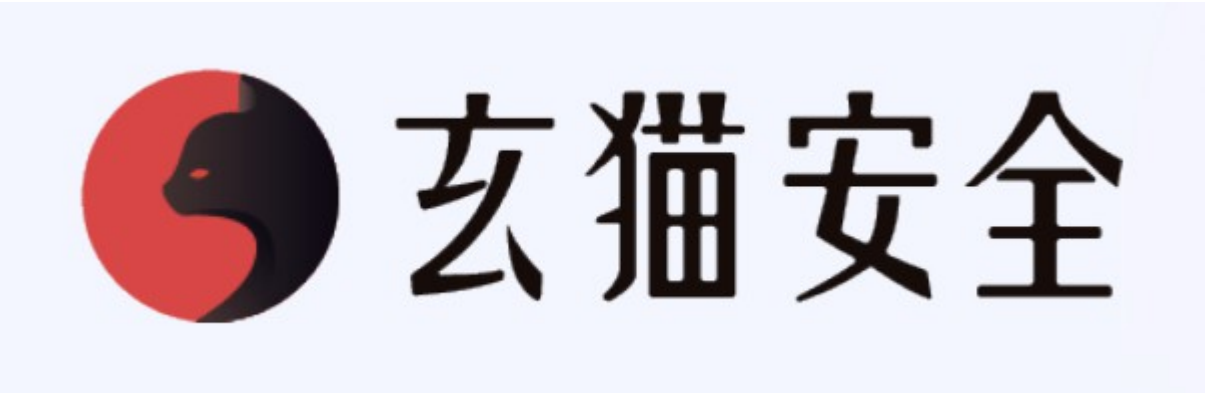
1.4.1. 声明

我们努力提供准确的翻译，可能有些部分不太准确，部分内容不太重要并没有进行翻译，如有需要请参见原文。

1.4.2. 原文地址

https://github.com/ethereum/go-ethereum/blob/master/docs/audits/2018-09-14_Clef-audit_NCC.pdf

关于玄猫安全



玄猫区块链安全实验室专注区块链安全领域，致力于提供区块链行业最专业的安全解决方案，团队成员来自于百度、阿里、360等国际顶尖安全团队，已为数十家交易所、电子钱包、智能合约等提供基础安全建设、渗透测试、漏洞挖掘、应急响应等安全服务。

玄猫安全实验室提供专业权威的智能合约审计服务、区块链专项应用评估、区块链平台安全评估等多项服务。

商务合作：Lyon.chen@xuanmao.org