Passwords are Dead:
WebAuthn for the security of webapps

Chris Volny (she/her)
Cisco :: Duo Security :: CloudSSO Services :: Sr. Software Engineer
About Speaker – Chris Volny

10+ years in infosec

**NORIS (.NET)**
- MFA + TS Credentials
- Reflective Dependency Graph
- XML Transformation / ETL

**GM / OnStar (Java)**
- DataStage / ETL
- SSO EE w/ SAP ERP
- Connected Vehicle Back-office

**VES (C++, Java, Bash)**
- (libvirt) Android Profiles-MDM
- Qt C++ Cross-Domain Bridge
- Yocto Linux + CI

**Duo Security (Python + JS)**
- Cloud SAML/M365/OIDC
- Twisted + Async Python
- JS + React

---

2008 - 2014
- Toledo, OH

2014 - 2017
- Detroit, MI

2017 - 2021
- Ann Arbor, MI

2021 -

10+ years in infosec
Agenda

- Why Passwordless?
- History, Auth Factors, and Cryptography
- FIDO and WebAuthn
- Demo and Usage
- System Design Considerations
- Q/A + Resources
Why Passwordless?

Over 80% of hacking breaches involve brute force or the use of lost or stolen credentials.

Verizon DBIR

70% of breaches were caused by outsiders.
86% of breaches were financially motivated.
43% of breaches were attacks on web application, more than double the results from last year.
27% of malware incidents can be attributed to ransomware.
“I Fight for the User.”

“I’m a user too!”
Passwords: a history

• “something you know”
• Roman Legion – friend from foe
• Defacto computer security since 1960s
  • Fernando Corbató, MIT CTSS
  • ... also first leaks:
    • Spring 1962 printed password file
    • 1966 motd and password files swapped
• We’ve been asking this since 2009:
Password Complexity / Search Space

- Think: time lock
- Choose: random xor memorable
- Rinse, repeat
- www.grc.com/haystack

GRC's Interactive Brute Force Password "Search Space" Calculator
(NOTHING you do here ever leaves your browser. What happens here, stays here.)

Enter and edit your test passwords in the field above while viewing the analysis below.

**Brute Force Search Space Analysis:**

<table>
<thead>
<tr>
<th>Search Space Depth (Alphabet):</th>
<th>26+26+10+33 = 95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Space Length (Characters):</td>
<td>4 characters</td>
</tr>
<tr>
<td>Exact Search Space Size (Count):</td>
<td>82,317,120</td>
</tr>
<tr>
<td>(count of all possible passwords with this alphabet size and up to this password's length)</td>
<td></td>
</tr>
<tr>
<td>Search Space Size (as a power of 10):</td>
<td>8.23 x 10^7</td>
</tr>
</tbody>
</table>

**Time Required to Exhaustively Search this Password's Space:**

| Online Attack Scenario: (Assuming one thousand guesses per second) | 22.87 hours |
| Offline Fast Attack Scenario: (Assuming one hundred billion guesses per second) | 0.000823 seconds |
| Massive Cracking Array Scenario: (Assuming one hundred trillion guesses per second) | 0.000000823 seconds |

Note that typical attacks will be online password guessing limited to, at most, a few hundred guesses per second.
Enter Password Managers

- Impossibly long passwords for everything
- Encrypted with one “strong” password
  - Hope you don’t forget it
  - Hope no one copies it
  - Hope that password was “strong”
Enter Multi-Factor

- Defense in depth
  - “something you know”
  - “something you have/are”
- “Quick, where’s my phone/YubiKey?”
- Variants:
  - Voice/SMS 2FA Hijack / Phishing
  - OTP Codes Exfiltrate / Phishing
  - Push Great - Duo
  - Certificates/PKI Great - WebAuthn
  - Maybe SE/TPM stored Excellent
Enter Single Sign-on

- Social Login
- SAML/OIDC
- Wild West
- Enterprise

Figure 1: Attribute Statement

OpenID Connect Protocol Suite

Underpinnings:
- OAuth 2.0 Core
- OAuth 2.0 Bearer
- OAuth 2.0 Assertions
- OAuth 2.0 JTW Profile
- OAuth 2.0 Responses
- JWT
- JWS
- JWE
- JWKR
- JWKA
- WebFinger

http://openid.net/connect
What are Security Keys?

- We see these a lot in MFA
- What are they?
  - Secure Enclave (SE)
    - Tamper / extraction resistant
    - Asymmetric cryptography
  - Can embed PK Credentials
  - Sometimes built in (platform)
    - Touch ID
  - Sometimes a peripheral (external)
    - YubiKey
  - Can use to unlock bigger vaults
(A)Symmetric Cryptography

- **Symmetric**
  - same key for crypt/decrypt
  - Confidentiality
  - Secures data at rest + transport session
  - DES, AES, Blowfish

- **Asymmetric**
  - One key for crypt, one for decrypt
  - RSA 1977 (Ellis/Cocks ‘73)
  - Confidentiality and Integrity
  - Secures transport negotiation (‘web)
  - RSA, DE, EC
  - SSH, PGP, TLS
FIDO Timeline/Philosophy

• Timeline
  • 2009 – PayPal and Validity Sensor talks
  • 2012 – FIDO Alliance Founded
  • 2014 – Samsung GS5 fingerprint e-shop
  • 2015 – FIDO1 published + BT/NFC
  • 2018 – FIDO2 published  CTAP/WebAuthn
  • 2019 – Wide platform adoption

• Philosophy
  • Strong crypto
  • Limited scope  Think: cookies + domain
  • Device attestation  Which devices to trust
FIDO2 = CTAP + WebAuthn

- **Client-to-Authenticator Protocols (CTAP)**
  - Hardware to OS API / Transport
  - System calls  Windows Hello, libfido2
  - USB, NFC, BLE, TPM Authenticators

- **Web Authentication API (WebAuthn)**
  - Just web applications
  - JavaScript API in browsers
  - Server-side libraries
FIDO2 = CTAP + WebAuthn

- Web Authentication API (WebAuthn)
  - JavaScript API in browsers

```javascript
const credential = await navigator.credentials.create({
  publicKey: publicKeyCredentialCreationOptions
});

const credential = await navigator.credentials.get({
  publicKey: publicKeyCredentialRequestOptions
});
```
Registration

Verify your identity with webauthn.io
Pick an option

USB security key
This device

Register

REGISTRATION BEGINS

USER APPROVAL

USER APPROVAL

SCAN NOW

REGISTRATION COMPLETE

KEY REGISTERED

NEW KEY CREATED

Using PUBLIC KEY CRYPTOGRAPHY
Login

Verify your identity

Confirm your fingerprint so webauthndemo.appspot.com can verify it's you.

Touch sensor

Cancel  Use screen lock

1. LOGIN

2. LOGIN CHALLENGE

3. KEY SELECTED

4. LOGIN COMPLETE

Using PUBLIC KEY CRYPTOGRAPHY
Demonstrations

• https://webauthn.io (Duo Labs) (Android)

• https://www.thevolny.net/ (me) (OS X)

Welcome

This is a homepage for thevolny.net, a work in progress.

This is rendered markdown, served via Django, and loaded into a react UI.

User authentication is strictly passwordless via restauthn and webauthn-client (see github).
See my past or upcoming Passwordless talks on how those work :-)
My Demo – High Level

- **Two Pieces:**
  - JS Library
    - Axios with CBOR Interceptors
  - Login, Register functions
  - Django Module + App
    - Django REST Framework
    - CBOR render/parsers (some base64)
    - Authenticator, LoginToken models
  - ApiViews
  - Really simple templates for testing
My Demo – JS Axios

```javascript
export const axios_cbor = axios.create();
axios_cbor.defaults.xsrfHeaderName = "X-CSRFToken";
axios_cbor.defaults.xsrfCookieName = "csrftoken";
axios_cbor.defaults.withCredentials = true;
axios_cbor.defaults.headers['content-type'] = 'application/cbor';
axios_cbor.defaults.method = 'POST';
axios_cbor.interceptors.request.use(cborRequestInterceptor);
axios_cbor.interceptors.response.use(cborResponseInterceptor);

/**
 * cborRequestInterceptor
 *
 * perform CBOR encoding and set responseType to arraybuffer
 * on outbound requests with content-type application/cbor.
 **/
export const cborRequestInterceptor = async function (request) {
  if (request.headers['content-type'] === 'application/cbor') {
    request.data = await encodeAsync(request.data);
    request.responseType = "arraybuffer"
    return request;
  }
  return request;
};

/**
 * cborResponseInterceptor
 *
 * perform CBOR decoding on inbound responses with
 * content-type application/cbor.
 **/
export const cborResponseInterceptor = async function (response) {
  if (response.headers['content-type'] === 'application/cbor') {
    const [data] = await decodeAll(Buffer.from(response.data));
    response.data = data;
    return response;
  }
  return response;
};
```
My Demo – JS Login

```javascript
function login(payload, setmessage) {
    const success_callback = (res) => {
        console.log('webauthn-login successful.', res);
        setData({...data, user: res.data.user})
        setmessage('Logged in successfully!', 'success')
        setTimeout() => setmessage('', '', 1000)
        setShowLogin(false)
        window.localStorage.setItem('username', res.data.user.username)
    };
    const failure_callback = (error, code) => {
        console.log('webauthn login failed.', error, code);
        setmessage('Failed authentication', 'danger')
    };
    webauthn_login(payload, success_callback, failure_callback);
}
```

```javascript
export const webauthn_login = (payload,
    success_callback,
    failure_callback = (error, code) => {
        console.log('webauthn-login failed:
        beginurl = '/api/auth/login/begin/
        compleuturl = '/api/auth/login/
        ax = axios_cber
    })
    const credentials_callback = (opts) => navigator.credentials.get(opts);
    const complete_payload_callback = (payload, assertion) => {
        return {
            "webauthn": new Uint8Array(assertion.rawId),
            "authenticlientData": new Uint8Array(assertion.response.clientDataJSON),
            "signature": new Uint8Array(assertion.response.signature),
        };
    }
    return webauthn_internal(payload, success_callback, failure_callback, credentials_callback,
        complete_payload_callback, beginurll, compleuturl, WEBAUTHN_LOGIN_FAIL_BEGIEN, WEBAUTHN_LOGIN_FAIL_COMPLETE, ax);
```

```javascript
export const webauthn_internal = (payload, success_callback, failure_callback,
    credentials_callback, complete_context_callback, beginurl, compleuturl, begin_failure_code, complete_failure_code, ax) => {
    console.log('webauthn:', {begin': beginurl, 'complete': compleuturl});
    ax.post(beginurl, payload)
    .then(res => res.data)
    .then(opts => credentials_callback(opts))
    .then(auth => {
        ax.post(compleuturl, complete_context_callback(payload, auth)
        .then(res => success_callback(res))
        .catch(error => failure_callback(error, complete_failure_code));
    }).catch(error => failure_callback(error, begin_failure_code));
```
class LoginToken(models.Model):
    token = models.CharField(('Token'), max_length=64, primary_key=True)
    user = models.ForeignKey(get_user_model(), related_name="tokens", on_delete=models.CASCADE)
    created = models.DateTimeField(('Created'), auto_now_add=True)
    expires = models.DateTimeField(('Expires'))

    class Meta:
        verbose_name = ('Token')
        verbose_name_plural = ('Tokens')

@property
def expired(self):
    return timezone.now() > self.expires

def redeem(self):
    if not self.expired:
        self.delete()
    return self.user
return False

def generate_token(self):
    return tokens.default_token_generator.make_token(self.user)

def renew(self):
    self.expires = timezone.now() + timezone.timedelta(minutes=EXPIRY)
    self.redeemed = None

def save(self, **kwargs):
    if not self.expires:
        self.renew()
    if not self.token:
        self.token = self.generate_token()
    return super(LoginToken, self).save(**kwargs)

def __str__(self):
    return f"{self.user.username}: {self.created}"


class Authenticator(models.Model):
    user = models.ForeignKey(get_user_model(), related_name="authenticators", on_delete=models.CASCADE)
    name = models.CharField(('Nickname'), max_length=180)
    created = models.DateTimeField(('Created'), auto_now_add=True)
    cred_id = models.TextField(unique=True)
    cred_data = models.TextField()
    counter = models.PositiveIntegerField(default=1)

    class Meta:
        verbose_name = ('Authenticator')
        verbose_name_plural = ('Authenticators')
        unique_together = ('user', 'name',)

    def inc_counter(self):
        self.counter += 1
    self.save()
    return self

@property
def cidr(self):
    return websafe_decode(self.cred_id)

@property
def credential(self):
    return AttestedCredentialData(websafe_decode(self.cred_data))

@credential.setter
def credential(self, cred):
    self.cred_data = websafe_encode(cred)
    self.cred_id = websafe_encode(cred.credential_id)

def __str__(self):
    return f"{self.user.username}: {md5(self.crid).hexdigest()} ([self.counter])"
class CborRenderer(BaseRenderer):
    media_type = "application/cbor"
    format = "cbor"
    charset = None
    render_style = "binary"

    def render(self, data, *args, **kwargs):
        return cbor2.dumps(data)

class Base64CborRenderer(BaseRenderer):
    media_type = "text/plain"
    format = "txt"
    charset = "utf-8"

    def render(self, data, *args, **kwargs):
        return base64.b64encode(cbor2.dumps(data))

class Base64JsonRenderer(JSONRenderer):
    def render(self, data, *args, **kwargs):
        return super().render(r Encode(data), *args, **kwargs)

class CborBrowsableAPIRenderer(BrowsableAPIRenderer):
    def get_default_renderer(self, view):
        return Base64JsonRenderer()

class CborParser(BaseParser):
    media_type = "application/cbor"
    renderer_class = CborRenderer

    def parse(self, stream, *args, **kwargs):
        return cbor2.load(stream)

class Base64CborParser(BaseParser):
    media_type = "text/plain"
    renderer_class = Base64CborRenderer

    def parse(self, stream, *args, **kwargs):
        data = base64.b64decode(stream.read())
        return cbor2.loads(data)
My Demo – Login Begin

```python
class WebauthnLoginBegin(BaseWebauthnLoginView):
    
    Webauthn Login Begin View

    Given an anonymous user, extract authentication data from request, use it to authenticate
    and generate a webauthn challenge, store state in session, and return challenge as response.

    def post(self, request, format=None):
        logger.warn('webauthn-login-begin: format={:}. request.data.keys():{}'.format)
        if request.user.is_authenticated:
            return Response(dict(detail="Already authenticated"), status=status.HTTP_401_UNAUTHORIZED)
            authargs = {k: v for k, v in request.data.items() if k in settings.FIDO2.LOGIN_FIDO2_ARGS}
            if authargs:
                user = auth.authenticate(request, passwordless=True, **authargs)
                if user:
                    credentials = [x.credential for x in user.userauthenticators.all()]
                    if credentials:
                        data, state = SERVER.authenticate_begin(credentials, user_verification=settings.FIDO2.USER_VERIFICATION)
                        request.session[settings.FIDO2.SESSION_STATE_KEY] = state
                        return Response(data)
                    logger.warn('No authenticators registered for user')
                else:
                    logger.warn('Bad payload: {}'.format(request.data))
            return Response(dict(detail="No Authentication challenge generated"), status=status.HTTP_400_BAD_REQUEST)
```
My Demo – Login Complete

- Gives:
  - credentialId
  - clientDataJSON
  - authenticatorData
  - Signature

- Gets:
  - Detail + UserInfo
    - username
    - full_name
    - is_staff

```python
class WebauthnLogin(BaseWebauthnLoginView):
    ...
    Webauthn Login Complete View

    Given anonymous user, state from login-begin in session, and the client’s response, complete the login
    virtual, and if valid, log the user in.
    
    def post(self, request, format=None):
        logger.info(f'webauthn-login.({format}): {request.data}'))
        if request.user.is_authenticated:
            return Response(dict(detail='Already authenticated'), status=status.HTTP_401_UNAUTHORIZED)
        authargs = {k: v for k, v in request.data.items()} if k in settings.FIDO2_LOGIN_FIELDS
        try:
            user = auth.authenticate(request, passwordless=True, **authargs)
        except:
            user = None
        if user:
            state = request.session.get('FIDO2_SESSION_STATE_KEY')
            cred_id = request.data.get('credentialId', None)
            client_json = request.data.get('clientDataJSON', None)
            auth_value = request.data.get('authenticatorData', None)
            signature = request.data.get('signature', None)
            if client_json and auth_value and signature:
                client_data = ClientData(client_json)
                auth_data = AuthenticatorData(auth_value)
                credentials = [x.credential for x in user.authenticators.all()]
                if state and credentials and cred_id and client_data and auth_data and signature:
                    try:
                        if SERVER.authenticate_complete(state, credentials, cred_id, client_data, auth_data, signature):
                            auth.login(request, user)
                        return Response(dict(detail='OK', user=user))
                    except Exception as e:
                        logger.error('Exception in webauthn-login.({}): {}'.format(e))
                        return Response(dict(detail='Bad request'), status=status.HTTP_400_BAD_REQUEST)
            return Response(dict(detail='Bad username'), status=status.HTTP_401_UNAUTHORIZED)
    ```
Design Considerations

• Single (passwordless) or Multifactor?
  • What’s your env’s posture?
  • Adaptive?
  • Username-less?

• Requirement Parameters
  • Authenticator type
    • Platform
    • Cross-platform
  • User Verification
    • Warm body, pin, biometric?
  • Attestation Level
    • Identity vs privacy

Security Policies for Every Situation

Get granular about who can access what and when. Duo lets you create custom access policies based on role, device, location, and many other contextual factors.

§ 4. Terminology

Attestation

Generally, attestation is a statement that serves to bear witness, confirm, or authenticate. In the WebAuthn context, attestation is employed to provide verifiable evidence as to the origin of an authenticator and the data it emits. This includes such things as credential IDs, credential key pairs, signature counters, etc.
Additional Notes

- Hybrid Password/Passwordless?
  - Challenge for username enumeration
- Do not roll your own crypto/security
- CBOR vs Base64
- Django Views/API to React = awkward
  - JSON blobs?
Questions?

- WebAuthn 101 [https://webauthn.guide/](https://webauthn.guide/)
- Duo WebAuthn Demo [https://webauthn.io/](https://webauthn.io/)
- FIDO Alliance [https://fidoalliance.org/](https://fidoalliance.org/)
- Django Extension [https://github.com/cvolny/django-restauthn/](https://github.com/cvolny/django-restauthn/)