OAuth2 Proxy in Kubernetes

Step by step guide of how to get up and running with oauth2-proxy and securing services in Kubernetes

First of all, before doing anything else, make sure your config is correct by running proxy locally without any kubernetes stuff, e.g. here is example for Azure App Registration:

```
docker run -it --rm --name=oauth2-proxy -p 4180:4180 quay.io/oauth2-pro
    --provider=oidc \
    --email-domain=* \
    --upstream=file:///dev/null \
    --http-address=0.0.0.0:4180 \
    --provider-display-name=azure \
    --client-id=********-****-****-****-************ \
    --client-secret=**_*********-*********************** \
    --redirect-url=http://localhost:4180/oauth2/callback \
    --oidc-issuer-url=https://login.microsoftonline.com/********-****-*
    --cookie-secret=********************************
```

And here is example for GitHub App:

```
docker run -it --rm --name=oauth2-proxy -p 4180:4180 quay.io/oauth2-pro
    --provider=github \
    --email-domain=* \
    --upstream=file:///dev/null \
    --http-address=0.0.0.0:4180 \
    --client-id=******************** \
    --client-secret=**************************************** \
    --redirect-url=http://localhost:4180/oauth2/callback \
    --cookie-secret=********************************
```

Note: for cookie secret use:

```
docker run -ti --rm python:3-alpine python -c 'import secrets,base64; p
```

And just open localhost:4180/oauth2/start

And only after successfull registration and redirection to a home page (which will show you 404 page not found) proceed further. Believe or not it will save you hours.

Nginx Ingress or how it all works

There are two key annotations for an external authentication in nginx ingress:

- `nginx.ingress.kubernetes.io/auth-url` - is an url which will be called to see if current request is authenticated or not
- `nginx.ingress.kubernetes.io/auth-signin` - is an url to which anonymous user will be redirected to authenticate

Before moving further we need to see how it actually works (otherwise it will be black magic)

Suppose we have following app which we want to hide behind authorization (nothing fancy, simple deployment, service and ingress):

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: demo
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app1
  namespace: demo
  labels:
    app: app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
        - name: app1
          image: mendhak/http-https-echo
          ports:
            - name: app1
              containerPort: 80
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: app1
  namespace: demo
spec:
  type: ClusterIP
  selector:
    app: app1
  ports:
    - name: app1
      protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app1
  namespace: demo
spec:
  rules:
    - host: app1.cub.marchenko.net.ua
      http:
        paths:
          - pathType: ImplementationSpecific
            path: /
            backend:
              service:
                name: app1
                port:
                  number: 80
```

For very first experiment we going to use httpbin.org which has endpoints we need

Add following annotations to ingress:

```
nginx.ingress.kubernetes.io/auth-url: https://httpbin.org/status/200
nginx.ingress.kubernetes.io/auth-signin: https://httpbin.org/html
```

Whenever request is comming to our app, underneath nginx will call `https://httpbin.org/status/200` which will return `200 OK`, so ingress will think that user is authenticated and proceed

We can test it:

```
curl -i -s http://app1.cub.marchenko.net.ua | head -n 1
# HTTP/1.1 200 OK
```

Now lets flip that and pretend we are not authenticated:

```
nginx.ingress.kubernetes.io/auth-url: https://httpbin.org/status/401
nginx.ingress.kubernetes.io/auth-signin: https://httpbin.org/html
```

```
curl -i -s http://app1.cub.marchenko.net.ua | grep -E "Location|HTTP"
HTTP/1.1 302 Moved Temporarily
Location: https://httpbin.org/html?rd=http://app1.cub.marchenko.net.ua%
```

As you can see, ingress is trying to redirect us to our fake signin page, also note that `rd` query string parameter added

Lets make some bashify on top on nginx config:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: demo
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: auth
  namespace: demo
data:
  default.conf: |
    server {
      location = /check {
        # if there is no "authorization" cookie we pretend that user is
        if ($cookie_authorization = "") {
          return 401;
        }

        # demo for authorization header
        # if ($http_authorization != "Bearer 123") {
        #   return 401;
        # }
```

```
          # if we land here then "authorization" cookie is present
          add_header Content-Type text/plain;
          return 200 "OK";
        }

        location = /login {
          add_header Set-Cookie "authorization=123;Domain=.cub.marchenko.n
          return 302 http://app1.cub.marchenko.net.ua;

          # https://docs.github.com/en/developers/apps/building-oauth-apps
          # note that we are redirecting back to auth
          # $arg_rd - stands for "rd" query string parameter
          # do not forget to replace "client_id"
          # we are cheating with "state" to pass "rd" query string back to
          # return 302 https://github.com/login/oauth/authorize?client_id=
        }

        # because of "redirect_uri" after successfull login we will be red
        # and because we have passed "rd" query string in "redirect_uri" w
        location = /callback {
          # note domain - we need that so cookie will be available on all
          add_header Set-Cookie "authorization=123;Domain=.cub.marchenko.n
          # $arg_state - stands for "state" query string parameter

          # did not work, variable is encoded and nginx redirect us to roo
          # return 302 $agr_state;
          return 302 http://app1.cub.marchenko.net.ua;
        }

        location = /logout {
          # remove cookie
          add_header Set-Cookie "authorization=;Domain=.cub.marchenko.net
          # idea was to redirect back to app1, which will see that we are
          # but it did not worked out, github remembers our decision and a
          # return 302 http://app1.cub.marchenko.net.ua;
          return 302 http://auth.cub.marchenko.net.ua;
        }

        location / {
          add_header Content-Type text/plain;
          return 200 "Auth Home Page\n";
        }
```

```
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth
  namespace: demo
  labels:
    app: auth
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth
  template:
    metadata:
      labels:
        app: auth
    spec:
      containers:
        - name: auth
          image: nginx:alpine
          ports:
            - name: auth
              containerPort: 80
          volumeMounts:
            - name: auth
              mountPath: /etc/nginx/conf.d/default.conf
              subPath: default.conf
      volumes:
        - name: auth
          configMap:
            name: auth
---
apiVersion: v1
kind: Service
metadata:
  name: auth
  namespace: demo
spec:
  type: ClusterIP
  selector:
    app: auth
  ports:
    - name: auth
      protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: auth
  namespace: demo
spec:
  rules:
    - host: auth.cub.marchenko.net.ua
      http:
        paths:
          - pathType: ImplementationSpecific
            path: /
            backend:
              service:
                name: auth
                port:
                  number: 80
```

Config is self explanatory and commented so lets give it a try:

```
# Check home page
curl auth.cub.marchenko.net.ua
# Auth Home Page

# Check anonymous user, should return 401
curl -s -i auth.cub.marchenko.net.ua/check | head -n 1
# HTTP/1.1 401 Unauthorized

# Check logged int, should return 200
curl -s -i -H "Cookie: authorization=123" auth.cub.marchenko.net.ua/che
# HTTP/1.1 200 OK

# Check login, should return 302
curl -s -i auth.cub.marchenko.net.ua/login?rd=https://mac-blog.org.ua/
# HTTP/1.1 302 Moved Temporarily
# Location: https://github.com/login/oauth/authorize?client_id=********
# Check callback, should return 302
curl -s -i 'http://auth.cub.marchenko.net.ua/callback?code=1234567890&s
# HTTP/1.1 302 Moved Temporarily
# Location: https://mac-blog.org.ua/

# Check logout, should return 302
curl -s -i  -H "Cookie: authorization=123" auth.cub.marchenko.net.ua/lo
# HTTP/1.1 302 Moved Temporarily
# Location: http://app1.cub.marchenko.net.ua
```

So we almost implemented our own auth proxy, the last thing is to change annotations:

```
nginx.ingress.kubernetes.io/auth-url: http://auth.cub.marchenko.net.ua/
nginx.ingress.kubernetes.io/auth-signin: http://auth.cub.marchenko.net.
```

Navigate to yours app1.cub.marchenko.net.ua and you should be redirected to login pages, after successfull login back to callback and back to app. Also you should see your cookie being set.

Technically it is how everything work underneath and is enought to move further, except one bonus point which is good to check right now

There are few other external annotations available which we might be interested in:

- `nginx.ingress.kubernetes.io/auth-signin-redirect-param: redirect_uri` - rename `rd` query string parameter to `redirect_uri`, which will hold url from where we have come to login page
- `nginx.ingress.kubernetes.io/auth-cache-key: $cookie_authorization` - will cache check results based on given cookie
- `nginx.ingress.kubernetes.io/auth-cache-duration: 200 202 401 5m` - how long cache should be valid, `200 202 401 5m` is default value and will cache responses with given status codes for 5 minutes

Try to see `kubectl logs -l app=auth -f` and after logging in refresh app few times, you should not see new requests

How oauth2-proxy is deployed

We gonna need to apply:

- `oauth2-proxy` deployment
- `oauth2-proxy` service

For each of our apps we will apply:

- `app1` deployment
- `app1` service
- `app1-oauth2-proxy` ingress for `/oauth2/*`
- `app1` ingress for `/*`

Note that there is no ingress for proxy, but two ingresses per app, one is usual ingress we all applied many times, and second one is to catch all requests to `/oauth2` and route them to our proxy service

oaut-proxy.yml

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: demo
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oauth2-proxy
  namespace: demo
  labels:
    app: oauth2-proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: oauth2-proxy
  template:
    metadata:
      labels:
        app: oauth2-proxy
    spec:
      containers:
        - name: oauth2-proxy
          image: quay.io/oauth2-proxy/oauth2-proxy:latest
          imagePullPolicy: Always
          args:
            - --provider=oidc
            - --email-domain=*
            - --upstream=file:///dev/null
            - --http-address=0.0.0.0:4180
            - --provider-display-name=azure
            # do not forget to change this
            - --client-id=********-****-****-****-************
            - --client-secret=**_*********-***********************
            - --redirect-url=http://app1.cub.marchenko.net.ua/oauth2/ca
            - --oidc-issuer-url=https://login.microsoftonline.com/*****
            - --cookie-secret=********************************
          ports:
            - containerPort: 4180
              protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: oauth2-proxy
  namespace: demo
  labels:
    app: oauth2-proxy
spec:
  ports:
    - name: http
      port: 4180
      protocol: TCP
      targetPort: 4180
  selector:
    app: oauth2-proxy
```

app2.yml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app2
  namespace: demo
  labels:
    app: app2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app2
  template:
    metadata:
      labels:
        app: app2
    spec:
      containers:
        - name: app2
          image: nginx:alpine
          ports:
            - name: app2
              containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app2
  namespace: demo
spec:
  type: ClusterIP
  selector:
    app: app2
  ports:
    - name: app2
      protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app2
  namespace: demo
  annotations:
    # technically it is same as saying redirect to "/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-url: 'http://$host/oauth2/auth'
    nginx.ingress.kubernetes.io/auth-signin: 'http://$host/oauth2/start
spec:
  rules:
    - host: app2.cub.marchenko.net.ua
      http:
        paths:
          - backend:
              service:
```

```yaml
                name: app2
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app2-oauth2-proxy
  namespace: demo
  annotations:
    # IMPORTANT
    # ---------
    # For Azure if you gonna get id_token it will be too big for nginx
    # Fix for:
    # WARNING: Multiple cookies are required for this session as it exce
    # Which leads to:
    # Error redeeming code during OAuth2 callback: token exchange failed
    nginx.ingress.kubernetes.io/proxy-buffer-size: '8k'
    nginx.ingress.kubernetes.io/proxy-buffers-number: '4'
spec:
  rules:
    - host: app2.cub.marchenko.net.ua
      http:
        paths:
          - path: /oauth2
            pathType: Prefix
            backend:
              service:
                name: oauth2-proxy
                port:
                  number: 4180
```

Note that for everything to work you gonna need https so need to deal with cert manager or cover with cloudflare

Custom Proxy

There is one issue with GitHub integration, because it has only one callback URL we are forced to register new apps for all our services which makes no sense at all (Azure from the other had has as many callbacks as you wish) https://www.callumpember.com/Kubernetes-A-Single-OAuth2-Proxy-For-Multiple-Ingresses/ - really good article describing possible workaround with nginx sidecar

But we are going to go even further and just implement our own proxy (why not, after playing with nginx configs it seems to be not so hard)

So it needs to be a service implementing the same logic as in nginx config, also we gonna need to encrypt cookie

Because I do not want to deal with docker images we gonna put whole service into config map it will be fun

So here is a starting point:

```
const http = require('http')
const https = require('https')
const crypto = require('crypto')
const assert = require('assert')

assert.ok(process.env.CLIENT_ID, 'CLIENT_ID environment variable is mis
assert.ok(process.env.CLIENT_SECRET, 'CLIENT_SECRET environment variable

process.env.ENCRYPTION_KEY = process.env.ENCRYPTION_KEY || crypto.randor
// process.env.COOKIE_DOMAIN = process.env.COOKIE_DOMAIN || null
process.env.COOKIE_MAX_AGE = process.env.COOKIE_MAX_AGE || 60 * 60
process.env.COOKIE_NAME = process.env.COOKIE_NAME || 'oauth3-proxy'
process.env.SCOPE = process.env.SCOPE || 'read:user,user:email'
process.env.PORT = process.env.PORT || 3000
process.env.REDIRECT_URL = process.env.REDIRECT_URL || `http://localhos

function exchange (code) {
  const data = JSON.stringify({
    client_id: process.env.CLIENT_ID,
    client_secret: process.env.CLIENT_SECRET,
    code: code
  })
  console.log(
    `curl -s -i -X POST -H 'Accept: application/json' -H 'Content-Type:
  )
  return new Promise((resolve, reject) => {
    const url = 'https://github.com/login/oauth/access_token'
    const method = 'POST'
    const headers = {
      'Content-Type': 'application/json',
      Accept: 'application/json'
    }
```

```
      const req = https.request(url, { headers, method }, (res) => {
        console.log(`${res.statusCode} ${res.statusMessage}`)
        let data = ''
        res.on('data', (chunk) => (data += chunk))
        res.on('end', () => {
          console.log(data)
          try {
            const json = JSON.parse(data)
            if (res.statusCode < 400) {
              resolve(json.access_token)
            } else {
              reject(json)
            }
          } catch (error) {
            reject(error)
          }
        })
      })
      req.on('error', (error) => {
        console.error(error)
        reject(error)
      })
      req.write(
        JSON.stringify({
          client_id: process.env.CLIENT_ID,
          client_secret: process.env.CLIENT_SECRET,
          code: code
        })
      )
      req.end()
    })
  }

  function encrypt (text) {
    const key = crypto.createHash('sha256').update(process.env.ENCRYPTION_
    const iv = crypto.randomBytes(16) // for AES this is always 16
    const cipher = crypto.createCipheriv('aes-256-cbc', Buffer.from(key),
    const encrypted = Buffer.concat([cipher.update(text), cipher.final()]
    return iv.toString('hex') + ':' + encrypted.toString('hex')
  }

  function decrypt (text) {
    const key = crypto.createHash('sha256').update(process.env.ENCRYPTION_
    const iv = text.split(':').shift()
    const decipher = crypto.createDecipheriv('aes-256-cbc', Buffer.from(ke
    const decrypted = decipher.update(Buffer.from(text.substring(iv.lengtl
    return Buffer.concat([decrypted, decipher.final()]).toString()
  }

  http.ServerResponse.prototype.send = function (status, data) {
    this.writeHead(status, { 'Content-Type': 'text/html' })
    this.write(data)
    this.write('\n')
    this.end()
  }

  http.ServerResponse.prototype.redirect = function (location) {
    this.setHeader('Location', location)
    this.writeHead(302)
    this.end()
  }

  http
    .createServer(async (req, res) => {
      if (req.method !== 'GET') {
        res.send(405, 'Method Not Allowed')
        return
      }

      const path = req.url.split('?').shift()
      if (path === '/') {
        const encrypted = new URLSearchParams(req.headers.cookie?.replace
        if (encrypted) {
          try {
            decrypt(encrypted)
            res.send(200, '<h1>oauth3-proxy</h1><form action="/logout"><i
          } catch (error) {
            console.warn(`Unable to decrypt cookie "${encrypted}"`)
            console.warn(error.name, error.message)
            res.setHeader('Set-Cookie', `${process.env.COOKIE_NAME}=;Max-
            res.send(401, error.message)
          }
        } else {
          res.send(200, '<h1>oauth3-proxy</h1><form action="/login"><inpu
        }
      } else if (path === '/check') {
```

```
      const encrypted = new URLSearchParams(req.headers.cookie?.replace
      if (!encrypted) {
        console.log(`Unauthorized - can not find "${process.env.COOKIE_I
        res.send(401, 'Unauthorized')
        return
      }
      try {
        decrypt(encrypted)
        res.send(200, 'OK')
      } catch (error) {
        console.warn(`Unable to decrypt cookie "${encrypted}"`)
        console.warn(error.name, error.message)
        res.send(401, error.message)
      }
    } else if (path === '/login') {
      const url = new URL('https://github.com/login/oauth/authorize')
      url.searchParams.set('client_id', process.env.CLIENT_ID)
      url.searchParams.set('redirect_uri', process.env.REDIRECT_URL)
      url.searchParams.set('scope', process.env.SCOPE)
      url.searchParams.set('state', new URL(`http://localhost${req.url}`
      res.redirect(url)
    } else if (path === '/callback') {
      const query = new URL(`http://localhost${req.url}`).searchParams
      const code = query.get('code')
      const state = query.get('state') || '/'
      try {
        const accessToken = await exchange(code)
        const encrypted = encrypt(accessToken)
        const domain = process.env.COOKIE_DOMAIN ? `;Domain=${process.er
        res.setHeader(
          'Set-Cookie',
          `${process.env.COOKIE_NAME}=${encrypted};Path=/;Max-Age=${proc
        )
        res.redirect(state)
      } catch (error) {
        res.send(500, JSON.stringify(error, null, 4))
      }
    } else if (path === '/logout') {
      res.setHeader('Set-Cookie', `${process.env.COOKIE_NAME}=;Max-Age=(
      res.redirect('/')
    } else {
      res.send(404, 'Not Found')
    }
```

```
  })
  .listen(process.env.PORT, () => console.log(`Listening: 0.0.0.0:${pro
```

And here is our deployment:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: demo
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: auth
  namespace: demo
data:
  oauth3-proxy.js: |
    const http = require('http')
    const https = require('https')
    const crypto = require('crypto')
    const assert = require('assert')

    assert.ok(process.env.CLIENT_ID, 'CLIENT_ID environment variable is
    assert.ok(process.env.CLIENT_SECRET, 'CLIENT_SECRET environment var

    process.env.ENCRYPTION_KEY = process.env.ENCRYPTION_KEY || crypto.r
    // process.env.COOKIE_DOMAIN = process.env.COOKIE_DOMAIN || null
    process.env.COOKIE_MAX_AGE = process.env.COOKIE_MAX_AGE || 60 * 60
    process.env.COOKIE_NAME = process.env.COOKIE_NAME || 'oauth3-proxy'
    process.env.SCOPE = process.env.SCOPE || 'read:user,user:email'
    process.env.PORT = process.env.PORT || 3000
    process.env.REDIRECT_URL = process.env.REDIRECT_URL || `http://loca

    function exchange(code) {
      const data = JSON.stringify({
        client_id: process.env.CLIENT_ID,
        client_secret: process.env.CLIENT_SECRET,
        code: code
      })
      console.log(
        `curl -s -i -X POST -H 'Accept: application/json' -H 'Content-T
      )
      return new Promise((resolve, reject) => {
        const url = 'https://github.com/login/oauth/access_token'
        const method = 'POST'
        const headers = {
          'Content-Type': 'application/json',
          Accept: 'application/json'
        }
        const req = https.request(url, { headers, method }, (res) => {
          console.log(`${res.statusCode} ${res.statusMessage}`)
          let data = ''
          res.on('data', (chunk) => (data += chunk))
          res.on('end', () => {
            console.log(data)
            try {
              const json = JSON.parse(data)
              if (res.statusCode < 400) {
                resolve(json.access_token)
              } else {
                reject(json)
              }
            } catch (error) {
              reject(error)
            }
          })
        })
        req.on('error', (error) => {
          console.error(error)
          reject(error)
        })
        req.write(
          JSON.stringify({
            client_id: process.env.CLIENT_ID,
            client_secret: process.env.CLIENT_SECRET,
            code: code
          })
        )
        req.end()
      })
    }

    function encrypt(text) {
      const key = crypto.createHash('sha256').update(process.env.ENCRYP
      const iv = crypto.randomBytes(16) // for AES this is always 16
      const cipher = crypto.createCipheriv('aes-256-cbc', Buffer.from(ke
      const encrypted = Buffer.concat([cipher.update(text), cipher.fina
      return iv.toString('hex') + ':' + encrypted.toString('hex')
    }

    function decrypt(text) {
      const key = crypto.createHash('sha256').update(process.env.ENCRYP
      const iv = text.split(':').shift()
      const decipher = crypto.createDecipheriv('aes-256-cbc', Buffer.fr
      const decrypted = decipher.update(Buffer.from(text.substring(iv.le
      return Buffer.concat([decrypted, decipher.final()]).toString()
    }

    http.ServerResponse.prototype.send = function (status, data) {
      this.writeHead(status, { 'Content-Type': 'text/html' })
      this.write(data)
      this.write('\n')
      this.end()
    }

    http.ServerResponse.prototype.redirect = function (location) {
      this.setHeader('Location', location)
      this.writeHead(302)
      this.end()
    }

    http
      .createServer(async (req, res) => {
        if (req.method !== 'GET') {
          res.send(405, 'Method Not Allowed')
          return
        }

        const path = req.url.split('?').shift()
        if (path === '/') {
          const encrypted = new URLSearchParams(req.headers.cookie?.rep
          if (encrypted) {
            try {
              decrypt(encrypted)
              res.send(200, '<h1>oauth3-proxy</h1><form action="/logout
            } catch (error) {
              console.warn(`Unable to decrypt cookie "${encrypted}"`)
```

```
            console.warn(error.name, error.message)
            res.setHeader('Set-Cookie', `${process.env.COOKIE_NAME}=;
            res.send(401, error.message)
          }
        } else {
          res.send(200, '<h1>oauth3-proxy</h1><form action="/login"><
        }
      } else if (path === '/check') {
        const encrypted = new URLSearchParams(req.headers.cookie?.rep
        if (!encrypted) {
          console.log(`Unauthorized - can not find "${process.env.COO
          res.send(401, 'Unauthorized')
          return
        }
        try {
          decrypt(encrypted)
          res.send(200, 'OK')
        } catch (error) {
          console.warn(`Unable to decrypt cookie "${encrypted}"`)
          console.warn(error.name, error.message)
          res.send(401, error.message)
        }
      } else if (path === '/login') {
        const url = new URL('https://github.com/login/oauth/authorize
        url.searchParams.set('client_id', process.env.CLIENT_ID)
        url.searchParams.set('redirect_uri', process.env.REDIRECT_URL
        url.searchParams.set('scope', process.env.SCOPE)
        url.searchParams.set('state', new URL(`http://localhost${req.
        res.redirect(url)
      } else if (path === '/callback') {
        const query = new URL(`http://localhost${req.url}`).searchPar
        const code = query.get('code')
        const state = query.get('state') || '/'
        try {
          const accessToken = await exchange(code)
          const encrypted = encrypt(accessToken)
          const domain = process.env.COOKIE_DOMAIN ? `;Domain=${proce
          res.setHeader(
            'Set-Cookie',
            `${process.env.COOKIE_NAME}=${encrypted};Path=/;Max-Age=$
          )
          res.redirect(state)
        } catch (error) {
              res.send(500, JSON.stringify(error, null, 4))
            }
          } else if (path === '/logout') {
            res.setHeader('Set-Cookie', `${process.env.COOKIE_NAME}=;Max-
            res.redirect('/')
          } else {
            res.send(404, 'Not Found')
          }
        })
        .listen(process.env.PORT, () => console.log(`Listening: 0.0.0.0:$
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth
  namespace: demo
  labels:
    app: auth
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth
  template:
    metadata:
      labels:
        app: auth
    spec:
      containers:
        - name: auth
          image: node:16-alpine
          command:
            - node
          args:
            - /oauth3-proxy.js
          env:
            - name: CLIENT_ID
              value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            - name: CLIENT_SECRET
              value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            - name: ENCRYPTION_KEY
              value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            - name: COOKIE_DOMAIN
```

```
              value: .cub.marchenko.net.ua
            - name: REDIRECT_URL
              value: http://auth.cub.marchenko.net.ua/callback
            - name: PORT
              value: '80'
          ports:
            - name: auth
              containerPort: 80
          volumeMounts:
            - name: auth
              mountPath: /oauth3-proxy.js
              subPath: oauth3-proxy.js
      volumes:
        - name: auth
          configMap:
            name: auth
---
apiVersion: v1
kind: Service
metadata:
  name: auth
  namespace: demo
spec:
  type: ClusterIP
  selector:
    app: auth
  ports:
    - name: auth
      protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: auth
  namespace: demo
spec:
  rules:
    - host: auth.cub.marchenko.net.ua
      http:
        paths:
          - pathType: ImplementationSpecific
            path: /
            backend:
              service:
                name: auth
                port:
                  number: 80
```

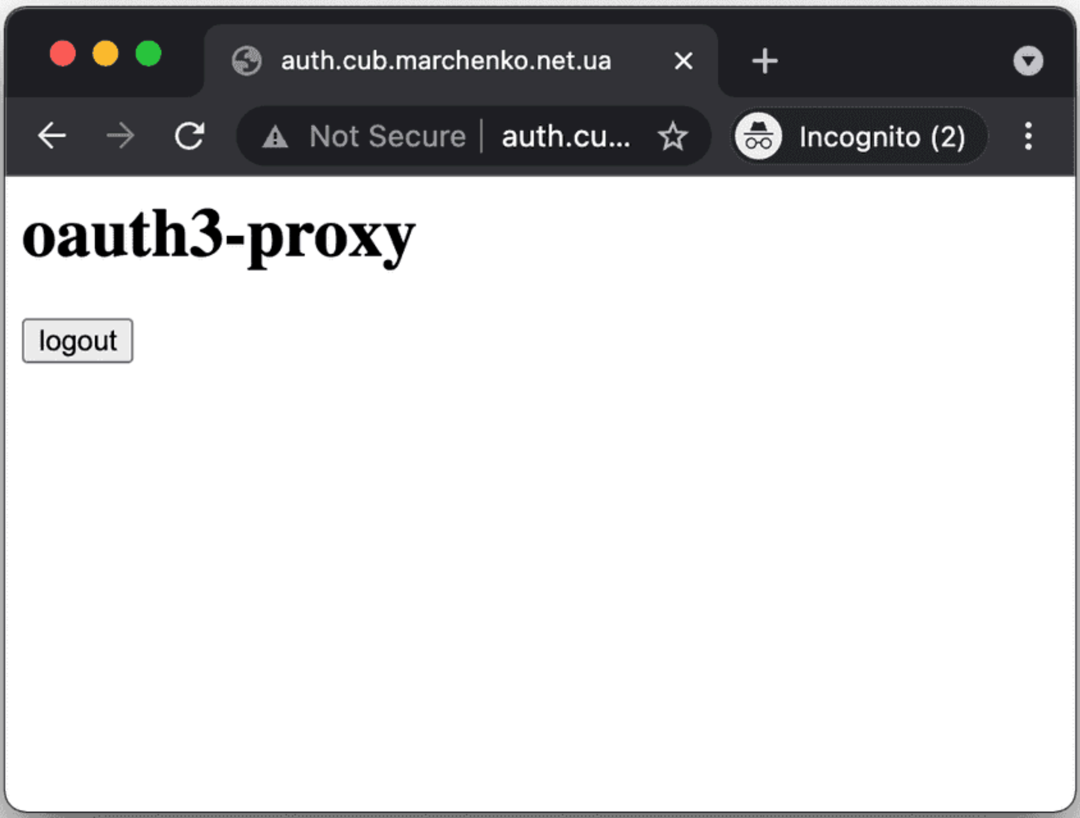
And sample app:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app3
  namespace: demo
  labels:
    app: app3
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app3
  template:
    metadata:
      labels:
        app: app3
    spec:
      containers:
        - name: app3
          image: nginx:alpine
          ports:
            - name: app3
              containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app3
  namespace: demo
spec:
  type: ClusterIP
  selector:
      app: app3
    ports:
      - name: app3
        protocol: TCP
        port: 80
        targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app3
  namespace: demo
  annotations:
    nginx.ingress.kubernetes.io/auth-url: 'http://auth.cub.marchenko.ne
    nginx.ingress.kubernetes.io/auth-signin: 'http://auth.cub.marchenko
spec:
  rules:
    - host: app3.cub.marchenko.net.ua
      http:
        paths:
          - backend:
              service:
                name: app3
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
```

If everything is ok inside proxy logs you will see something like:

```
Listening: 0.0.0.0:80
Unauthorized - can not find "oauth3-proxy" cookie in given cookies "unde
Unauthorized - can not find "oauth3-proxy" cookie in given cookies "unde
curl -s -i -X POST -H 'Accept: application/json' -H 'Content-Type: appl
200 OK
{"access_token":"gho_***********","token_type":"bearer","scope":"read:us
```

And see something like this screenshot on a home page:



oauth3-proxy

Azure Active Directory Proxy

Here is one more example of **aad-proxy**

```
package main

import (
  "context"
  "crypto/rand"
  "encoding/base64"
  "encoding/json"
  "fmt"
  "io"
  "log"
  "net/http"
  "os"
  "time"

  "github.com/coreos/go-oidc/v3/oidc"
  "golang.org/x/oauth2"
```

```
)

func main() {
  clientId := os.Getenv("AAD_CLIEN_ID")
  clientSecret := os.Getenv("AAD_CLIEN_SECRET")
  tenantId := os.Getenv("AAD_TENANT_ID")
  callbackUrl := os.Getenv("AAD_CALLBACK_URL")
  cookieDomain := os.Getenv("AAD_COOKIE_DOMAIN")

  ctx := context.Background()

  provider, err := oidc.NewProvider(ctx, fmt.Sprintf("https://sts.windo
  if err != nil {
    log.Fatal(err)
  }

  verifier := provider.Verifier(&oidc.Config{ClientID: clientId})
  config := oauth2.Config{
    ClientID:     clientId,
    ClientSecret: clientSecret,
    Endpoint:     provider.Endpoint(),
    RedirectURL:  callbackUrl,
    Scopes:       []string{oidc.ScopeOpenID, "profile", "email"},
  }

  http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    cookie, err := r.Cookie("id_token")
    if err != nil {
      log.Println("home handler, unable to retrieve id_token cookie: " 

      // TODO: its not an error - render home page html for anonymous u
      http.Error(w, "Unauthorized", http.StatusUnauthorized)
      return
    }

    idToken, err := verifier.Verify(ctx, cookie.Value)
    if err != nil {
      log.Println("home handler, unable to verify id_token: " + err.Err
      http.Error(w, "Unauthorized", http.StatusUnauthorized)
      return
    }

    // TODO: render html page
		user := User{}
		idToken.Claims(&user)
		data, err := json.Marshal(user)
		if err != nil {
			http.Error(w, err.Error(), http.StatusInternalServerError)
			return
		}
		w.Write(data)
	})

	http.HandleFunc("/check", func(w http.ResponseWriter, r *http.Request
		cookie, err := r.Cookie("id_token")
		if err != nil {
			log.Println("check handler, unable to get id_token cookis: " + er
			http.Error(w, "Unauthorized", http.StatusUnauthorized)
			return
		}

		idToken, err := verifier.Verify(ctx, cookie.Value)
		if err != nil {
			log.Println("check handler, unable to verify id token: " + err.Er
			http.Error(w, "Unauthorized", http.StatusUnauthorized)
			return
		}

		user := User{}
		idToken.Claims(&user)
		log.Println("check handler, success: " + user.Email)

		fmt.Fprintf(w, "OK")
	})

	http.HandleFunc("/login", func(w http.ResponseWriter, r *http.Request
		rd := r.URL.Query().Get("rd")
		if rd == "" {
			rd = "/"
		}

		state, err := randString(16)
		if err != nil {
			log.Println("login handler, unable create state: " + err.Error())
			// TODO: user facing page, need html representation
			http.Error(w, "Internal error", http.StatusInternalServerError)
```

```
      return
    }
    nonce, err := randString(16)
    if err != nil {
      log.Println("login handler, unable create nonce: " + err.Error())
      // TODO: user facing page, need html representation
      http.Error(w, "Internal error", http.StatusInternalServerError)
      return
    }

    ttl := int((5 * time.Minute).Seconds())

    setCallbackCookie(w, r, "rd", rd, cookieDomain, ttl)
    setCallbackCookie(w, r, "state", state, cookieDomain, ttl)
    setCallbackCookie(w, r, "nonce", nonce, cookieDomain, ttl)

    log.Println("login handler, rd: " + rd)

    url := config.AuthCodeURL(state, oidc.Nonce(nonce))
    log.Println("login handler, redirecting to: " + url)
    http.Redirect(w, r, url, http.StatusFound)
  })

  http.HandleFunc("/callback", func(w http.ResponseWriter, r *http.Reque
    state, err := r.Cookie("state")
    if err != nil {
      log.Println("callback handler, unable to get state from cookie: "
      // TODO: user facing page, need html representation
      http.Error(w, "state not found", http.StatusBadRequest)
      return
    }
    if r.URL.Query().Get("state") != state.Value {
      log.Println("callback handler, state from cookie and identity prov
      // TODO: user facing page, need html representation
      http.Error(w, "state did not match", http.StatusBadRequest)
      return
    }

    oauth2Token, err := config.Exchange(ctx, r.URL.Query().Get("code"))
    if err != nil {
      log.Println("callback handler, unable to exchange code for access
      // TODO: user facing page, need html representation
      http.Error(w, "Failed to exchange token: "+err.Error(), http.Statu
      return
    }

    rawIDToken, ok := oauth2Token.Extra("id_token").(string)
    if !ok {
      log.Println("callback handler, unable to get id_token from oauth2
      // TODO: user facing page, need html representation
      http.Error(w, "No id_token field in oauth2 token.", http.StatusIn
      return
    }

    idToken, err := verifier.Verify(ctx, rawIDToken)
    if err != nil {
      log.Println("callback handler, unable to verify id_token: " + err
      // TODO: user facing page, need html representation
      http.Error(w, "Failed to verify ID Token: "+err.Error(), http.Sta
      return
    }

    nonce, err := r.Cookie("nonce")
    if err != nil {
      log.Println("callback handler, unable get nonce from cookie: " +
      // TODO: user facing page, need html representation
      http.Error(w, "nonce not found", http.StatusBadRequest)
      return
    }
    if idToken.Nonce != nonce.Value {
      log.Println("callback handler, nonce in cookie and id_token did n
      // TODO: user facing page, need html representation
      http.Error(w, "nonce did not match", http.StatusBadRequest)
      return
    }

    user := User{}
    idToken.Claims(&user)

    setCallbackCookie(w, r, "id_token", rawIDToken, cookieDomain, int(t

    log.Println("callback handler, successfully logged in " + user.Emai

    rd, err := r.Cookie("rd")
    if err != nil || rd.Value == "" {
      rd.Value = "/"
    }

    http.Redirect(w, r, rd.Value, http.StatusFound)
  })

  http.HandleFunc("/logout", func(w http.ResponseWriter, r *http.Reques
    setCallbackCookie(w, r, "id_token", "", cookieDomain, 0)

    rd := r.URL.Query().Get("rd")
    if rd == "" {
      rd = "/"
    }

    http.Redirect(w, r, rd, http.StatusFound)
  })

  log.Println("listening on http://0.0.0.0:8080")
  log.Fatal(http.ListenAndServe(":8080", nil))
}

type User struct {
  // Id    string   `json:"sub"`
  Name  string   `json:"name"`
  Email string   `json:"unique_name"` // unique_name, upn
  Roles []string `json:"roles"`
}

func randString(nByte int) (string, error) {
  b := make([]byte, nByte)
  if _, err := io.ReadFull(rand.Reader, b); err != nil {
    return "", err
  }
  return base64.RawURLEncoding.EncodeToString(b), nil
}

func setCallbackCookie(w http.ResponseWriter, r *http.Request, name, va
  c := &http.Cookie{
    Name:     name,
    Value:    value,
    Domain:   domain,
    MaxAge:   ttl,
    Secure:   r.TLS != nil,
    HttpOnly: true,
  }
  http.SetCookie(w, c)
}
```

And its demo deployment

```
# TODO: for leanup do not forget to remove mac-temp-2021-11-21-auth and

# namespace, just for demo and easier cleanup
---
apiVersion: v1
kind: Namespace
metadata:
  name: mac

# aad-proxy deployment, service and ingress
# availables at: https://mac-temp-2021-11-21-auth.mac-blog.org.ua/
# endpoints: /          - home page will show if you are logged in or no
#            /login     - will redirect to azure login
#            /callback  - handle login, verify tokens, extract claims, sa
#            /logout    - handle logout, removes cookie and redirect use
#            /check     - internal, used by ingress to decide whether use
# usage:
# after applying aad-proxy just add following annotations to any ingres
#
#   nginx.ingress.kubernetes.io/auth-url: "https://mac-temp-2021-11-21-
#   nginx.ingress.kubernetes.io/auth-signin: "https://mac-temp-2021-11-
#   nginx.ingress.kubernetes.io/auth-cache-key: $cookie_id_token
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aad-proxy
  namespace: mac
  labels:
    app: aad-proxy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aad-proxy
  template:
    metadata:
      labels:
        app: aad-proxy
    spec:
      containers:
        - name: aad-proxy
          image: mac2000/aad-proxy
          env:
            - name: AAD_CLIEN_ID
              value: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
            - name: AAD_CLIEN_SECRET
              value: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            - name: AAD_TENANT_ID
              value: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
            - name: AAD_CALLBACK_URL
              value: https://mac-temp-2021-11-21-auth.mac-blog.org.ua/ca
            - name: AAD_COOKIE_DOMAIN
              value: .mac-blog.org.ua
          ports:
            - name: aad-proxy
              containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: aad-proxy
  namespace: mac
spec:
  type: ClusterIP
  selector:
    app: aad-proxy
  ports:
    - name: aad-proxy
      protocol: TCP
      port: 80
      targetPort: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: aad-proxy
  namespace: mac
  annotations:
    # IMPORTANT - azure gives us really big cookies which wont fit into
    # -----------------------------------------------------------------
    # Fix for: WARNING: Multiple cookies are required for this session a
    # Which leads to: Error redeeming code during OAuth2 callback: toke
    nginx.ingress.kubernetes.io/proxy-buffer-size: "8k"
    nginx.ingress.kubernetes.io/proxy-buffers-number: "4"
spec:
  rules:
    - host: mac-temp-2021-11-21-auth.mac-blog.org.ua
      http:
        paths:
          - pathType: ImplementationSpecific
            path: /
            backend:
              service:
                name: aad-proxy
                port:
                  number: 80
---

# Usage demo, sample app
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app1
  namespace: mac
  labels:
    app: app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
        - name: app1
          image: nginx:alpine
          ports:
            - name: app1
            containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app1
  namespace: mac
spec:
  type: ClusterIP
  selector:
    app: app1
  ports:
    - name: app1
      protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app1
  namespace: mac
  annotations:
    # POI: all we need to do to protect any app
    nginx.ingress.kubernetes.io/auth-url: "https://mac-temp-2021-11-21-
    nginx.ingress.kubernetes.io/auth-signin: "https://mac-temp-2021-11-
    nginx.ingress.kubernetes.io/auth-cache-key: $cookie_id_token
spec:
  rules:
    - host: mac-temp-2021-11-21-app.mac-blog.org.ua
      http:
        paths:
          - pathType: ImplementationSpecific
            path: /
            backend:
              service:
                name: app1
                port:
                  number: 80
```

top home search