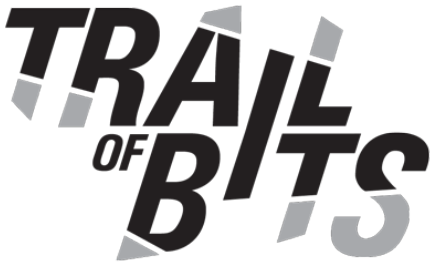

CrossChain-Bridge

Security Assessment

July 12, 2021

Prepared For:
Zhengxin Gao | *Anyswap*
zhengxin.gao@anyswap.exchange

Prepared By:
Fredrik Dahlgren | *Trail of Bits*
fredrik.dahlgren@trailofbits.com

Gabby Beck | *Trail of Bits*
gabby.beck@trailofbits.com

Alex Useche | *Trail of Bits*
alex.useche@trailofbits.com

Changelog:
July 12, 2021: Initial report draft
August 3, 2021: Added Appendix C: Fix Log

Executive Summary

From June 14 to July 9, 2021, Anyswap engaged Trail of Bits to review the security of the CrossChain-Bridge (Anyswap v1/v2) application. The CrossChain-Bridge allows users to swap tokens from a connected wallet and to transfer tokens between a set of supported chains. Trail of Bits conducted this assessment over eight person-weeks, with three engineers working from commit hash `95a9bf6` from the CrossChain-Bridge repository.

At the beginning of the audit, Trail of Bits obtained and built the source code locally, and Anyswap assisted us in building a dynamic testing environment. We used semgrep, CodeQL, gosec, and GCatch, among other static analysis tools, to survey the codebase.

On the first day of the assessment, we conducted an initial review of the Anyswap MPC Node repository, believing it to be within the scope of the audit. However, on the second day of the audit, the Anyswap team informed us that the MPC node had already been audited by another team and asked us to focus on the CrossChain-Bridge. We therefore excluded the Anyswap MPC Node codebase from the scope of the assessment but documented the few issues we had discovered in it up to that point (TOB-ACCB-001, TOB-ACCB-002, TOB-ACCB-003).

Following this reprioritization, we conducted a detailed manual review of the bridge source code, starting with the API and server logic. We performed an in-depth review of the bridge implementation for each supported token, focusing on identifying any improperly used cryptographic functions or issues that could lead to injection attacks or a loss of funds. Our review also included in-depth analysis of the `worker`, `riskctrl`, and `dcrm` logic. At Anyswap's request, we conducted a limited best-effort review of the CrossChain-Router codebase, working from commit `163326e`, and the production deployments at https://anyswap.exchange and https://stable.anyswap.exchange.

Our review resulted in 19 findings ranging from medium to informational severity. Several issues involve the use of insecure cryptographic functions (TOB-ACCB-10, TOB-ACCB-012), which could lead to the exposure of sensitive data. Another pertains to the application's lack of rate-limiting controls, which could leave it susceptible to denial-of-service attacks (TOB-ACCB-015). Additionally, the codebase's lack of unit tests (TOB-ACCB-004) increases the likelihood of logic bugs and vulnerabilities. The low- and informational-severity findings stem from issues including missing HTTP Strict-Transport-Security headers for production deployments (TOB-ACCB-017) and unchecked errors (TOB-ACCB-005), which could lead to application panics.

A number of the issues identified during the audit result from a failure to adhere to cryptography best practices for off-chain applications; others involve deficiencies in the documentation, error handling, data validation, and unit testing. In aggregate, these

findings are indications of a codebase lacking in maturity and of a software development life cycle in which security was not prioritized.

Going forward, Anyswap should develop unit, property, and integration tests for the individual components of the system. Unit tests should cover cases in which users submit invalid or unexpected input to the application via the APIs. Anyswap should also integrate go-fuzz into its CI pipeline to stress-test the parsers and APIs exposed by the CrossChain-Bridge. Constructing adequate automated tests for the expected input, expected output, and validation of the API endpoints will require documentation on those aspects of the system. The documentation should also detail interactions with external entities such as smart contracts and blockchain nodes and should be developed into a threat model for the CrossChain-Bridge.

On Saturday, July 10, 2021, Anyswap alerted Trail of Bits to an attack that resulted in a loss of funds, pointing to the Anyswap MPC Node codebase as the likely source of the issue. Anyswap asked us for assistance in determining the root cause of the attack. As a result, we spent an additional day reviewing this portion of the code to help Anyswap in its remediation efforts, despite the code's exclusion from the scope of our audit. This review culminated in the discovery of an additional cryptographic issue involving the potential for an invalid curve attack, documented in TOB-ACCB-019.

Update: On July 27, 2021, Trail of Bits reviewed fixes implemented for the issues presented in this report. See a detailed review of the current status of each issue in Appendix C.

Project Dashboard

Application Summary

Name	CrossChain-Bridge
Version	`95a9bf6`
Type	Blockchain
Platform	Go

Engagement Summary

Dates	June 7–July 9, 2021
Method	Full knowledge
Consultants Engaged	3
Level of Effort	8 person-weeks

Vulnerability Summary

Total Medium-Severity Issues	7	■■■■■■■
Total Low-Severity Issues	5	■■■■■
Total Informational-Severity Issues	6	■■■■■■
Total Undetermined-Severity Issues	1	■
Total	19	

Category Breakdown

Cryptography	6	■■■■■■
Data Validation	1	■
Timing	2	■■
Error Reporting	1	■
Undefined Behavior	2	■■
Configuration	2	■■
Access Controls	2	■■
Audit and Logging	1	■
Denial of Service	1	■

Data Exposure	1	■
Total	19	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Moderate.** The codebase lacks proper checks on access key permissions (TOB-ACCB-008, TOB-ACCB-009), and the functions involved in admin access control are poorly documented.
Arithmetic	**Strong.** The portions of the codebase that require precise arithmetic use `math/big` to prevent floating-point rounding errors and integer overflows or underflows.
Data Validation	**Moderate.** The RPC middleware handles most of the data validation. However, in many cases, request parameters are not validated before being used in transaction logic, which often leads to the return of unsanitized Go errors in server responses.
Monitoring	**Moderate.** The codebase includes various logging statements, which are saved to text files; however, that is the extent of the centralized logging strategy.
Cryptography	**Moderate.** The cryptographic functions are organized in a centralized manner. In general, though, the code does not demonstrate cryptography best practices for off-chain applications.
Function Composition	**Strong.** The codebase is cleanly separated into well-defined packages. It is generally easy to find each implemented functionality and to build and test individual components of the system.
Resource & Rate Limiting	**Weak.** There does not appear to be any limit on the rate at which requests can be sent to the CrossChain-Bridge or CrossChain-Router (TOB-ACCB-015).
Specification	**Weak.** The code has very little documentation and no high-level documentation whatsoever. This makes it very difficult to understand and reason about the security properties of the system and ultimately limited the scope of our review.
Testing & Verification	**Weak.** The Anyswap code does not contain any tests.

Engagement Goals

The engagement was scoped to provide a security assessment of the CrossChain-Bridge application.

Specifically, we sought to answer the following questions:

- Are the back-end components prone to generic data validation errors that could lead to injection vulnerabilities?
- Are the components vulnerable to memory corruption issues?
- Do any of the components leak sensitive information through errors or server responses sent to clients?
- Do the components handle sensitive data in a safe and cryptographically secure manner?
- Could an attacker influence the arithmetic operations to his or her advantage?
- Could an attacker reduce the system availability?
- Could an attacker perform unauthorized operations?

We also conducted a best-effort review of the CrossChain-Router application and performed light testing of the production deployments of the CrossChain-Bridge and CrossChain-Router.

Coverage

CrossChain-Bridge. To audit this codebase, we set up and launched the application locally. The Anyswap team also set up a test environment in a server provided by Trail of Bits. We then leveraged the API to conduct dynamic testing and examined the logs created by the application. We focused on gaining an understanding of how the application processes input received via REST and RPC endpoints exposed to users and tested the code against attacks of a range of difficulty levels. We also performed an in-depth review of the bridge implementation for each token supported by the CrossChain-Bridge. In addition to dynamic testing, we conducted manual static analysis of the CrossChain-Bridge code, leveraged tools such as gosec, errcheck, and semgrep, and confirmed findings resulting from dynamic analysis through a manual code review.

CrossChain-Router. In the last week of the audit, we conducted a best-effort review of the CrossChain-Router codebase. Because this codebase shares a significant amount of code with the CrossChain-Bridge, we focused on reviewing code unique to the router application, including the logic in the `/router` directory and the RPC endpoints exposed by the application.

Anyswap MPC Node. This codebase was not within the scope of the audit. However, after Anyswap notified Trail of Bits of an attack that resulted in a loss of funds, Trail of Bits began investigating the computation of ECDSA signatures to help identify the vulnerability affecting the Anyswap code. This analysis does not constitute a complete security review of the Anyswap MPC Node or a full incident response investigation.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Let `ec2.GenSafeRandomInt` return `(L/2 - 1)`-bit primes and implement a check in `random.GetSafeRandomPrimeInt2` verifying that the resulting safe prime is less than `2^(L/2)`.** (Note that, with overwhelming probability, this property will be true, and the check will not significantly impact the running time of the algorithm.) TOB-ACCB-001

❑ **Ensure that the return value from `random.GetRandomInt` is checked before it is used in `random.GetRandomIntFromZn`, `random.GetRandomIntFromZnStar`, and `ec2.Commit`.** TOB-ACCB-002

❑ **Create a channel in `dcrm.GetCurNodeSignInfo` to which each goroutine writes output.** That way, individual elements of the output can be safely appended to `ret` in a single designated goroutine. TOB-ACCB-003

❑ **Add thorough unit and integration tests to the codebase to facilitate more complicated testing scenarios.** TOB-ACCB-004

❑ **Implement checks for all errors returned by important functions.** If a function call fails, take appropriate action (or at least log the error). Doing so will prevent incorrect computations. TOB-ACCB-005

❑ **Update `StartScanJob` to read from the correct configuration.** TOB-ACCB-006

❑ **Update the `GetTransactionStatus` API to return both a `tokens.TxStatus` structure and an error, and have `GetTransactionReceipt` and `GetTransactionStatus` return `tokens.ErrRPCQueryError` when a call to the remote RPC gateway fails.** TOB-ACCB-007

❑ **Either remove the function or add a check verifying the permissions of pre-existing files.** Additionally, consider updating the permissions of a file to read-only (`0400`) after the file has been written. TOB-ACCB-008

❑ **Use `os.Stat` to check the file permissions of the key and password files and ensure that the application returns an error if they are not set to `0400`.** TOB-ACCB-009

❑ **Authenticate the entire content of the key file.** Take care to encode each field in the `encryptedKeyJSONVx` structure to bytes, to compute the MAC over the resulting byte array, and to store the resulting byte array with the MAC, rather than the authenticated JSON structure. Note that the encoding must be *canonical* (i.e., must use a fixed number of bytes for each field) to avoid canonicalization issues. TOB-ACCB-010

❑ **Change the code such that it no longer executes `defer utxoLock.Lock()`, add a line calling `utxoLock.Unlock()` to the end of the function, and, instead of using `!b.isUtxoLocked`, check whether `lockedUtxos[key] == 0`.** TOB-ACCB-011

❑ **If the protocol version is upgraded, remove version 1 of the protocol from the application.** TOB-ACCB-012

❑ **To prevent leaks of plaintext information, replace `crypto.ToECDSAUnsafe` with `crypto.ToECDSA` and ensure that the function returns a `keystore.ErrDecrypt` error if the call to `crypto.ToECDSA` fails.** TOB-ACCB-013

❑ **Document all API endpoints exposed by the CrossChain-Bridge APIs.** Additionally, remove any endpoints that are no longer required or should not be exposed to users from the code. TOB-ACCB-014

❑ **Consider restricting the rate at which a single IP address can make requests.** Alternatively, it may be possible to impose these rate-limiting controls on a given set of blockchain addresses. TOB-ACCB-015

❑ **Deprecate the use of TLSv1.1 in the production environments of the CrossChain-Bridge and CrossChain-Router (at https://anyswap.exchange and https://stable.anyswap.exchange, respectively).** TOB-ACCB-016

❑ **Use the HTTP STS header to prevent the accidental referencing of the HTTP protocol in place of HTTPS.** TOB-ACCB-017

❑ **Configure the server to conceal its version number.** TOB-ACCB-018

❑ **Adjust the code responsible for verifying commitments such that it also ensures that each elliptic curve point is on the `secp256k1` curve.** TOB-ACCB-019

Long Term

❑ **Refactor the entire prime-generation implementation and rename its methods to better reflect their purposes.** For example, the function `ec2.GenRandomInt` does not generate *any* random integers; it generates prime numbers and should probably be called

`ec2.GenRandomPrime`. The function `random.GetSafeRandomInt` does not return an integer or a safe prime; it returns a random prime number and should probably be called `random.GetRandomPrime`. Additionally, document the `ec2.GenRandomInt` and `ec2.GenRandomSafePrime` entry points to inform users of the meaning of the length parameter. TOB-ACCB-001

❑ **Review the codebase for potential `nil` pointer dereferences.** TOB-ACCB-002

❑ **Run the code using the Go race detector to uncover any race conditions.** TOB-ACCB-003

❑ **Incorporate unit, integration, and end-to-end tests into the CI/CD pipeline.** TOB-ACCB-004

❑ **Use a static analyzer that can detect missing error checks as part of the development process, and handle all issues it identifies.** Explicitly mark unimportant cases as "ignored," assigning the returned error to the blank identifier (the underscore character). Consider adding a checksum for important state files to detect random filesystem errors or even introducing cryptographic authentication code to prevent attackers from manipulating them. Lastly, consider loading the file at the `stateCacheFilePath` path only once, at the node's startup, to improve efficiency. TOB-ACCB-005

❑ **Audit calls to `ioutil.WriteFile` and `os.WriteFile` to ensure that they always check file permissions if sensitive data is written to disk.** TOB-ACCB-008

❑ **Audit calls to `ioutil.ReadFile`, `os.ReadFile`, and `os.Open` to ensure that they always check file permissions when sensitive data is read from disk.** TOB-ACCB-009

❑ **Formulate a plan for migrating away from version 1 of the protocol as quickly as possible.** Additionally, address the timing leak in `pkcs7Unpad` and rewrite `aesCBCDecrypt` so that it does not leak information about the plaintext padding to the rest of the application (or the user). TOB-ACCB-012

❑ **Review server configurations to ensure all standards are up to date and adhere to best practices.** TOB-ACCB-016

❑ **Be mindful of server configuration best practices and consider using services such as SSL Labs to quickly review information on non-ideal SSL/TLS configurations.** TOB-ACCB-017

❑ **Review the configuration of all server software to prevent the public exposure of sensitive information that could be used to fingerprint the system.** TOB-ACCB-018

Findings Summary

#	Title	Type	Severity
1	MPC node safe prime-generation algorithm drops one bit of entropy	Cryptography	Low
2	Potential nil pointer dereference in random.GetRandomIntFromZn, random.GetRandomIntFromZnStar, and ec2.Commit	Data Validation	Low
3	Appending data from multiple goroutines to one slice is not concurrency-safe	Timing	Undetermined
4	Insufficient test coverage	Error Reporting	Medium
5	Unchecked errors	Undefined Behavior	Low
6	Worker reads from the wrong chain configuration when starting transaction pool scans	Configuration	Informational
7	Bridge.getStableReceipt returns the wrong error type if Bridge.getTransactionReceipt fails to reach the remote API endpoint	Error Reporting	Informational
8	SaveECDSA should verify file permissions if the file exists	Access Controls	Informational
9	LoadKeyStore should verify permissions on the key and password files	Access Controls	Medium
10	keystore.DecryptKey makes cryptographic choices based on untrusted data	Cryptography	Medium
11	Thread deadlocks on utxoLock RWMutex	Timing	Informational
12	aesCBCDecrypt is vulnerable to CBC padding oracle attacks	Cryptography	Medium
13	crypto.ToECDSAUnsafe discards error codes	Cryptography	Informational
14	Undocumented API endpoints	Auditing and	Informational

		Logging	
15	Lack of rate-limiting controls	Denial of Service	Medium
16	Support for insecure TLS protocols and weak ciphers	Cryptography	Medium
17	Lack of HTTP Strict-Transport-Security (HSTS) header	Configuration	Low
18	Server exposes software version in HTTP headers	Data Exposure	Low
19	Risk of invalid curve attacks in MPC nodes	Cryptography	Medium

1. MPC node safe prime-generation algorithm drops one bit of entropy

Severity: Low | Difficulty: High
Type: Cryptography | Finding ID: TOB-ACCB-001
Target: `Anyswap-MPCNode/internal/common/math/random/random.go`

Description

The `ec2.GenRandomInt` and `ec2.GenRandomSafePrime` functions are used to generate safe primes for the Paillier encryption and zero-knowledge proofs implemented by the MPC node code. (The length, `L`, given to the two functions denotes the size of the resultant key, not the generated safe primes.)

The `ec2.GenRandomInt` function calls `random.GetSafeRandomInt` with a length of `L/2` to generate a prime of `(L/2 - 2)` bits, which is then written to the `RndInt` channel.

```
func GetSafeRandomInt(length int) *big.Int {
    var rndInt *big.Int
    var err error
    for {
        rndInt, err = rand.Prime(rand.Reader, length-2)
        if err == nil {
            //fmt.Println("Generate Safe Random Int Success!")
            break
        }

        time.Sleep(time.Duration(1000000)) //1000 000 000 == 1s
    }

    return rndInt
}
```

Figure 1.1: The `random.GetSafeRandomInt` function generates a random prime of length `L-2`.

To generate a safe prime, `ec2.GenRandomSafePrime` reads a prime of `(L/2 - 2)` bits, `q`, from the `RndInt` channel and then calls `random.GetSafeRandomPrimeInt2`; this function returns `p = 2*q + 1` if `p` is prime and returns `nil` if it is not. Since `q` is a prime of `(L/2 - 2)` bits, `p` is a prime of `(L/2 - 1)` bits, and not a prime of `L/2` bits as expected.

This same issue is present in `random.GetSafeRandomPrimeInt`, but this function does not appear to be used in the code.

The loss of one bit of entropy does not appear to have a significant impact on the codebase; however, since we have not reviewed the MPC node codebase, we cannot definitively state that it is not a serious issue. Note too that in certain applications, notably the nonce used for ECDSA signatures, the loss of a single bit of entropy can seriously weaken the entire system.

Recommendations

Short term, let `ec2.GenSafeRandomInt` return `(L/2 - 1)`-bit primes and implement a check in `random.GetSafeRandomPrimeInt2` verifying that the resulting safe prime is less than `2^(L/2)`. (Note that, with overwhelming probability, this property will be true, and the check will not significantly impact the running time of the algorithm.)

Long term, refactor the entire prime-generation implementation and rename its methods to better reflect their purposes. For example, the function `ec2.GenRandomInt` does not generate *any* random integers; it generates prime numbers and should probably be called `ec2.GenRandomPrime`. The function `random.GetSafeRandomInt` does not return an integer or a safe prime; it returns a random prime number and should probably be called `random.GetRandomPrime`. Additionally, document the `ec2.GenRandomInt` and `ec2.GenRandomSafePrime` entry points to inform users of the meaning of the length parameter.

We also recommend that Anyswap remove the length parameter from all functions in which the length is not explicitly used by the key-generation API (e.g., `ec2.GenRandomSafePrime`, `ec2.GenerateKeyPair`, and `ntilde.GenerateNtildeH1H2`). Passing a length to these functions suggests to the user that it is possible to alter the key size by altering the input length, which is not the case. It also makes the API more difficult to understand and more prone to misuse.

2. Potential nil pointer dereference in random.GetRandomIntFromZn, random.GetRandomIntFromZnStar, and ec2.Commit

Severity: Low | Difficulty: High
Type: Data Validation | Finding ID: TOB-ACCB-002
Target: `Anyswap-MPCNode/internal/common/math/random/random.go`

Description
The `random.GetRandomInt` function is called from `random.GetRandomIntFromZn`, `random.GetRandomIntFromZnStar`, and `ec2.Commit` to generate a random integer of a given length. The function may return `nil` if `rand.Int` fails to read enough data from the provided random number generator.

```
func GetRandomInt(length int) *big.Int {
    one := big.NewInt(1)
    maxi := new(big.Int).Lsh(one, uint(length))
    maxi = new(big.Int).Sub(maxi, one)
    rndNum, err := rand.Int(rand.Reader, maxi)
    if err != nil {
        return nil
    }

    return rndNum
}
```

Figure 2.1: `GetRandomInt` may return a `nil` value if the random number generator `rand.Reader` fails to generate enough randomness.

None of the call sites check for a `nil` return value before dereferencing the `big.Int` pointer returned by `GetRandomInt`. A nil pointer dereference resulting from a `nil` return value will cause a segmentation violation and a panic.

The same issue is present in `random.GetRandomPrimeInt`, but this function is not used anywhere in the codebase.

```
   for {
        rndNumZn = GetRandomInt(n.BitLen())
        if rndNumZn.Cmp(n) < 0 && rndNumZn.Cmp(zero) >= 0 {
            break
        }
    }
```

Figure 2.2: The return value from `GetRandomInt` is dereferenced without a `nil` return value check in `random.GetRandomIntFromZn`.

```
for {
     rndNumZnStar = GetRandomInt(n.BitLen())
     if rndNumZnStar.Cmp(n) < 0 && rndNumZnStar.Cmp(one) >= 0 &&
         gcdNum.GCD(nil, nil, rndNumZnStar, n).Cmp(one) == 0 {
         break
     }
 }
```

Figure 2.3: The return value from `GetRandomInt` is dereferenced without a `nil` return value check in `random.GetRandomIntFromZnStar`.

```
rnd := random.GetRandomInt(256)

sha3256 := sha3.New256()

sha3256.Write(rnd.Bytes())
```

Figure 2.4: The return value from `GetRandomInt` is dereferenced without a `nil` return value check in `ec2.Commit`.

Recommendations

Short term, ensure that the return value from `random.GetRandomInt` is checked before it is used in `random.GetRandomIntFromZn`, `random.GetRandomIntFromZnStar`, and `ec2.Commit`.

Long term, review the codebase for potential `nil` pointer dereferences.

3. Appending data from multiple goroutines to one slice is not concurrency-safe

Severity: Undetermined
Type: Timing
Target: `Anyswap-MPCNode/dcrm/sign.go`

Difficulty: N/A
Finding ID: TOB-ACCB-003

Description
The `dcrm.GetCurNodeSignInfo` function loops over the key-value pairs in the map `LdbPubKeyData.Map` and for each pair creates a goroutine that appends data to the `ret` slice. This practice is not concurrency-safe and may cause memory corruption.

```
func GetCurNodeSignInfo(geter_acc string) ([]*SignCurNodeInfo, string, error) {
       var ret []*SignCurNodeInfo
       var wg sync.WaitGroup
       LdbPubKeyData.RLock()
       for k, v := range LdbPubKeyData.Map {
           wg.Add(1)
           go func(key string,value interface{}) {
               defer wg.Done()

               vv,ok := value.(*AcceptSignData)
               [...]

               los := ...
               ret = append(ret, los)
               [...]
           }(k,v)
       }
       LdbPubKeyData.RUnlock()
       wg.Wait()
       return ret, "", nil
}
```

Figure 3.1: The `dcrm.GetCurNodeSignInfo` function creates an anonymous goroutine for each key-value pair in `LdbPubKeyData.Map`, and each goroutine appends data to the `ret` slice.

Using semgrep, we identified three other instances of this issue in the Anyswap MPC Node repository:

- `Anyswap-MPCNode/dcrm/lockout.go` (line 206)
- `Anyswap-MPCNode/dcrm/reqaddr.go` (line 255)
- `Anyswap-MPCNode/dcrm/reshare.go` (line 196)

Since we have not reviewed the MPC node codebase, the severity of this issue is undetermined. In general, however, race conditions can cause memory corruption or data loss and can be exploitable by a remote attacker.

Recommendations
Short term, create a channel in `dcrm.GetCurNodeSignInfo` to which each goroutine writes output. That way, individual elements of the output can be safely appended to `ret` in a single designated goroutine.

Long term, run the code using the Go race detector to uncover any race conditions.

4. Insufficient test coverage

Severity: Medium
Difficulty: Low
Type: Error Reporting
Finding ID: TOB-ACCB-004
Target: `CrossChain-Bridge, CrossChain-Router`

Description
The CrossChain-Bridge codebase does not include sufficient test cases. Robust unit and integration tests are critical to the detection of certain bugs and logic errors early in the development process. Projects should include thorough coverage against bad input as well as "happy path" scenarios, as it greatly increases users' and developers' confidence in the functionality of code.

Test architecture also benefits from unit tests. These tests verify that subsets of the application work locally as expected and allow developers to effectively detect inconsistencies in code behavior before deployment.

Exploit Scenario
Anyswap discovers a bug in the application and writes a patch to update it. The new code causes backward incompatibility and deviations from the application's expected behavior. However, because of the insufficient test coverage, these issues are not caught by the test suite.

Recommendations
Short term, add thorough unit and integration tests to the codebase to facilitate more complicated testing scenarios.

Long term, incorporate unit, integration, and end-to-end tests into the CI/CD pipeline.

5. Unchecked errors

Severity: Low | Difficulty: High
Type: Undefined Behavior | Finding ID: TOB-ACCB-005
Target: `{CrossChain-Bridge, CrossChain-Router}/rpc/restapi/api.go`, `{CrossChain-Bridge, CrossChain-Router}/rpc/server/server.go`

Description
The code lacks checks for errors returned by certain functions upon a failure. This incorrect error handling may cause undefined behavior or, under certain circumstances, a Go panic.

```
func initRouterSwapRouter(r *mux.Router) {
       rpcserver := rpc.NewServer()
       rpcserver.RegisterCodec(rpcjson.NewCodec(), "application/json")
       _ = rpcserver.RegisterService(new(rpcapi.RouterSwapAPI), "swap")
```

Figure 5.1: A call to `rpcserver.RegisterService` results in an error, which is ignored.

```
func writeResponse(w http.ResponseWriter, resp interface{}, err error) {
   // Note: must set header before write header
   if err == nil {
       w.Header().Set("Content-Type", "application/json")
   }
   w.WriteHeader(http.StatusOK)
   if err == nil {
       jsonData, _ := json.Marshal(resp)
       _, _ = w.Write(jsonData)
   }
```

Figure 5.2: An error returned by the function that writes API responses is ignored.

Exploit Scenario
An error occurs when writing a server response. Because the error is not detected, it is very difficult to troubleshoot and diagnose the issue.

Recommendations
Short term, implement checks for all errors returned by important functions. If a function call fails, take appropriate action (or at least log the error).

Long term, use a static analyzer that can detect missing error checks as part of the development process, and handle all issues it identifies. Explicitly mark unimportant cases as "ignored," assigning the returned error to the blank identifier (the underscore character).

6. Worker reads from the wrong chain configuration when starting transaction pool scans

Severity: Informational
Type: Configuration
Target: `CrossChain-Bridge/worker/scan.go`

Difficulty: N/A
Finding ID: TOB-ACCB-006

Description
In `StartScanJob`, the `worker` starts a number of asynchronous scanning jobs depending on the configurations provided for the source and destination chains.

```
func StartScanJob(isServer bool) {
    srcChainCfg := tokens.SrcBridge.GetChainConfig()
    if srcChainCfg.EnableScan {
        go tokens.SrcBridge.StartChainTransactionScanJob()
        if srcChainCfg.EnableScanPool {
            go tokens.SrcBridge.StartPoolTransactionScanJob()
        }
        if btc.BridgeInstance != nil {
            go btc.BridgeInstance.StartSwapHistoryScanJob()
        }
    }

    dstChainCfg := tokens.DstBridge.GetChainConfig()
    if dstChainCfg.EnableScan {
        go tokens.DstBridge.StartChainTransactionScanJob()
        if srcChainCfg.EnableScanPool {
            go tokens.DstBridge.StartPoolTransactionScanJob()
        }
    }
}
```

Figure 6.1: The `StartScanJob` function reads from the wrong configuration when starting a destination chain pool-scanning job.

When determining whether to start a scan of the destination chain transaction pool, the function mistakenly reads from the source chain configuration, rather than the destination chain configuration.

Recommendations
Short term, update `StartScanJob` to read from the correct configuration.

7. Bridge.getStableReceipt returns the wrong error type if Bridge.getTransactionReceipt fails to reach the remote API endpoint

Severity: Informational
Type: Error Reporting
Target: `CrossChain-Bridge/tokens/eth/verifytx.go`

Difficulty: N/A
Finding ID: TOB-ACCB-007

Description
The bridge API `GetTransactionStatus` returns a pointer to a default-initialized `tokens.TxStatus` structure if the call to `GetTransactionReceipt` fails.

```
func (b *Bridge) GetTransactionStatus(txHash string) *tokens.TxStatus {
    var txStatus tokens.TxStatus
    txr, url, err := b.GetTransactionReceipt(txHash)
    if err != nil {
        log.Trace("GetTransactionReceipt fail", "hash", txHash, "err", err)
        return &txStatus
    }
    [...]
```

Figure 7.1: `GetTransactionStatus` returns a pointer to a default-initialized `tokens.TxStatus` structure, which makes it difficult for the caller to determine whether the call succeeded or failed.

A failure could be caused (for example) by connection issues with the remote RPC gateway.

The `txstatus.Receipt` field is read in `getStableReceipt`. If the receipt is default-initialized (as `nil`), the function returns a `tokens.ErrTxNotStable` error, incorrectly indicating that the transaction has not yet been confirmed.

```
func (b *Bridge) getStableReceipt(swapInfo *tokens.TxSwapInfo) (*types.RPCTxReceipt, error) {
	txStatus := b.GetTransactionStatus(swapInfo.Hash)
	swapInfo.Height = txStatus.BlockHeight  // Height
	swapInfo.Timestamp = txStatus.BlockTime // Timestamp
	receipt, ok := txStatus.Receipt.(*types.RPCTxReceipt)
	if !ok || receipt == nil {
		return nil, tokens.ErrTxNotStable
	}
```

Figure 7.2: The getStableReceipt function returns a `tokens.ErrTxNotStable` error if the remote RPC endpoint is not reachable, indicating that the transaction has not yet been confirmed.

Recommendations
Short term, update the `GetTransactionStatus` API to return both a `tokens.TxStatus` structure and an error, and have `GetTransactionReceipt` and `GetTransactionStatus` return `tokens.ErrRPCQueryError` when a call to the remote RPC gateway fails.

8. SaveECDSA should verify file permissions if the file exists

Severity: Informational | Difficulty: N/A
Type: Access Controls | Finding ID: TOB-ACCB-008
Target: `{CrossChain-Bridge, CrossChain-Router}/tools/crypto/crypto.go`

Description
The `SaveECDSA` function can be used to export an ECDSA private key to a file. If the file to which a key will be exported does not yet exist, it is created with `0600` permissions. However, if the file already exists, its permissions are not updated.

```
func SaveECDSA(file string, key *ecdsa.PrivateKey) error {
	k := hex.EncodeToString(FromECDSA(key))
	return ioutil.WriteFile(file, []byte(k), 0600)
}
```

Figure 8.1: If the given file exists, the file permissions are not checked or updated.

The severity of this issue was set to informational because the function is not used by the application. However, it could be called in a refactored version of the CrossChain-Bridge.

Exploit Scenario
A future version of the CrossChain-Bridge uses `SaveECDSA` to store ECDSA private keys to disk. A malicious user creates a file with lax permissions (e.g., `0666`) at the given location. When the function is called, the permissions are not checked or updated, which means that the file containing the private key will be world-readable and world-writable.

Recommendations
Short term, either remove the function or add a check verifying the permissions of pre-existing files. Additionally, consider updating the permissions of a file to read-only (`0400`) after the file has been written.

Long term, audit calls to `ioutil.WriteFile` and `os.WriteFile` to ensure that they always check file permissions if sensitive data is written to disk.

9. LoadKeyStore should verify permissions on the key and password files

Severity: Medium | Difficulty: High
Type: Access Controls | Finding ID: TOB-ACCB-009
Target: {CrossChain-Bridge, CrossChain-Router}/tools/loadkeystore.go

Description

The `tools.LoadKeyStore` function loads the key and password from a file. The function should check that the file permissions are set correctly (i.e., to `0400`) and should fail with an error if they are too permissive.

```
func LoadKeyStore(keyfile, passfile string) (*keystore.Key, error) {
	keyjson, err := ioutil.ReadFile(keyfile)
	if err != nil {
		return nil, fmt.Errorf("read keystore fail %v", err)
	}
	passdata, err := ioutil.ReadFile(passfile)
	if err != nil {
		return nil, fmt.Errorf("read password fail %v", err)
	}
	passwd := strings.TrimSpace(string(passdata))
	key, err := keystore.DecryptKey(keyjson, passwd)
	if err != nil {
		return nil, fmt.Errorf("decrypt key fail %v", err)
	}
	return key, nil
}
```

Figure 9.1: To ensure defense in depth, `tools.LoadKeyStore` should check the file permissions on the key and password files and fail if they are not set correctly.

The same issue is present in `crypto.LoadECDSA`, which calls `os.Open` to read a private ECDSA key from a file.

Exploit Scenario

An Anyswap developer sets up the bridge server but forgets to set restrictive permissions on the key and password files. Since the application does not check the files' permissions, the configuration mistake goes unnoticed. A malicious user with access to the server reads the two files, recovers the private key, and uses it to impersonate the bridge when communicating with the DCRM nodes.

Recommendations

Short term, use `os.Stat` to check the file permissions of the key and password files and ensure that the application returns an error if they are not set to `0400`.

Long term, audit calls to `ioutil.ReadFile`, `os.ReadFile`, and `os.Open` to ensure that they always check file permissions when sensitive data is read from disk.

10. keystore.DecryptKey makes cryptographic choices based on untrusted data

Severity: Medium | Difficulty: Medium
Type: Cryptography | Finding ID: TOB-ACCB-010
Target: {CrossChain-Bridge, CrossChain-Router}/tools/keystore/passphrase.go

Description
The key file passed to `keystore.DecryptKey` is a JSON dictionary that contains cipher and key derivation parameters. These include the algorithm version (version 1 or 3), the encryption algorithms (`aes-128-cbc` and `pbkdf2` in version 1 and `aes-128-ctr` and `scrypt` in version 3), and the algorithm parameters. This data is not authenticated, which means that an attacker with write permissions to the key file could alter the input parameters and replace the algorithms used to decrypt the encrypted key data.

```
func DecryptKey(keyjson []byte, auth string) (*Key, error) {
	// Parse the json into a simple map to fetch the key version
	m := make(map[string]interface{})
	if err := json.Unmarshal(keyjson, &m); err != nil {
		return nil, err
	}
	// Depending on the version try to parse one way or another
	var (
		keyBytes, keyID []byte
		err             error
	)
	if version, ok := m["version"].(string); ok && version == "1" {
		k := new(encryptedKeyJSONV1)
		err = json.Unmarshal(keyjson, k)
		if err != nil {
			return nil, err
		}
		keyBytes, keyID, err = decryptKeyV1(k, auth)
	} else {
		k := new(encryptedKeyJSONV3)
		err = json.Unmarshal(keyjson, k)
		if err != nil {
			return nil, err
		}
		keyBytes, keyID, err = decryptKeyV3(k, auth)
	}
	[...]
}
```

Figure 10.1: The keystore version is not authenticated.

Although we have not found a way to use this deficiency to attack the encryption directly, the use of unauthenticated parameters to determine cryptographic algorithms and

parameters is generally not secure. This practice is sometimes known as “in-band protocol negotiation” and has led to vulnerabilities in a number of protocols, including in JSON web tokens and the AWS Encryption SDK.

The same issue is present in `decryptKeyV1`, `decryptKeyV3`, `DecryptDataV3`, and `getKDFKey`, which all read unauthenticated data from input.

Exploit Scenario
A malicious user edits the key file and changes the initialization vector (IV). Since the key derivation function (KDF) parameters (the algorithm and algorithm parameters) remain unchanged, the message authentication code (MAC) will still be verified, but the ciphertext will decrypt to a different plaintext. This plaintext will very likely be a valid private key, in which case `keystore.DecryptKey` will not return an error. As a result, transactions will be signed with the wrong private key, causing communication with the DCRM nodes to fail.

Recommendations
Short term, authenticate the entire content of the key file. Take care to encode each field in the `encryptedKeyJSONVx` structure to bytes, to compute the MAC over the resulting byte array, and to store the resulting byte array with the MAC, rather than the authenticated JSON structure. Note that the encoding must be *canonical* (i.e., must use a fixed number of bytes for each field) to avoid canonicalization issues.

11. Thread deadlocks on utxoLock RWMutex

Severity: Informational
Difficulty: Low
Type: Timing
Finding ID: TOB-ACCB-011
Target: `CrossChain-Bridge/tokens/colx/lockUtxo.go`

Description
The `setUnlockUtxoCond` function acquires `utxoLock` for writing and then immediately attempts to lock it for reading, resulting in a deadlock. In addition, the function attempts to lock `utxoLock` for writing immediately before the function returns, which prevents any subsequent threads from gaining access to it. The function is called as part of attempts to aggregate unspent transaction outputs (UTXOs) and, in bridges using ColossusXT coin, prevents the thread running `StartAggregateJob` from progressing.

```
func (b *Bridge) isUtxoLocked(key utxokey) bool {
        utxoLock.RLock()
        defer utxoLock.RUnlock()
        return (lockedUtxos[key] == 1)
}

[...]

func (b *Bridge) setUnlockUtxoCond(key utxokey, cond func() bool) error {
        utxoLock.Lock()
        defer utxoLock.Lock()
        if !b.isUtxoLocked(key) {
                return errNotLocked
        }
        unlockConds[key] = cond
        return nil
}
```

Figure 11.1: The function attempts to lock the reader/writer mutex for both writing and reading and then, before returning, attempts to lock it for writing again.

Recommendations
Short term, change the code such that it no longer executes `defer utxoLock.Lock()`, add a line calling `utxoLock.Unlock()` to the end of the function, and, instead of using `!b.isUtxoLocked`, check whether `lockedUtxos[key] == 0`.

12. aesCBCDecrypt is vulnerable to CBC padding oracle attacks

Severity: Medium | Difficulty: High
Type: Cryptography | Finding ID: TOB-ACCB-012
Target: {CrossChain-Bridge, CrossChain-Router}/tools/keystore/aes.go

Description
In version 1 of the protocol, keys are encrypted using AES-128 in cipher-block chaining (CBC) mode. To decrypt a version 1 key, `keystore.DecryptKey` calls `decryptKeyV1`, which derives a MAC key and an encryption key using `getKDFKey`. The function then calls `aesCBCDecrypt` to decrypt the ciphertext.

```
func aesCBCDecrypt(key, cipherText, iv []byte) ([]byte, error) {
	aesBlock, err := aes.NewCipher(key)
	if err != nil {
		return nil, err
	}
	decrypter := cipher.NewCBCDecrypter(aesBlock, iv)
	paddedPlaintext := make([]byte, len(cipherText))
	decrypter.CryptBlocks(paddedPlaintext, cipherText)
	plaintext := pkcs7Unpad(paddedPlaintext)
	if plaintext == nil {
		return nil, ErrDecrypt
	}
	return plaintext, err
}
```

Figure 12.1: The `aesCBCDecrypt` function will return `nil`, `keystore.ErrDecrypt` if `pkcs7Unpad` fails. Otherwise, it will return `plaintext`, `nil`. This implementation creates an opportunity for a padding oracle attack against the decryption algorithm.

The `aesCBCDecrypt` function decrypts the given ciphertext and then verifies and removes the PKCS-7 padding by calling `pkcs7Unpad`.

```
func pkcs7Unpad(in []byte) []byte {
	[...]

	for i := len(in) - 1; i > len(in)-int(padding)-1; i-- {
		if in[i] != padding {
			return nil
		}
	}
	return in[:len(in)-int(padding)]
}
```

Figure 12.2: The `pkcs7Unpad` function exits early if a padding byte is found to be invalid, leading to the creation of a timing oracle. The oracle can then be used to launch a padding oracle attack against the decryption algorithm.

The `pkcs7Unpad` function will exit early if the padding cannot be verified. An attacker could leverage this timing difference to execute a CBC padding oracle attack against the decryption algorithm. Since `aesCBCDecrypt` will return `keystore.ErrDecrypt` if the padding fails to verify and a `nil` error if the verification succeeds, an attacker could simply monitor the return values from `aesCBCDecrypt` to determine whether the padding was correct.

If the plaintext is at least 16 bytes long, it will not be possible to execute this attack against the current version of the application. This is because the attack requires the attacker to mutate the ciphertext before it is decrypted. However, when the plaintext is at least 16 bytes long, the ciphertext is protected by a MAC, which is verified before the ciphertext is decrypted. Since the algorithm is currently used to encrypt 32-byte private keys, these keys remain safe from the attack.

For plaintext shorter than 16 bytes, an attacker could simply mutate the IV, which is not protected by the MAC. Thus, if version 1 of the protocol were ever used to encrypt short plaintext (of 15 bytes at the most), the attack could be used to recover the entirety of the plaintext without the password.

Exploit Scenario
A future version of the CrossChain-Bridge uses version 1 of the protocol to encrypt a 15-byte token. A malicious user who has read-write access to the key file (but cannot read the password file) edits the IV and starts the application. By monitoring the error message returned from the bridge, the malicious user is able to successively determine each byte of the plaintext and thus recover the encrypted token.

Recommendations
Short term, if the protocol version is upgraded, remove version 1 of the protocol from the application.

Long term, formulate a plan for migrating away from version 1 of the protocol as quickly as possible. Additionally, address the timing leak in `pkcs7Unpad` and rewrite `aesCBCDecrypt` so that it does not leak information about the plaintext padding to the rest of the application (or the user).

13. crypto.ToECDSAUnsafe discards error codes

Severity: Informational Difficulty: N/A
Type: Cryptography Finding ID: TOB-ACCB-013
Target: {CrossChain-Bridge, CrossChain-Router}/tools/keystore/passphrase.go

Description
After a private key has been decrypted, `keystore.DecryptKey` calls `crypto.ToECDSAUnsafe` to convert the plaintext to an ECDSA private key.

```
func DecryptKey(keyjson []byte, auth string) (*Key, error) {
	[...]

	key := crypto.ToECDSAUnsafe(keyBytes)

	return &Key{
		ID:         uuid.UUID(keyID),
		Address:    crypto.PubkeyToAddress(key.PublicKey),
		PrivateKey: key,
	}, nil
}
```

Figure 13.1: The `keystore.DecryptKey` function calls `crypto.ToECDSAUnsafe` to convert the plaintext to an ECDSA private key.

If `toECDSA` returns an error, the `crypto.ToECDSAUnsafe` function will discard it and return `nil`. This will not be detected by `keystore.DecryptKey` and will cause a `nil` pointer exception when the key is dereferenced.

Recommendations
Short term, to prevent leaks of plaintext information, replace `crypto.ToECDSAUnsafe` with `crypto.ToECDSA` and ensure that the function returns a `keystore.ErrDecrypt` error if the call to `crypto.ToECDSA` fails.

14. Undocumented API endpoints

Severity: Informational
Type: Auditing and Logging
Target: `CrossChain-Bridge`

Difficulty: N/A
Finding ID: TOB-ACCB-014

Description
Several API endpoints exposed by the CrossChain-Bridge API are omitted from the RPC README.md document. These methods include `swap.GetSwapStatistics`, `swap.GetRawSwapin`, and `swap.GetLatestScanInfo`. While this deficiency in the documentation does not represent a security risk, it is best to explicitly document the intended purposes and behaviors of all API endpoints. Furthermore, keeping an up-to-date inventory of all API endpoints would help Anyswap track the endpoints that should be deprecated and the functionality that should not be exposed to users.

Recommendations
Short term, document all API endpoints exposed by the CrossChain-Bridge APIs. Additionally, remove any endpoints that are no longer required or should not be exposed to users from the code.

15. Lack of rate-limiting controls

Severity: Medium
Type: Denial of Service
Target: `CrossChain-Bridge, CrossChain-Router`

Difficulty: High
Finding ID: TOB-ACCB-015

Description
Because the CrossChain-Bridge application does not impose rate-limiting controls, users can submit multiple requests each second. Trail of Bits was able to submit 860 requests to the `swap.GetSwapout` RPC endpoint in 30 seconds without encountering any attempts to limit the rate of the requests.

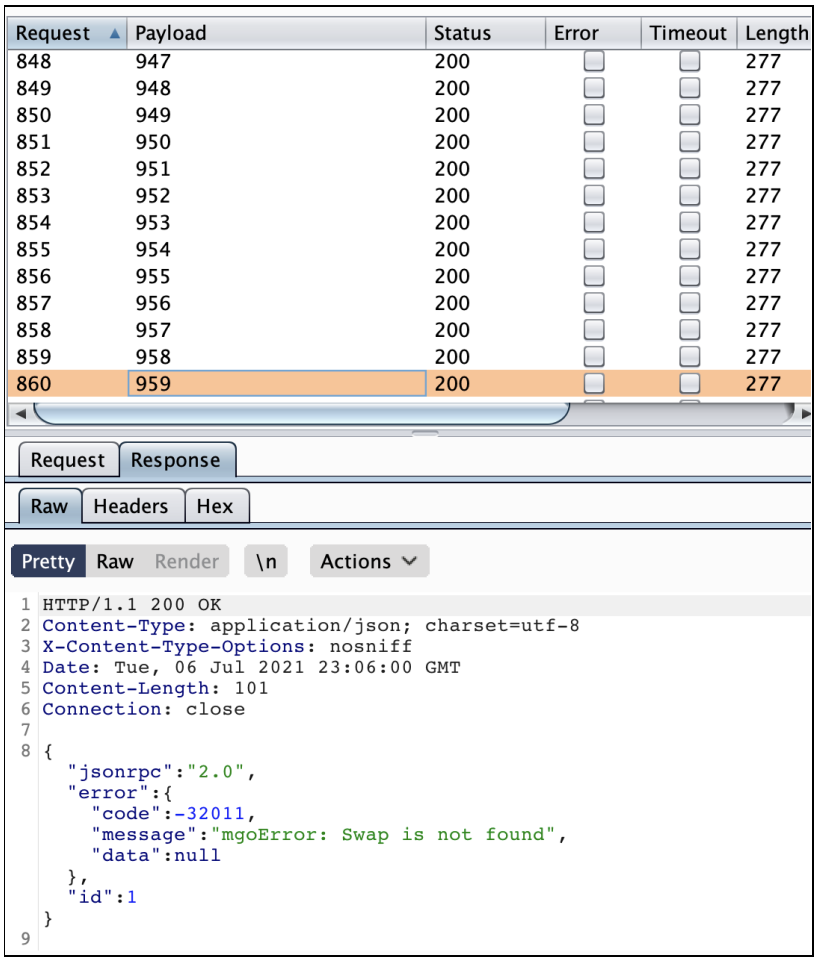

Figure 15.1: The requests sent to the API were not impeded by errors or rate-limiting controls.

Exploit Scenario
An attacker sends multiple requests each second to a process-intensive endpoint using multiple threads. This could result in an exhaustion of resources or could cause services like Infura to block the bridge back end from making additional requests. Either outcome could result in a denial-of-service condition, preventing users from accessing the service.

Recommendations
Short term, consider restricting the rate at which a single IP address can make requests. Alternatively, it may be possible to impose these rate-limiting controls on a given set of blockchain addresses.

16. Support for insecure TLS protocols and weak ciphers

Severity: Medium | Difficulty: High
Type: Cryptography | Finding ID: TOB-ACCB-016
Target: `https://anyswap.exchange`, `https://stable.anyswap.exchange`

Description
The production environments of the CrossChain-Bridge and CrossChain-Router support TLSv1.0 and TLSv1.1, both of which are considered weak transport layer protocols. The applications also support the following weak ciphers:

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

TLSv1.2 was introduced shortly after version 1.1 to address various unresolved security concerns. Popular browsers such as Google Chrome have also deprecated TLSv1.1.

For more information, review the SSL Labs links in the “References” section below.

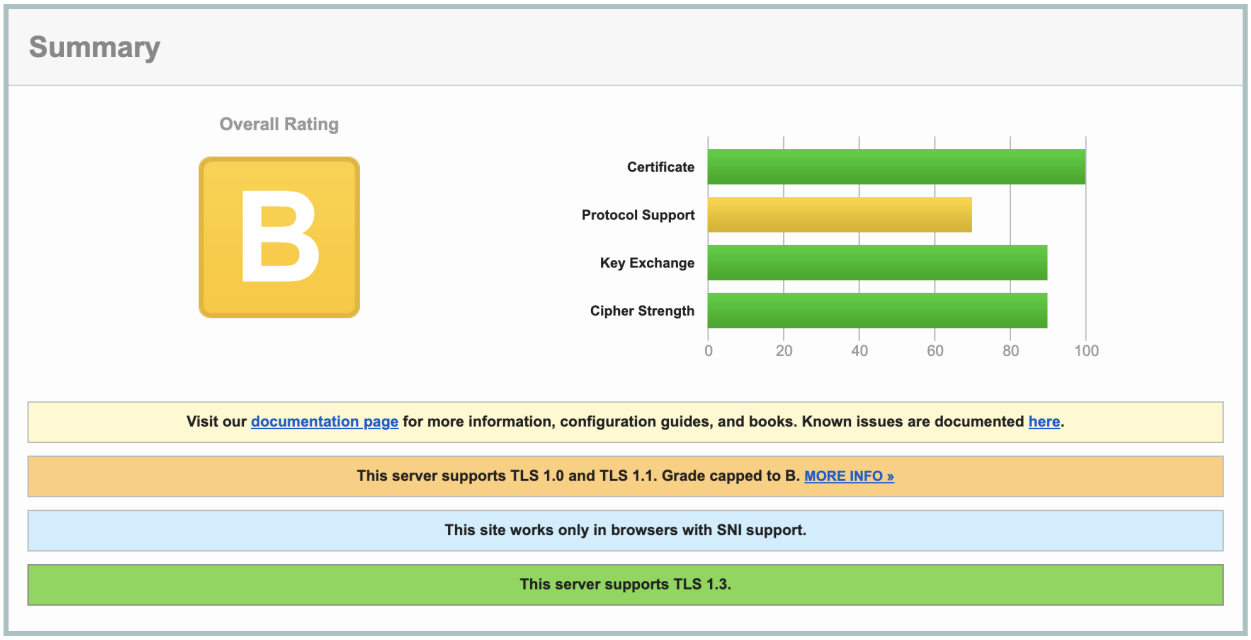

Figure 16.1: Insecure protocol support reported for https://anyswap.exchange

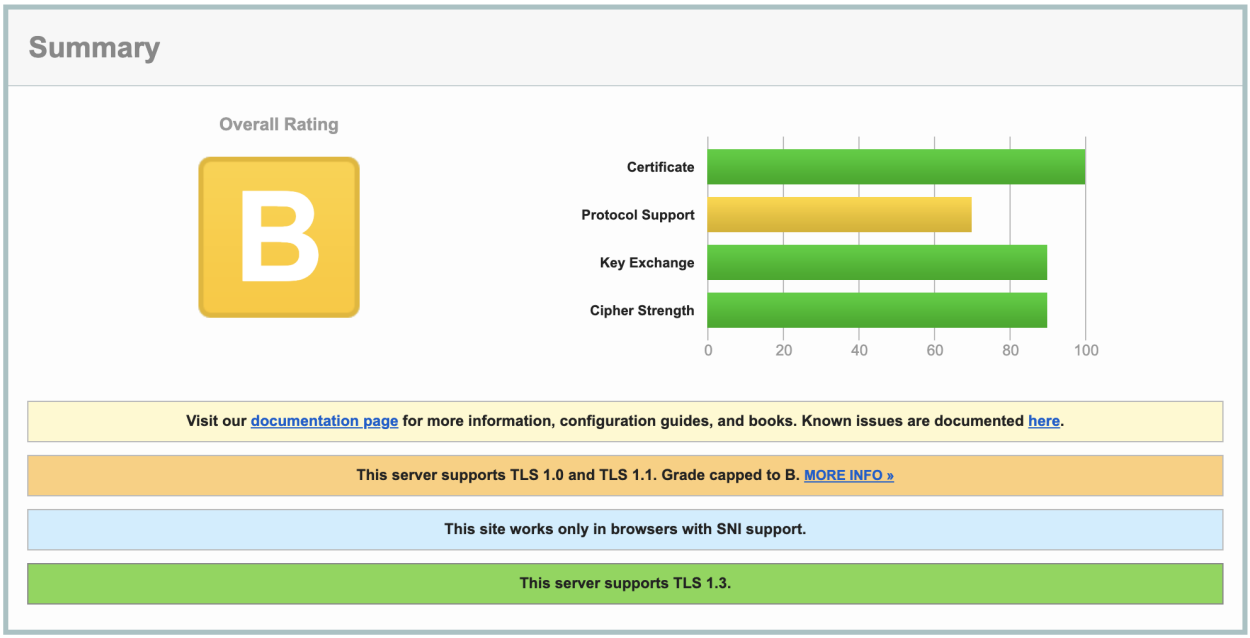

Figure 16.2: Insecure protocol support reported for https://stable.anyswap.exchange

Exploit Scenario
Bob is a user of the Anyswap exchange. Eve, an attacker on Bob's local network, notices that Bob is connecting to https://anyswap.exchange using TLSv1.0. Security weaknesses in TLSv1.0 enable Eve to discern sensitive data sent over the wire by Bob.

Recommendations
Short term, deprecate the use of TLSv1.1 in the production environments of the CrossChain-Bridge and CrossChain-Router (at https://anyswap.exchange and https://stable.anyswap.exchange, respectively).

Long term, review server configurations to ensure all standards are up to date and adhere to best practices.

References
- SSL Labs, "anyswap.exchange"
- SSL Labs, "stable.anyswap.exchange"
- Chrome Status, "Feature: TLS 1.0 and TLS 1.1 (removed)"

17. Lack of HTTP Strict-Transport-Security (HSTS) header

Severity: Low
Difficulty: High
Type: Configuration
Finding ID: TOB-ACCB-017
Target: `https://anyswap.exchange`, `https://stable.anyswap.exchange`

Description

The production environments of the CrossChain-Bridge and CrossChain-Router do not enable HTTP HSTS, a feature that prevents downgrade attacks in which requests are redirected from an HTTPS link to an HTTP link. Enabling HSTS will ensure that HTTP links are converted to HTTPS links and will prevent the use of HTTP. It is best to avoid using HTTP, because it exposes data in transit as cleartext, which can be intercepted or modified via a man-in-the-middle attack.

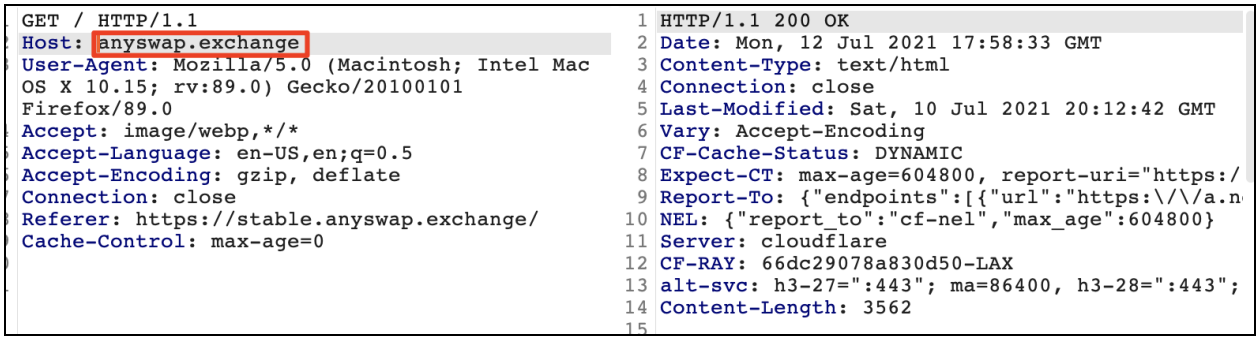

Figure 17.1: There is no HSTS header for https://anyswap.exchange.

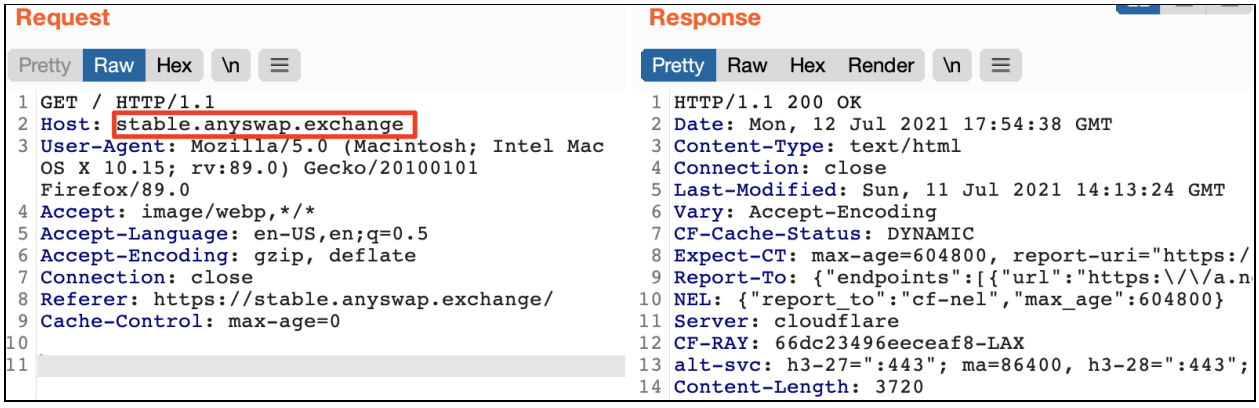

Figure 17.2: There is no HSTS header for https://stable.anyswap.exchange.

Exploit Scenario

A CrossChain-Bridge user connects to an insecure network, such as a public Wi-Fi network. An attacker then intercepts the connection and changes the request, redirecting it to an unencrypted version of the CrossChain-Bridge.

Recommendations

Short term, use the HTTP STS header to prevent the accidental referencing of the HTTP protocol in place of HTTPS.

Long term, be mindful of server configuration best practices and consider using services such as SSL Labs to quickly review information on non-ideal SSL/TLS configurations.

References

- https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

18. Server exposes software version in HTTP headers

Severity: Low
Difficulty: High
Type: Data Exposure
Finding ID: TOB-ACCB-018
Target: `https://anyswap.exchange`, `https://stable.anyswap.exchange`

Description
Server details, including technology and version details, are exposed in HTTP headers by the production deployments of the CrossChain-Bridge and CrossChain-Router, making it easy to fingerprint the server. Malicious actors could use this information to perform version-specific attacks against the server.

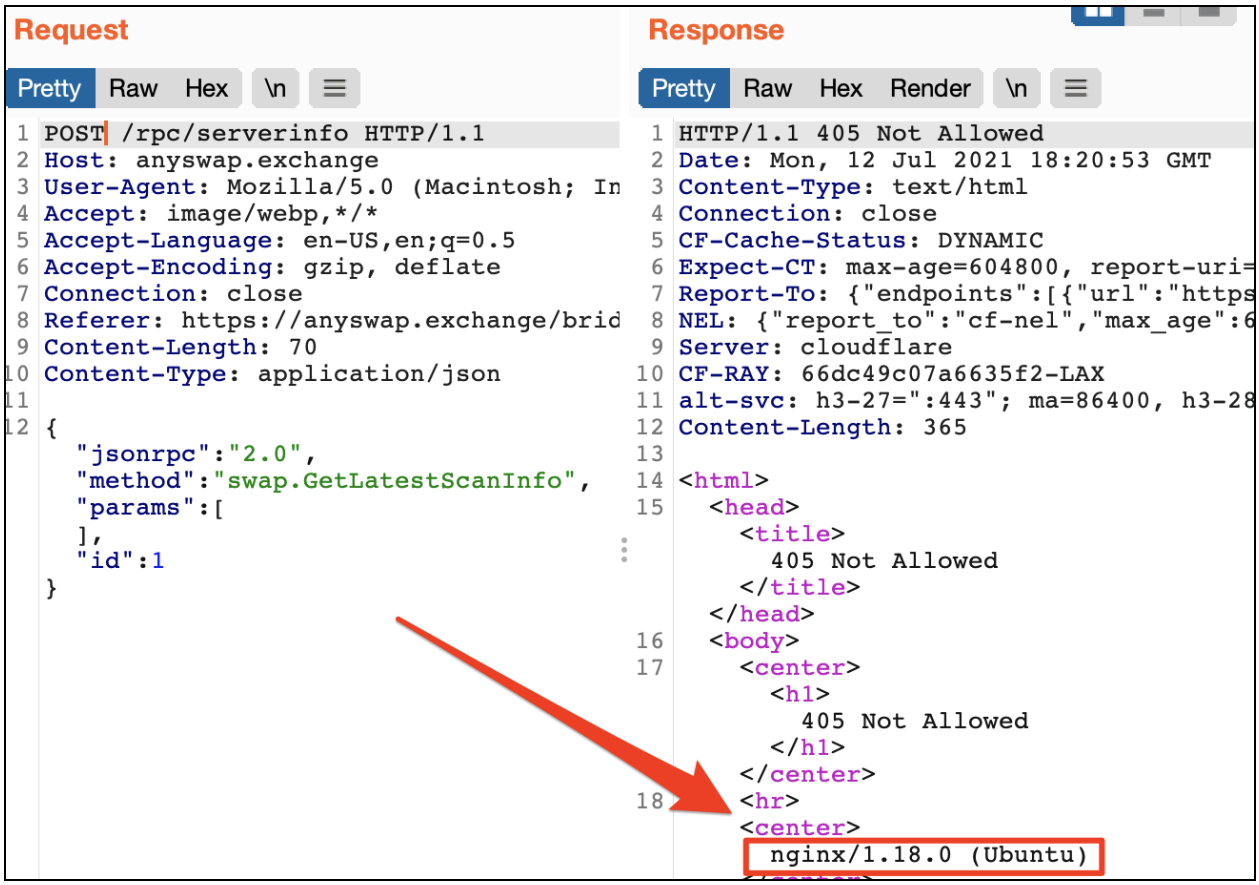

Figure 18.1: A server version displayed in a server response.

Exploit Scenario
An attacker discovers the server's version by examining its banner and is able to identify an effective version-specific exploit.

Recommendations
Short term, configure the server to conceal its version number.

Long term, review the configuration of all server software to prevent the public exposure of sensitive information that could be used to fingerprint the system.

References

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server

19. Risk of invalid curve attacks in MPC nodes

Severity: Medium
Difficulty: Medium
Type: Cryptography
Finding ID: TOB-ACCB-019
Target: `Anyswap-MPCNode/dcrm/sign.go`

Description
Anyswap has implemented a multi-party computation protocol for computing an ECDSA digital signature. It is imperative that signature nonces, especially ECDSA nonces, be computed securely. At a high level, to begin this process, each party generates a random value, `gamma`, and uses scalar multiplication to multiply it by the base elliptic curve point. Each party then makes a cryptographic commitment to the resulting elliptic curve point. Before revealing the elliptic curve point, the parties broadcast their commitments; this forces malicious parties to select an elliptic curve point before seeing the other points. Once the commitments have been broadcast, the elliptic curve points are revealed.

To compute the `r` value of an ECDSA signature, each party adds together these elliptic curve points and scalar-multiplies them by additional random scalar values also generated by the parties. Before the elliptic curve points are added together, the broadcast commitments need to be verified to ensure that the parties were not behaving maliciously. However, this verifies only the integrity of the commitments. There is no check to verify the integrity of the elliptic curve points—that is, to verify that the elliptic curve points are on the `secp256k1` curve. Without this check, the protocol could be vulnerable to an invalid curve attack.

Exploit Scenario
An attacker, Eve, sets up an MPC node and generates elliptic curve points that are on a twist of the `secp256k1` curve. Eve then participates in the protocol, and the parties use her points, which lie on an invalid curve, to jointly compute the ECDSA signature. As a result, Eve is able to extract the ECDSA private key and to steal other users' funds.

Recommendations
Short term, adjust the code responsible for verifying commitments such that it also ensures that each elliptic curve point is on the `secp256k1` curve.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose

	reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components
Arithmetic	Related to the proper use of mathematical operations and semantics
Assembly Use	Related to the use of inline assembly
Centralization	Related to the existence of a single point of failure
Code Stability	Related to the recent frequency of code updates
Upgradeability	Related to contract upgradeability
Function Composition	Related to separation of the logic into functions with clear purposes
Front-Running	Related to resilience against front-running
Key Management	Related to the existence of proper procedures for key generation, distribution, and access
Monitoring	Related to the use of events and monitoring procedures
Specification	Related to the expected codebase documentation
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.)

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed, and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.

Missing	The component was missing.
Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

This appendix contains findings that do not have immediate or obvious security implications.

1. Functions such as `GetSignerChainId` (`tokens/etc/bridge.go`), `VerifyChainId` (`tokens/eth/bridge.go`), and `IsValidAddress` (`tokens/eth/address.go`) should use named constants to identify network and chain IDs. This would make the code easier to audit and maintain.
2. The codebase includes very sparse comments, which can make it difficult to understand its components' interactions and to determine when security invariants must be upheld. We recommend that Anyswap add more comments throughout the codebase detailing the high-level purpose of each package and each function as well as any security invariants that must be upheld.
3. In line 18 of the `tools/sendemail.go` code, which is used to send emails, the host and port are concatenated using `Sprintf`. It would be better to use the `net.JoinHostPort` function, as it supports IPv6 address literals, unlike `Sprintf`.
4. The function `parseErc20EncodedData` (`tokens/eth/verifyerc20tx.go`) validates the size of input after decoding the input. This is not currently an issue, since the input is padded to the correct size, but we would still recommend that the input size be checked before the input is parsed.
5. Several type assertion checks are missing from the code, including from `tools/rlp/decode.go:367` and `worker/accept.go:307`. To find all instances of missing checks, run the `unchecked-type-assertion` rule against the codebase.

C. Fix Log

After the initial assessment, Anyswap informed Trail of Bits that it had addressed issues identified in the audit through various pull requests. The audit team verified each fix to ensure that it would appropriately address the corresponding issue. The results of this audit are provided in the table below.

ID	Title	Severity	Status
1	MPC node safe prime-generation algorithm drops one bit of entropy	Low	Fixed
2	Potential nil pointer dereference in random.GetRandomIntFromZn, random.GetRandomIntFromZnStar, and ec2.Commit	Low	Fixed
3	Appending data from multiple goroutines to one slice is not concurrency-safe	Undetermined	Fixed
4	Insufficient test coverage	Medium	Not fixed
5	Unchecked errors	Low	Fixed
6	Worker reads from the wrong chain configuration when starting transaction pool scans	Informational	Fixed
7	Bridge.getStableReceipt returns the wrong error type if Bridge.getTransactionReceipt fails to reach the remote API endpoint	Informational	Fixed
8	SaveECDSA should verify file permissions if the file exists	Informational	Fixed
9	LoadKeyStore should verify permissions on the key and password files	Medium	Fixed
10	keystore.DecryptKey makes cryptographic choices based on untrusted data	Medium	Fixed
11	Thread deadlocks on utxoLock RWMutex	Informational	Fixed
12	aesCBCDecrypt is vulnerable to CBC padding oracle attacks	Medium	Fixed
13	crypto.ToECDSAUnsafe discards error codes	Informational	Fixed

14	Undocumented API endpoints	Informational	Fixed
15	Lack of rate-limiting controls	Medium	Fixed
16	Support for insecure TLS protocols and weak ciphers	Medium	Fixed
17	Lack of HTTP Strict-Transport-Security (HSTS) header	Low	Fixed
18	Server exposes software version in HTTP headers	Low	Fixed
19	Risk of invalid curve attacks in MPC nodes	Medium	Fixed

For additional information on each fix, please refer to the Detailed Fix Log on the following page.

Detailed Fix Log

Finding 1: MPC node safe prime-generation algorithm drops one bit of entropy
Fixed. The MPC node code now generates primes of `L/2` bits, no longer dropping one bit of entropy. The prime-generation algorithm also checks that the resulting primes are less than `2^(L/2)`.

Finding 2: Potential nil pointer dereference in random.GetRandomIntFromZn, random.GetRandomIntFromZnStar, and ec2.Commit

Fixed. The `random.GetRandomIntFromZn`, `random.GetRandomIntFromZnStar`, and `ec2.Commit` functions now check that the objects returned from `random.GetRandomInt` are not nil.

Finding 3: Appending data from multiple goroutines to one slice is not concurrency-safe

Fixed. There is now a channel in the `dcrm.GetCurNodeSignInfo` function to which each goroutine can write output.

Finding 4: Insufficient test coverage

Not fixed. The unit test coverage remains insufficient.

Finding 5: Unchecked errors

Fixed. All errors caught in `rpc/server/server.go` and `rpc/restapi/api.go` are now handled.

Finding 6: Worker reads from the wrong chain configuration when starting transaction pool scans

Fixed. The worker now reads from the correct chain configuration variable, `dstChainCfg`.

Finding 7: Bridge.getStableReceipt returns the wrong error type if Bridge.getTransactionReceipt fails to reach the remote API endpoint

Fixed. The `GetTransactionStatus` API now returns both a `tokens.TxStatus` structure and an error. In addition, the `GetTransactionReceipt` and `GetTransactionStatus` functions return `tokens.ErrRPCQueryError` errors whenever a call to the remote RPC gateway fails.

Finding 8: SaveECDSA should verify file permissions if the file exists

Fixed. The `SaveECDSA` function was removed, as it was not in use.

Finding 9: LoadKeyStore should verify permissions on the key and password files

Fixed. The `SafeReadFile` function, which verifies that file permissions are set to `0400`, is now used to read key and password files when `LoadKeyStore` is called.

Finding 10: keystore.DecryptKey makes cryptographic choices based on untrusted data

Fixed. Support for version 1 of the algorithm has been removed, so that version can no longer be altered.

Finding 11: Thread deadlocks on utxoLock RWMutex

Fixed. The Anyswap team updated the code so that `defer utxoLock.Unlock()` is called instead of `defer utxoLock.Lock()`.

Finding 12: aesCBCDecrypt is vulnerable to CBC padding oracle attacks

Fixed. Support for version 1 of the protocol, and therefore support for CBC mode, has been removed.

Finding 13: crypto.ToECDSAUnsafe discards error codes

Fixed. Errors resulting from calls to `crypto.ToECDSAUnsafe` are now captured and returned by the `DecryptKey` function.

Finding 14: Undocumented API endpoints

Fixed. The Anyswap team developed documentation on the previously undocumented endpoints.

Finding 15: Lack of rate-limiting controls

Fixed. The tollbooth library is now used to enforce request rate limits on the API server.

Finding 16: Support for insecure TLS protocols and weak ciphers

Fixed. Support for insecure TLS protocols and weak ciphers has been removed from the production environments of the CrossChain-Bridge and CrossChain-Router applications.

Finding 17: Lack of HTTP Strict-Transport-Security (HSTS) header

Fixed. Server responses of both the CrossChain-Bridge and CrossChain-Router applications now include an HSTS header.

Finding 18: Server exposes software version in HTTP headers

Fixed. Server version information is no longer returned in server responses.

Finding 19: Risk of invalid curve attacks in MPC nodes

Fixed. Checks have been added to the commitment verification code to ensure that elliptic curve points are on the curve.