Competitive Security Assessment

MirrorWorld - MPC Wallet

Feb 14th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	MirrorWorld - MPC Wallet
Platform & Language	Go
Codebase	• https://github.com/mirrorworld-universe/eddsa_protocol • audit commit - a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79 • final commit - bae7a5d81ec1ce21334a1f617042afe18ab8725f • https://github.com/mirrorworld-universe/two-party-ed25519 • audit commit - 2d7d03cec669b8e17002b6afc1c14713d2ac1ee0 • final commit - a71367f12df9629fe5e91a08652400d1dc877cc2
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

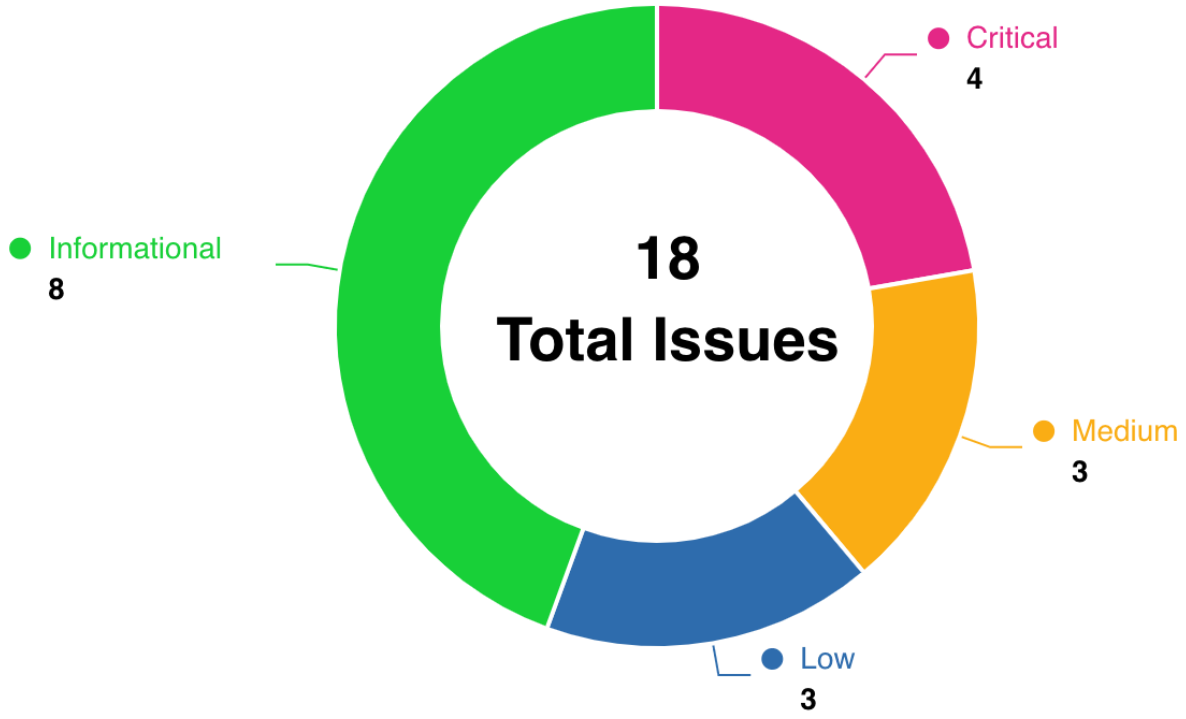
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	4	0	0	3	0	1
Medium	3	0	0	2	0	1
Low	3	0	0	3	0	0
Informational	8	0	2	6	0	0

Audit Scope

File	Commit Hash
./musig2/keygen_test.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./musig2/signature_test.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./musig2/signature.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./musig2/keygen.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/scalar.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/edwards25519.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/point.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/fe.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/const.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./ed25519/tool.go	a635c09f597b29fb2f8e8e3cfc8b4bb96cec9b79
./middleware/validator/validator.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/validator/common_validators.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/logger.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/dao/mysql.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/dao/dao.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/binding/bindjson.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/trace_id.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./middleware/authentication/jwt_authentication.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./config/base.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/err_code/code.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/base_resp/jsonresp.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/solana.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/keygen_test.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/utils.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/sign_test.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/serialization.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0

./internal/tss/sign.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/tss/keygen.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/uuid/uuid.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./internal/logging/logger.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./controller/party.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./controller/mpc.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./finder/wallet_finder.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./model/db/wallet.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./model/rest/p0.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./model/rest/p1.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./model/rest/party.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./model/rest/mpc.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./service/party.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./service/mpc.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./routes/router.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./global/initialize.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./global/global.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./global/constants.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0
./main.go	2d7d03cec669b8e17002b6afc1c14713d2ac1ee0

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
MWW-1	A hard-coded JWT token that is included in the code as a comment	Information Leakage	Medium	Fixed	iczc
MWW-2	Add `.gitignore` to avoid information leakage	Information Leakage	Critical	Fixed	iczc, co2kim
MWW-3	Base64 malleable risk	Logical	Informational	Acknowledged	zircon
MWW-4	CSRF due to CORS middleware with a setting that allows requests from any origin	Logical	Medium	Fixed	iczc

MWW-5	Disclosure of sensitive data information	Logical	Low	Fixed	BradMoonUESTC
MWW-6	Extra comma returned in `ed25519/BytesToStr` function	Code Style	Informational	Fixed	alansh
MWW-7	Insecure Seed Generation for Long Term	Logical	Low	Fixed	lfzkoala
MWW-8	Non-standard Scalar Generation for Partial Nonce.	Logical	Informational	Acknowledged	lfzkoala
MWW-9	The alias "validator2" in main.go of the two-party-ed25519 project is unnecessary	Code Style	Informational	Fixed	iczc
MWW-10	The return value of the function should be checked	Logical	Low	Fixed	zircon
MWW-11	Unauthorized access to user id vulnerability	Logical	Critical	Fixed	zircon
MWW-12	Validate Input Length/Size	Logical	Informational	Fixed	lfzkoala
MWW-13	Vulnerable to Dual Public Key Attack	Logical	Critical	Declined	lfzkoala
MWW-14	Wrong Scalar25519 Decoding Method	Logical	Critical	Fixed	lfzkoala
MWW-15	fix typo	Code Style	Informational	Fixed	comcat
MWW-16	gc could be optimized in `mulsig2/KeyAggregateN` function	Logical	Informational	Fixed	alansh
MWW-17	performance issue in `ed25519/PowerN` function	DOS	Medium	Declined	alansh
MWW-18	unnecessary initialization in `mulsig2/KeyAggregateN` function	Logical	Informational	Fixed	alansh

MWW-1:A hard-coded JWT token that is included in the code as a comment

Category	Severity	Code Reference	Status	Contributor
Information Leakage	Medium	• code/two-party-ed25519/middleware/authentication/jwt_authentication.go#L20	Fixed	iczc

Code

```
20:     //tokenStr := "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTE5LCJldGhfYWRkcmVzcyI6ImJNZ2piNlROOW9UbTRWMmVDZ3A3S3NjNUFEbUwycW02N2RqQXJKZ25IYnIiLCJzb2xfYWRkcmVzcyI6IjRSQVdoNE1ranNqWHRkMTRXaXA1bzRRMW9ndndXVldZOXVndGg4M0FXcHVRIiwiZW1haWwiOm51bGwsIndhbGxldCI6eyJldGhfYWRkcmVzcyI6IjB4Yzk4MTlBYmE3OUY4ZkI0QTY0MDM1NUI4MGEzQzU1YWZhNDQ4MUM4NyIsInNvbF9hZGRyZXNzIjoiYk1namI2VE45b1RtNFYyZUNncDdLc2M1QURtTDJxbTY3ZGpBckpnbkhiciJ9LCJpYXQiOjE2NjA4OTA4MTAsImV4cCI6NDI1Mjg5MDgxMCwianRpIjoiYXV0aDoxMTkifQ.WQmRozDCx8MaD5FaAT7iL17zjD3CRxw_sAd0rRUCCHs"
```

Description

iczc : There is a hard-coded JWT token in the form of a comment, which is intended for testing purposes. This may pose a security risk as it may be obtained by unauthorized persons and used to access protected resources, meaning that an attacker can carry out some operations with the identity of the userid in this JWT.

Recommendation

iczc : It is recommended that the hard-coded JWT token in the comment be removed and use unit tests with a separate JWT secret key to verify functionality.

Client Response

Fixed.

MWW-2:Add `.gitignore` to avoid information leakage

Category	Severity	Code Reference	Status	Contributor
Information Leakage	Critical	• code/two-party-ed25519/.ssh/id_ed25519#L1-L7 • code/two-party-ed25519/.env#L1-L16	Fixed	iczc, co2kim

Code

```
1:ENV=staging
2:
3:P0Url=http://localhost:3000
4:P1Url=http://localhost:3000
5:#P1Url=https://test-sea-staging.mirrorworld.fun
6:DEPLOY_PARTY=both
7:SOLANA_RPC_URL=https://solana-devnet.g.alchemy.com/v2/Kw4imBatcI2gG7DCS_W0omvNSwGGZeWb
8:
9:MYSQL_USER=root
10:MYSQL_HOST=127.0.0.1
11:MYSQL_PASSWORD=123456
12:MYSQL_DB_NAME=mw_dev
13:MYSQL_PORT=3306
14:
15:#jwt
16:AUTH_SECRET=G3qkt0XXVYLZf6X
```

```
1:-----BEGIN OPENSSH PRIVATE KEY-----
2:b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
3:QyNTUxOQAAACBRMHB1CB3SeSCy4T0y5OLmaCDHDU2iOZn09uSppdlTBQAAAJCZLMTqmSzE
4:6gAAAAtzc2gtZWQyNTUxOQAAACBRMHB1CB3SeSCy4T0y5OLmaCDHDU2iOZn09uSppdlTBQ
5:AAAEC01vNs7tkYzs8M1w0kKQadDMtCKFsAMeHQZDUGWWrd2FEwcHUIHdJ5ILLhPTLk4uZo
6:IMcNTaI5mfT25Kml2VMFAAAADGpvemhlQHJjdC5haQE=
7:-----END OPENSSH PRIVATE KEY-----
```

Description

iczc : The SSH private key used to download go private dependencies is kept in the codebase, which makes it possible for anyone to clone the private repository from GitHub with this key.

iczc : The .env file contains sensitive information, such as a JWT secret key, thus the attacker can forge identity by

signing any userid with the JWT secret.
co2kim : The `.ssh/id_ed25519` file containing `OPENSSH PRIVATE KEY` is in the repository. This information should never be exposed outside the project team.

Recommendation

iczc : Remove this key from the codebase and build the docker image in private CI.
iczc : Remove the .env file and add it to .gitignore, and provide a .env.template without the credentials.
co2kim : Remove the `.ssh` file from the repository and regenerate a new SSH private key as the current one in repo is no longer secrete.

Client Response

We removed the .env file and added the .env.template file. As to the ssh key, we didn't delete it since it's only used by our private CI to build docker image. Also, the repo is a private repo.If we decide to open-source it, we will remove the key in future.

MWW-3:Base64 malleable risk

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/two-party-ed25519/internal/tss/serialization.go#L67-L84	Acknowledged	zircon

Code

```
67:     data, err := base64.StdEncoding.DecodeString(txBase64)
68:     if err != nil {
69:             panic(err)
70:     }
71:
72:     tx, err := solana.TransactionFromDecoder(bin.NewBinDecoder(data))
73:     if err != nil {
74:             panic(err)
75:     }
76:     return tx
77:}
78:
79:func Base64ToTxWithNil(txStr string) *solana.Transaction {
80:     tx := TxFromBase64String(txStr)
81:     tx.Signatures = nil
82:     return tx
83:}
```

Description

zircon : Base64 is not a rigorous serialization algorithm and is not suitable for use in blockchain systems.

The impact is that an attacker may be able to construct transactions with malleability, resulting in double spending and loss of assets on transactions.

Consider below POC code:

```
s := "00=="
b, err := base64.StdEncoding.DecodeString(s)
fmt.Println(s, b, err) //00== [211] <nil>

s2 := base64.StdEncoding.EncodeToString(b)
b2, err := base64.StdEncoding.DecodeString(s)
fmt.Println(s2, b2, err) //0w== [211] <nil>
```

Reference: https://eprint.iacr.org/2022/361.pdf

Recommendation

zircon : Don't use base64 to serialize tx or other data, base58 is a better choice.

Client Response

We did not fix it this time because we have other mechanisms to protect the wallet(e.g. JWT token), we will include this in the future release.

MWW-4:CSRF due to CORS middleware with a setting that allows requests from any origin

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/two-party-ed25519/routes/router.go#L19	Fixed	iczc

Code

```
19:             AllowOrigins:     []string{"*"},
```

Description

iczc : The two-party-ed25519 server uses the Gin framework to configure CORS middleware for handling cross-origin resource sharing. The configuration allows requests from any origin by setting AllowOrigins to "*", which could potentially open up the application to a CSRF (Cross-site Request Forgery) vulnerability. This is because it allows any website to access the API by sending requests, allowing an attacker to potentially make unauthorized requests.

Recommendation

iczc : It is recommended to restrict the allowed origins to only trusted websites.

Client Response

Fixed.

MWW-5:Disclosure of sensitive data information

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/two-party-ed25519/global/initialize.go#L16-L20 • code/two-party-ed25519/global/initialize.go#L35	Fixed	BradMoonUESTC

Code

```
16:     log.Println("current env:", config.Base.Env)
17:     log.Println("current deployParty:", config.Base.DEPLOY_PARTY)
18:     if config.Base.DEPLOY_PARTY != DEPLOY_PARTY_P1 {
19:             log.Println("P1 Url:", config.Base.P1Url)
20:     }

35:     log.Println("connecting to DB, settings=", dsn)
```

Description

BradMoonUESTC : The use of `log.println` plaintext logging may result in the disclosure of sensitive information about the database

Recommendation

BradMoonUESTC : Do not print sensitive information in the log file.

Client Response

Fixed.

MWW-6:Extra comma returned in `ed25519/BytesToStr` function

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/eddsa_protocol/ed25519/tool.go#L16-L23 • code/two-party-ed25519/internal/tss/utils.go#L50-L57	Fixed	alansh

Code

```
16:func BytesToStr(b []byte) string {
17:     ia := ByteArrayToInt(b)
18:     s := "["
19:     for _, v := range ia {
20:             s += strconv.Itoa(v) + ", "
21:     }
22:     return s + "]"
23:}

50:func BytesToStr(s []byte) string {
51:     i := BytesToInts(s)
52:     r := "["
53:     for _, v := range i {
54:             r += Int2str(v) + ", "
55:     }
56:     return r + "]"
57:}
```

Description

alansh : `BytesToStr` should omit trailing ","

Recommendation

alansh : Consider below fix in the `BytesToStr` function

```
func BytesToStr(b []byte) string {
        ia := ByteArrayToInt(b)
        s := "["
        for i, v := range ia {
                if i != len(ia)-1 {
                        s += strconv.Itoa(v) + ", "
                } else {
                        s += strconv.Itoa(v)
                }

        }
        return s + "]"
}
```

Client Response

Fixed.

MWW-7:Insecure Seed Generation for Long Term

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/eddsa_protocol/musig2/signature.go#L46	Fixed	lfzkoala

Code

```
46:             seedInt, _ := new(big.Int).SetString("047d196f89599e87258a8ed3041da020724314cce124b5d488ed9632c322acd8", 16)
```

Description

lfzkoala : The `GeneratePartialNonce` function generates the seed by setting a fixed string every time when `seed` is not nil, but for long term it's not secure because we should get a new string for a certain period.

Recommendation

lfzkoala : I'd recommend generate a fresh string every time we call the function when `seed` is not nil or rotate the string after a certain time.

Client Response

Fixed.

MWW-8:Non-standard Scalar Generation for Partial Nonce.

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/eddsa_protocol/musig2/signature.go#L100	Acknowledged	lfzkoala

Code

```
100:    h := sha512.Sum512(fullBytes)
```

Description

lfzkoala : The function `generatePartialNonceInternal` computes `h := sha512.Sum512(fullBytes)` but this method is not standard and not providing full randomness (since only using hash function doesn't make sure the output is random enough).

Recommendation

lfzkoala : Just using sha512 to generate the scalar h is not a standard approach, one of common approaches should be using HKDF to derive a random value and generate the corresponding scalar. The goal here is that we must introduce randomness to the HKDF to make the output non-deterministic because deterministic nonces open up threshold schemes to potential nonce-reuse attacks. We continue to use the HKDF that takes in context about what is going to be signed as it adds some protection against bad local randomness. The sample code is like

```
        randNonce := make([]byte, SeedSize)
        if _, err := io.ReadFull(rand.Reader, randNonce); err != nil {
                return nil, err
        }
        var secret []byte
        secret = append(secret, s.Bytes()...)
        secret = append(secret, randNonce...)
        info := []byte("ted25519nonce")
        info = append(info, p.Bytes()...)
        info = append(info, m...)
        return hkdf.New(sha256.New, secret, nil, info), nil
```

Client Response

We did not fix it this time because the message to hash already contains a random number, the possibility of collision is even lower. We will include this in the future release.

MWW-9:The alias "validator2" in main.go of the two-party-ed25519 project is unnecessary

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/two-party-ed25519/main.go#L8	Fixed	iczc

Code

```
8:      validator2 "github.com/mirrorworld-universe/two-party-ed25519/middleware/validator"
```

Description

iczc : The main.go of the two-party-ed25519 project imports a package named "validator" and assigns it an alias "validator2". This alias is used to avoid naming conflicts with other packages that may have the same name, however, it is not strictly necessary if there are no other packages with the same name being imported in the same scope.

Recommendation

iczc : The alias "validator2" on line 8 can be removed.

Client Response

Fixed.

MWW-10:The return value of the function should be checked

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/two-party-ed25519/controller/party.go#L22-L23 • code/two-party-ed25519/controller/party.go#L23 • code/two-party-ed25519/service/party.go#L40 • code/two-party-ed25519/middleware/authentication/jwt_authentication.go#L46 • code/two-party-ed25519/controller/party.go#L50-L51 • code/two-party-ed25519/internal/tss/solana.go#L61-L66 • code/two-party-ed25519/internal/tss/sign.go#L63 • code/two-party-ed25519/controller/party.go#L87-L90 • code/two-party-ed25519/controller/mpc.go#L91-L94 • code/two-party-ed25519/internal/tss/utils.go#L109-L110 • code/two-party-ed25519/service/mpc.go#L109-L110 • code/two-party-ed25519/internal/tss/solana.go#L116 • code/two-party-ed25519/service/mpc.go#L156-L157	Fixed	zircon

		• **code/two-party-ed25519/controller/mpc.go#L169-L172** • **code/two-party-ed25519/controller/mpc.go#L247-L250**		

Code

```
22:     seed, _ := new(big.Int).SetString(reqBody.SKSeed, 10)
23:     kp, wallet, _ := service.KeyGen(seed, reqBody.UserId, reqBody.PartyId)

23:     kp, wallet, _ := service.KeyGen(seed, reqBody.UserId, reqBody.PartyId)

40:     partyIdx, _ := strconv.Atoi(partyId)

46:             s_f64, _ := strconv.ParseFloat(s_f64_str, 64)

50:             msgBN, _ := new(big.Int).SetString(msgBN, 10)
51:             msg = msgBN.Bytes()

61:     userPubkey, _ := solana.PublicKeyFromBase58(user)
62:     payerPubkey, _ := solana.PublicKeyFromBase58(payer)
63:     toPubkey, _ := solana.PublicKeyFromBase58(to)
64:     tokenMintPubkey, _ := solana.PublicKeyFromBase58(tokenMint)
65:     userMintAccountPublicKey, _, _ := solana.FindAssociatedTokenAddress(userPubkey,
tokenMintPubkey)
66:     toMintAccountPublicKey, _, _ := solana.FindAssociatedTokenAddress(toPubkey, tokenMintPubkey)

63:     txMsgBytes, _ := tx.MarshalBinary()

87:     otherAggMsgBN, _ := new(big.Int).SetString(reqBody.OtherAggMsgBN, 10)
88:     otherAggMsg := []*tss.AggMsg1{
89:             tss.NewAggMsg1FromBN(otherAggMsgBN),
90:     }

91:     if len(reqBody.MsgBN) > 0 {
92:             temp, _ := new(big.Int).SetString(reqBody.MsgBN, 10)
93:             msg = temp.Bytes()
94:     }

109:    bn, _ := new(big.Int).SetString(resp.AggMsgBN, 10)
110:    return tss.NewAggMsg1FromBN(bn), resp.Pubkey

109:    seed, _ := new(big.Int).SetString(record.SeedBN, 10)
110:    kp := musig2.NewKeyPair(seed)

116:    pk, _ := solana.PublicKeyFromBase58(pubkey)

156:    bn, _ := new(big.Int).SetString(resp.PartialSigBN, 10)
```

```
157:    return bn.Bytes()

169:    if len(reqBody.MsgBN) > 0 {
170:            temp, _ := new(big.Int).SetString(reqBody.MsgBN, 10)
171:            msg = temp.Bytes()
172:    }

247:    if len(req.MsgBN) > 0 {
248:            temp, _ := new(big.Int).SetString(req.MsgBN, 10)
249:            msg = temp.Bytes()
250:    }
```

Description

zircon :

```
temp, _ := new(big.Int).SetString(//...skip...//, 10)
msg = temp.Bytes()
```

See the definition of SetString:

```
func (z *Int) SetString(s string, base int) (*Int, bool) {
        return z.setFromScanner(strings.NewReader(s), base)
}
```

SetString sets z to the value of s, interpreted in the given base, and returns z and a boolean indicating success. The entire string (not just a prefix) must be valid for success. If SetString fails, the value of z is undefined but the returned value is nil.

The impact is that the malicious string can cause the program to crash when calling `temp.Bytes()`

zircon : These errors have to be handled when the function returns an error type.

The impact is that if these errors are not handled, the program may crash or run in the wrong way, causing serious problems.

Recommendation

zircon : Check the return value of when calling `new(big.Int).SetString(...)`

Consider below fix in the `MPCTransferSol` function

```
temp, success := new(big.Int).SetString(//...skip...//, 10)
if(!success){
    return
}
msg = temp.Bytes()
```

zircon : Check the return error of when calling functions.

Consider below fix:

```
A, err := FUNC()
if(err != nil){
    return err
}
```

Client Response

Fixed.

MWW-11:Unauthorized access to user id vulnerability

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/two-party-ed25519/middleware/authentication/jwt_authentication.go#L48	Fixed	zircon

Code

```
48:             c.Request.Header.Add("id", strconv.Itoa(s_int))
```

Description

zircon : The `JWTAuthentication` function check the jwt token in the request http header, if the token is valid, add the user id in the context of request header. See below:

```
c.Request.Header.Add("id", strconv.Itoa(s_int))
```

and then the logical get user id in the following context:

```
userId := c.Request.Header.Get("id")
// do something private
```

But the type of id in header is `[]string`, it can have many values, include the id post by attacker.

See the definition of `Header` struct and `Add/Get` functions:

```
// Defined in: /usr/local/go/src/net/http/header.go
type Header map[string][]string

func (h Header) Add(key, value string) {
        textproto.MIMEHeader(h).Add(key, value)
}

func (h Header) Get(key string) string {
        return textproto.MIMEHeader(h).Get(key)
}

// Defined in: /usr/local/go/src/net/textproto/header.go
type MIMEHeader map[string][]string

func (h MIMEHeader) Add(key, value string) {
        key = CanonicalMIMEHeaderKey(key)
        h[key] = append(h[key], value)
}

func (h MIMEHeader) Get(key string) string {
        if h == nil {
                return ""
        }
        v := h[CanonicalMIMEHeaderKey(key)]
        if len(v) == 0 {
                return ""
        }
        return v[0]
}
```

`return v[0]` means return the id that attacker want to access.

The impact is that the attacker can access any user's account.

Consider below POC contract

```
curl http://xxxx/mpc/transfer-sol
   -H "Content-Type: application/json"
   -H "Accept: application/json"
   -H "Authorization: Bearer xxx"
   -H "id: 123"
   -d "[json data]"
```

Recommendation

zircon : Remove id field in raw http header.

Consider below fix in the `JWTAuthentication()` function

```
c.Request.Header.Del("id")
c.Request.Header.Add("id", strconv.Itoa(s_int))
```

Client Response

Fixed.

MWW-12:Validate Input Length/Size

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/eddsa_protocol/musig2/signature.go#L197	Fixed	lfzkoala

Code

```
197:    // is the sum of partial_nonces[i] from all parties
```

Description

lfzkoala : `PartialSign` function immediately uses its input but not validate them.

Recommendation

lfzkoala : Checking each input value's length/size, for example, (pseudocode)

```
if len(s) != 64: raise Exception("signature length is wrong")
    if len(pk) != 32: raise Exception("public-key length is wrong")
```

Checking this is also important for detecting the dual public key attack which is relevant to another issue.

Client Response

Fixed.

MWW-13:Vulnerable to Dual Public Key Attack

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/eddsa_protocol/musig2/signature.go#L187-L253	Declined	lfzkoala

Code

```
187:func PartialSign(otherNonces [][2]*alg_ed25519.Ed25519Point, myPriN PrivatePartialNonces,
188:    myPubN PublicPartialNonces, aggKey PublicKeyAgg, myKp KeyPair, message []byte)
*PartialSignature {
189:
190:    R := make([]*alg_ed25519.Ed25519Point, 0)
191:
192:    if len(otherNonces) < 1 {
193:            return nil
194:    }
195:
196:    // Sum up the partial nonces from all parties index-wise, meaning,  R[i]
197:    // is the sum of partial_nonces[i] from all parties
198:    for _, oNonce := range otherNonces {
199:            for i := 0; i < len(oNonce); i++ {
200:                    nonce := oNonce[i].Ge
201:                    accumNonce := myPubN.R[i].ECPAddPoint(&nonce)
202:                    R = append(R, accumNonce)
203:            }
204:    }
205:
206:    // Compute b as hash of nonces, b = hash(fullBytes)
207:    // fullBytes = [3, aggKey.PublicKey, R, message]
208:    bn3 := new(big.Int).SetInt64(3).Bytes()
209:    fullBytes := make([]byte, 0)
210:    fullBytes = append(fullBytes, bn3...)
211:
212:    aggBytes := aggKey.AggPublicKey.AsBytes()
213:    fullBytes = append(fullBytes, aggBytes[:]...)
214:
215:    for _, RR := range R {
216:            RRbyte := RR.AsBytes()
217:            fullBytes = append(fullBytes, RRbyte[:]...)
218:    }
219:    fullBytes = append(fullBytes, message...)
220:
221:    h := sha512.Sum512(fullBytes)
222:    b := alg_ed25519.ECSFromBigInt(new(big.Int).SetBytes(h[:]))
223:
224:    // R = R_0 + b•R_1 + ... + b^(v-1)•R_v
225:    effectiveR := R[0]
226:    for i, nonceRi := range R[1:] {
```

```
227:            bp := b.PowerN(i)
228:            bnR := nonceRi.ECPMul(&bp.Fe)
229:            effectiveR = effectiveR.ECPAddPoint(&bnR.Ge)
230:    }
231:
232:    // r = r_0 + b•r_1 + ... + b^(v-1)•r_v
233:    effectiveSmailR := myPriN.SmailR[0]
234:    for i, nonceri := range myPriN.SmailR[1:] {
235:            bp := b.PowerN(i)
236:            bnr := bp.Mul2(nonceri)
237:            temp := effectiveSmailR.Add2(&bnr)
238:            effectiveSmailR = &temp
239:    }
240:
241:    // Compute Fiat-Shamir challenge of signature
242:    sigChallenge := signatureK(effectiveR, &aggKey.AggPublicKey, message)
243:
244:    // Computes the partial signature
245:    p1 := sigChallenge.Mul2(&aggKey.MusigCoefficient)
246:    p2 := p1.Mul2(&myKp.ExtendedPrivateKey.PrivateKey)
247:    partialSignature := p2.Add2(effectiveSmailR)
248:
249:    return &PartialSignature{
250:            MyPartialS: &partialSignature,
251:            R:          effectiveR,
252:    }
253:}
```

Description

lfzkoala : The code is vulnerable to a critical vulnerability which Chalkias recently exposed a vulnerability. The important statement is that the signing algorithm should only take private key as input. More details see https://medium.com/asecuritysite-when-bob-met-alice/explaining-the-chalkias-ed25519-vulnerability-84443a01a92b. See the list of unsafe libraries https://github.com/MystenLabs/ed25519-unsafe-libs. This is important to not revealing the private key information. We can only take into the private key and derive the public key from it. To avoid double public key attack.

Recommendation

lfzkoala : Investigate the dual public key attack as described in the links above and significantly change the code accordingly. Basically the signing/partial signing methods should only take input the `privateKey` and `privateNonce`

Investigate the dual public key attack as described in the links above and significantly change the code accordingly. Basically the signing/partial signing methods should only take input the `privateKey` and `privateNonce` and `publicKey` and `publicNonce` should be only derived from `privateKey` and `privateNonce`. That is, `publicKey` and `publicNonce` should not be in the input.

Client Response

According to the paper “MuSig2: Simple Two-Round Schnorr Multi-Signatures”, one party should send its public nonce (not private nonce) to the other party. If private nonce is sent to a different party, a security concern may be raised.

MWW-14:Wrong Scalar25519 Decoding Method

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/eddsa_protocol/musig2/keygen.go#L110-L112	Fixed	lfzkoala

Code

```
110:    privateKey[0] &= 248
111:    privateKey[31] &= 63
112:    privateKey[31] |= 64
```

Description

lfzkoala : The `privateKey` decoding method in `NewKeyPairFromSeed` function is not following the Scalar25519 decoding approach listed in, for example, line 323-325 in https://author-tools.ietf.org/idnits?url=https://www.ietf.org/archive/id/draft-irtf-cfrg-curves-07.txt This may lead to privateKey information leakage.

Recommendation

lfzkoala : Follow the IETF standard and make sure the decoding method is correct. The current code uses `privateKey[0] &= 248` `privateKey[31] &= 63` `privateKey[31] |= 64` but it should be `privateKey[0] &= 248` `privateKey[31] &= 127` `privateKey[31] |= 64`

Client Response

Fixed.

MWW-15:fix typo

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/two-party-ed25519/internal/tss/sign.go#L16	Fixed	comcat

Code

```
16:     println("[step1] publicNonces[1]=", BytesToStr(publicNonces.R[0].AsByteArray()))
```

Description

comcat : there is a typo in sign.go `signStepOne` function, it should be R[1] instead of R[0]

```
func SignStepOne(kp *musig2.KeyPair, msg []byte) (*AggMsg1, *SecretAggStepOne) {
        ...
        println("[step1] publicNonces[0]=", BytesToStr(publicNonces.R[0].AsByteArray()))
        println("[step1] publicNonces[1]=", BytesToStr(publicNonces.R[0].AsByteArray()))
}
```

Recommendation

comcat : fix the typo

```
func SignStepOne(kp *musig2.KeyPair, msg []byte) (*AggMsg1, *SecretAggStepOne) {
        ...
        println("[step1] publicNonces[0]=", BytesToStr(publicNonces.R[0].AsByteArray()))
        println("[step1] publicNonces[1]=", BytesToStr(publicNonces.R[1].AsByteArray()))
}
```

Client Response

Fixed.

MWW-16:gc could be optimized in `mulsig2/KeyAggregateN` function

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	code/eddsa_protocol/musig2/keygen.go#L256-L268	Fixed	alansh

Code

```
256:            if key != secondKey {
257:                    // [1, curPk, pk0, pk1]
258:                    fullBytes := make([]byte, 0)
259:                    fullBytes = append(fullBytes, bn1...)
260:                    curKeyByte := key.Ge.AsBytes()
261:                    fullBytes = append(fullBytes, curKeyByte[:]...)
262:
263:                    for _, v := range pubKeys {
264:                            vb := v.Ge.AsBytes()
265:                            fullBytes = append(fullBytes, vb[:]...)
266:                    }
267:
268:                    h := sha512.Sum512(fullBytes)
```

Description

alansh : The size of `fullBytes` is predictable, so the loop can be optimized with only a single instance of size 32*(2+len(pubKeys)) (The same applies for `KeyAggregateNFromBytes`)

Recommendation

alansh : Consider below optimization in the `KeyAggregateN` function

```
        fullBytes := make([]byte, 32*(2+len(pubKeys)))
        for _, key := range pubKeys {
                //musig_coefficient := alg_ed25519.ECSFromBytes(bn1)
                musig_coefficient := alg_ed25519.ECSFromBytes(new(big.Int).SetInt64(1).Bytes())
                if key != secondKey {
                        // [1, curPk, pk0, pk1]
                        fullBytes = fullBytes[:0]
                        fullBytes = append(fullBytes, bn1...)
                         .....
```

Client Response

Fixed.

MWW-17:performance issue in `ed25519/PowerN` function

Category	Severity	Code Reference	Status	Contributor
DOS	Medium	• code/eddsa_protocol/ed25519/scalar.go#L34-L41	Declined	alansh

Code

```
34:func (e *Ed25519Scalar) PowerN(n int) Ed25519Scalar {
35:     result := e
36:     for i := 0; i < n; i++ {
37:             r := result.Mul2(e)
38:             result = &r
39:     }
40:     return *result
41:}
```

Description

alansh : When n is large, `PowerN` will have serious performance issue.

Recommendation

alansh : Consider below square and multiply optimization

```
func (e *Ed25519Scalar) PowerN(n int) Ed25519Scalar {
        if n < 0 {
                panic("invalid n")
        }

        x := e.ToBigInt()
        y := big.NewInt(1)
        q := Q()
        for n > 0 {
                if n%2 == 1 {
                        y = big.NewInt(0).Mod(big.NewInt(0).Mul(x, y), q)
                }
                x = big.NewInt(0).Mod(big.NewInt(0).Mul(x, x), q)

                n /= 2
        }
        return ECSFromBigInt2(y)
}
```

Client Response

There is no need to optimize the powerN(n) in our case. The reason is that n is decided by the number of parties(in our case, n=2). Moreover, we don't see the need to increase n to a big number since it greatly increases the communication overhead between parties and it affects the user's experience.

MWW-18:unnecessary initialization in `mulsig2/KeyAggregateN` function

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/eddsa_protocol/musig2/keygen.go#L255	Fixed	alansh

Code

```
255:          musig_coefficient := alg_ed25519.ECSFromBytes(new(big.Int).SetInt64(1).Bytes())
```

Description

alansh : The initialization of `musig_coefficient` is uncessary

Recommendation

alansh : Consider below fix in the `KeyAggregateN` function

```
var musig_coefficient alg_ed25519.Ed25519Scalar
```

Client Response

Fixed.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.