173 lines (152 sloc) | 7.09 KB ...

```python
from fido2.client import Fido2Client
from fido2.server import Fido2Server, PublicKeyCredentialRpEntity
from fido2.webauthn import AttestationObject, AuthenticatorData, CollectedClientData
from django.template.context_processors import csrf
from django.views.decorators.csrf import csrf_exempt
from django.shortcuts import render
# from django.template.context import RequestContext
import simplejson
from fido2 import cbor
from django.http import HttpResponse
from django.conf import settings
from .models import *
from fido2.utils import websafe_decode, websafe_encode
from fido2.webauthn import AttestedCredentialData
from .views import login, reset_cookie
import datetime
from .Common import get_redirect_url
from django.utils import timezone


def recheck(request):
    """Starts FIDO2 recheck"""
    context = csrf(request)
    context["mode"] = "recheck"
    request.session["mfa_recheck"] = True
    return render(request, "FIDO2/recheck.html", context)


```

```python
def getServer():
    """Get Server Info from settings and returns a Fido2Server"""
    rp = PublicKeyCredentialRpEntity(id=settings.FIDO_SERVER_ID, name=settings.FIDO_SERVER_NAME)
    return Fido2Server(rp)


def begin_registeration(request):
    """Starts registering a new FIDO Device, called from API"""
    server = getServer()
    registration_data, state = server.register_begin({
        u'id': request.user.username.encode("utf8"),
        u'name': (request.user.first_name + " " + request.user.last_name),
        u'displayName': request.user.username,
    }, getUserCredentials(request.user.username))
    request.session['fido_state'] = state

    return HttpResponse(cbor.encode(registration_data), content_type = 'application/octet-stream')


@csrf_exempt
def complete_reg(request):
    """Completes the registeration, called by API"""
    try:
        data = cbor.decode(request.body)

        client_data = CollectedClientData(data['clientDataJSON'])
        att_obj = AttestationObject((data['attestationObject']))
        server = getServer()
        auth_data = server.register_complete(
            request.session['fido_state'],
            client_data,
            att_obj
        )
        encoded = websafe_encode(auth_data.credential_data)
        uk = User_Keys()
        uk.username = request.user.username
        uk.properties = {"device": encoded, "type": att_obj.fmt, }
        uk.owned_by_enterprise = getattr(settings, "MFA_OWNED_BY_ENTERPRISE", False)
        uk.key_type = "FIDO2"
        uk.save()
        return HttpResponse(simplejson.dumps({'status': 'OK'}))
    except Exception as exp:
        import traceback
        print(traceback.format_exc())
        try:
            from raven.contrib.django.raven_compat.models import client
            client.captureException()
        except:
            pass
```

```
        return HttpResponse(simplejson.dumps({'status': 'ERR', "message": "Error on server, please


def start(request):
    """Start Registeration a new FIDO Token"""
    context = csrf(request)
    context.update(get_redirect_url())
    return render(request, "FIDO2/Add.html", context)


def getUserCredentials(username):
    credentials = []
    for uk in User_Keys.objects.filter(username = username, key_type = "FIDO2"):
        credentials.append(AttestedCredentialData(websafe_decode(uk.properties["device"])))
    return credentials


def auth(request):
    context = csrf(request)
    return render(request, "FIDO2/Auth.html", context)


def authenticate_begin(request):
    server = getServer()
    credentials = getUserCredentials(request.session.get("base_username", request.user.username))
    auth_data, state = server.authenticate_begin(credentials)
    request.session['fido_state'] = state
    return HttpResponse(cbor.encode(auth_data), content_type = "application/octet-stream")


@csrf_exempt
def authenticate_complete(request):
    try:
        credentials = []
        username = request.session.get("base_username", request.user.username)
        server = getServer()
        credentials = getUserCredentials(username)
        data = cbor.decode(request.body)
        credential_id = data['credentialId']
        client_data = CollectedClientData(data['clientDataJSON'])
        auth_data = AuthenticatorData(data['authenticatorData'])
        signature = data['signature']
        try:
            cred = server.authenticate_complete(
                request.session.pop('fido_state'),
                credentials,
                credential_id,
                client_data,
                auth_data,
```

```
                signature
            )
        except ValueError:
            return HttpResponse(simplejson.dumps({'status': "ERR",
                                                  "message": "Wrong challenge received, make sure
                                content_type = "application/json")
        except Exception as excep:
            try:
                from raven.contrib.django.raven_compat.models import client
                client.captureException()
            except:
                pass
            return HttpResponse(simplejson.dumps({'status': "ERR",
                                                  "message": excep.message}),
                                content_type = "application/json")

        if request.session.get("mfa_recheck", False):
            import time
            request.session["mfa"]["rechecked_at"] = time.time()
            return HttpResponse(simplejson.dumps({'status': "OK"}),
                                content_type = "application/json")
        else:
            import random
            keys = User_Keys.objects.filter(username = username, key_type = "FIDO2", enabled = 1)
            for k in keys:
                if AttestedCredentialData(websafe_decode(k.properties["device"])).credential_id ==
                    k.last_used = timezone.now()
                    k.save()
                    mfa = {"verified": True, "method": "FIDO2", 'id': k.id}
                    if getattr(settings, "MFA_RECHECK", False):
                        mfa["next_check"] = datetime.datetime.timestamp((datetime.datetime.now() +
                            seconds = random.randint(settings.MFA_RECHECK_MIN, settings.MFA_RECHEC
                    request.session["mfa"] = mfa
                    try:
                        authenticated = request.user.is_authenticated
                    except:
                        authenticated = request.user.is_authenticated()
                    if not authenticated:
                        res = login(request)
                        if not "location" in res: return reset_cookie(request)
                        return HttpResponse(simplejson.dumps({'status': "OK", "redirect": res["loc
                                            content_type = "application/json")
                    return HttpResponse(simplejson.dumps({'status': "OK"}),
                                        content_type = "application/json")
    except Exception as exp:
        return HttpResponse(simplejson.dumps({'status': "ERR", "message": exp.message}),
                            content_type = "application/json")
```