August 2023

Azure B2C 0-Day

An Exploit Chain from Public Keys to Microsoft Bug Bounty

praetorian

whoami

- John Novak
- Technical Director with praetorian
- Focus on IoT, mobile, web, *cloud* security assessments
- Background & interest in math, crypto

Crypto Proof of Work

- ID token for msrcweb.b2clogin.com →
- Verify Token with key @ https://msrcweb.b2clogin.com/9f449d34-93e9-437f-8082-7127fc8ab474/discovery/v2.0/keys?p=b2c_1a_multitenant_signupsignin
- (Previously) valid on API endpoints like: https://api.msrc.microsoft.com/portal/v2.0/vulnerabilityReport
- Could enumerate Microsoft BB submissions using only the email address of a victim

```
from jwcrypto import jwt, jwk
tok="eyJ0..." (QR code)
k={"kid":"kco…"} (from OpenID Key link)
key=jwk.JWK(**k)
jwt.JWT(key=key,jwt=tok,check_claims=False)
```

praetorian

Background Info

Breadcrumbs to Crypto Misuse
Side-Channel Attack
MSRC as a Target
Recommendations & Disclosure

Symmetric Encryption

- Algorithms: AES, Salsa20, etc.
- Block cipher or Stream cipher
- Block cipher modes – ECB, CBC, CTR, etc.
- Encrypt & decrypt with same secret key
  - $E_{secret}(plain) = cipher$
  - $D_{secret}(cipher) = plain$
- Pros: no size constraints, fast, provides confidentiality
- Cons: requires key exchange/negotiation before use, some integrity

Asymmetric Encryption

- Algorithms: RSA, ECC
- Encrypt with public key, decrypt with private key
  - $E_{public}(plain) = cipher$
  - $D_{private}(cipher) = plain$
- Pros: key negotiation not required, provides confidentiality
- Cons: size constrained, slow, no integrity
- Examples: email, GPG

JSON Web Token: JSON Web Signature (JWS)

- Elements: JOSE Header, payload, signature
- Example:

```
eyJhbGciOiJIUzI1NiIsImtpZCI6ImpYaHVEZUlsWkZqaWVrdHcifQ
.
eyJhZG1pbiI6ZmFsc2UsImVtYWlsIjrigJ1oZWxsby5kZWZjb25AcHJhZXRvcmlhbi5jb20iLCJleHAiOjE2NjU2ODA3NzQsI
mlhdCI6MTY2NTY3OTg3NH0
.
N-eG7un4SyoFVPYX5XF6RJ7h2HaJlOQWArXZcjifmqQ
```

- Decoded:

```
{"alg":"HS256","kid":"jXhuDeIlZFjiektw”}
.
{"admin":false,"email":”hello.defcon@praetorian.com","exp":1665680774,"iat":1665679874}
.
37 e7 86 ee e9 f8 4b 2a 05 54 f6 17 e5 71 7a 44 9e e1 d8 76 89 94 e4 16 02 b5 d9 72 38 9f 9a a4
```

- Integrity ✅, Confidentiality ❌

praetorian

JSON Web Encryption (JWE)

- Elements: JOSE Header, cek, iv, ciphertext, authentication tag
- Example:

```
eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2R0NNIiwia2lkIjoiTGtJRWFQeUJKSUxydmtzUiJ9 .
5UrukUmsT0mgFkVbxwDhM9tj3CMQCrF8rU8C1M8gUUEKe7jwUAWeVA .
VbH3oU7UhuM66n1j .
aTKQTfNXwKqgE7PASxbxv70BIEz6OVmbxDg5m5yu2TMIz8t9vi01nyZixlqV2k6XUF8eZxvJTl7dYvRtUXE-
FkAbj5HD533HGC3v9hLJS3v0CRVgpw .
x0tRgXulK9TZUGpTF0zxUQ
```

- Decoded:

```
{"alg":"A256KW","enc":"A256GCM","kid":"LkIEaPyBJILrvksR"} ...
```

- Integrity 🤷 (✅ / ❌), Confidentiality ✅

praetorian

OAuth 2.0: Authorization Code Flow

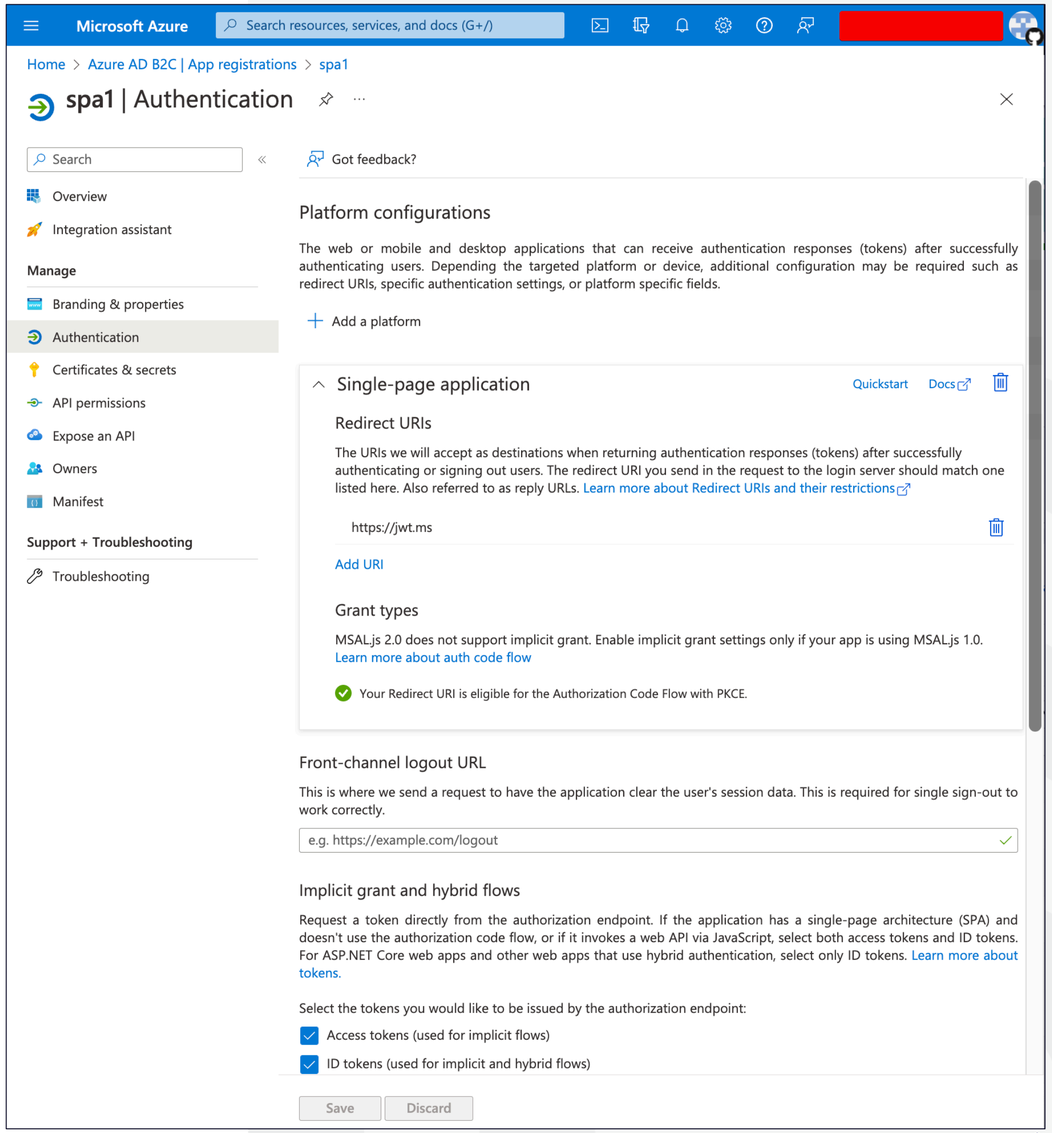
- User authenticates, receives an authorization code (in URI fragment)
- User exchanges authorization code for access token & **refresh token**
- User exchanges refresh token for a new access token & **refresh token**
- Hybrid flow returns id token (not access token) as well as authorization code

Azure Active Directory B2C

- ”Business to Consumer” – B2C
- Offloads authentication & session management from business developed apps
- Configurable login flows
- Integrates with social accounts (Twitter, Facebook, Google) or any SAML provider
- Microsoft has extensive documentation not repeated here

Applications

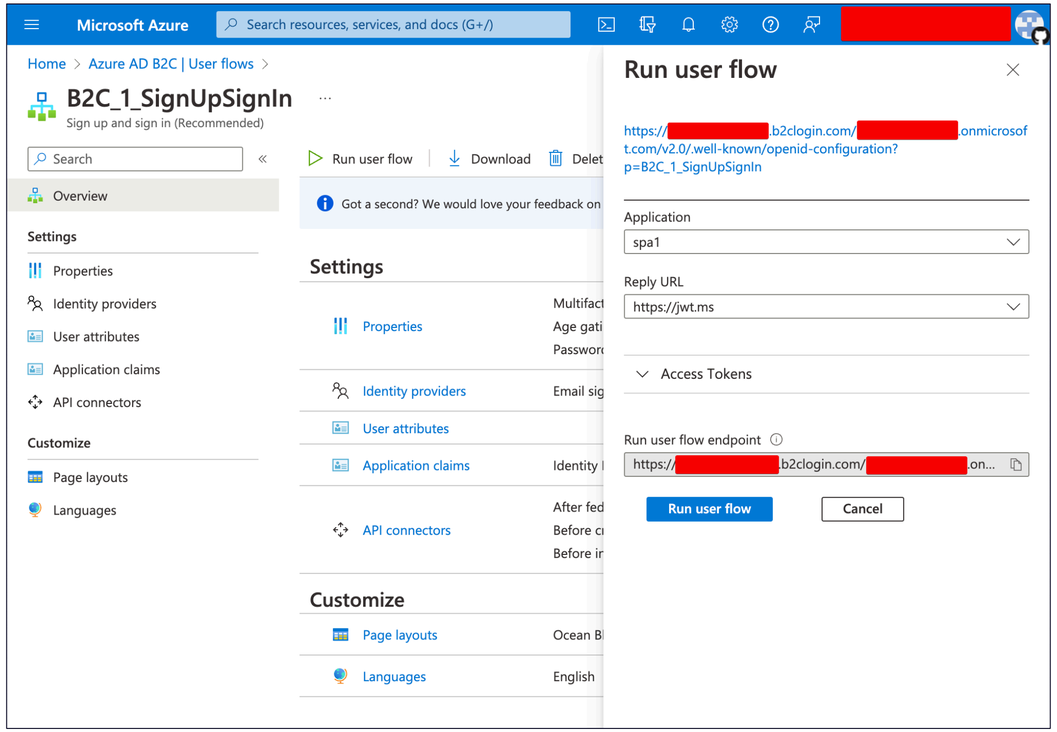
- Web applications, Single Page Applications (SPA), or native applications (mobile & desktop)
- Each has a unique client ID
- Chosen login flows – Implicit Flow, Authorization Code Flow, … with PKCE
- Choose redirect URIs

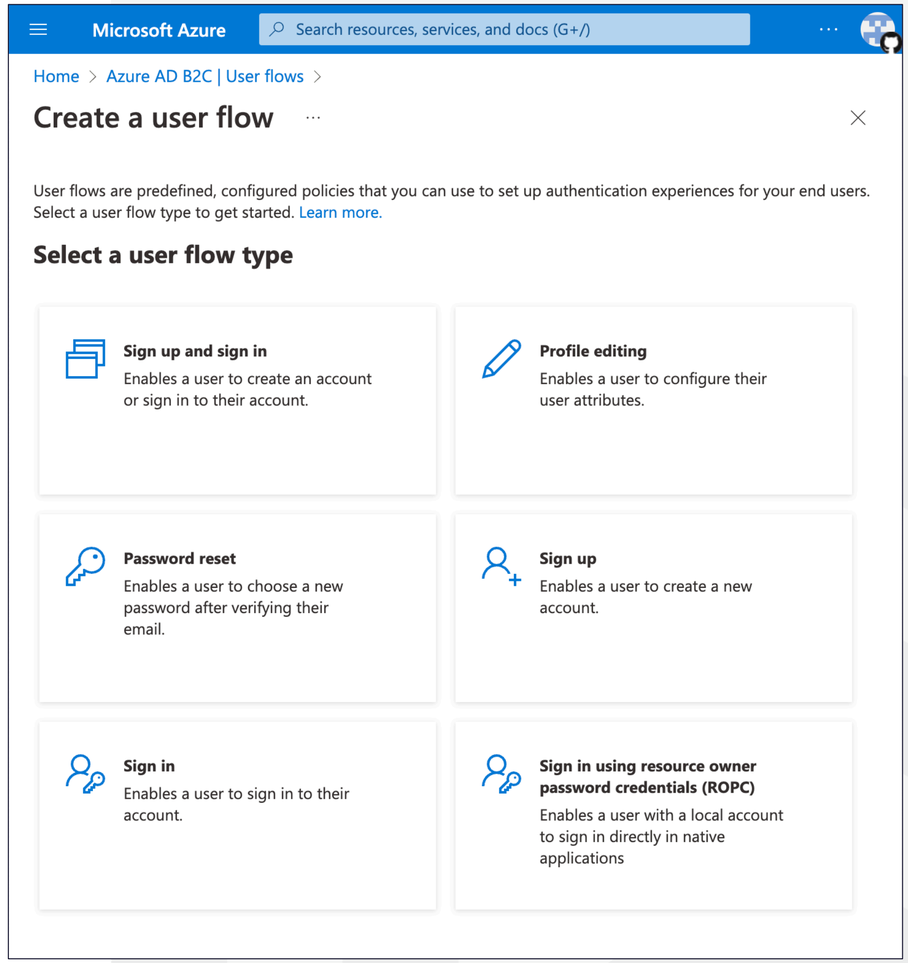


praetorian

User Flows

- Create basic user flows
- Run with a configured application
- Prefixed with “B2C_1_”

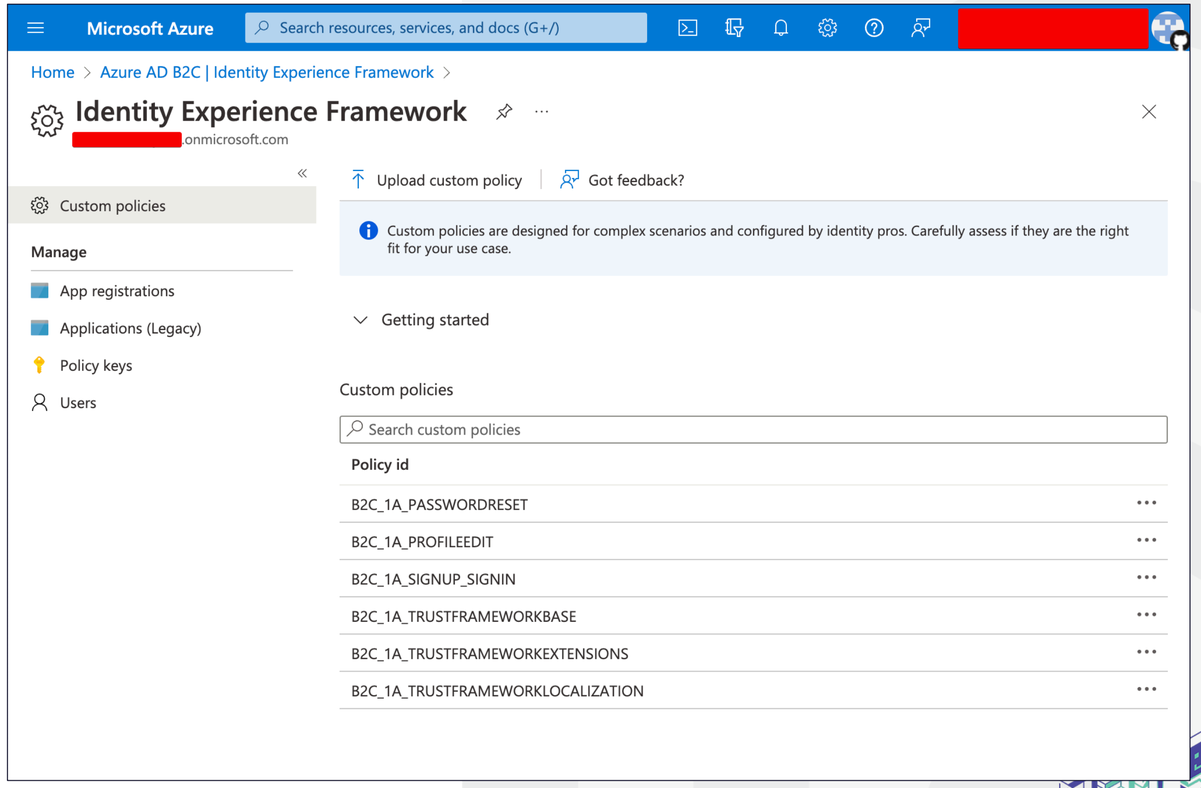




praetorian

Identity Experience Framework

- Create custom user flows using XML syntax
  - Various starter packs and community-developed samples
- Azure Tutorial describes:
  - Adding signing and encryption keys
  - Setting framework policies
- Each flow is prefixed with “B2C_1A_”



praetorian

Enumerating B2C Domains

- All B2C authn/authz domains are *.b2clogin.com
- Subdomain enumeration
  - https://subdomains.whoisxmlapi.com/... – 1642 matches
  - Filter out test/qa environments
- For this vulnerability, need a victim:
  1. Azure B2C tenant
  2. client_id for a configured application (using PKCE/Hybrid Flow)
  3. IEF flow, ex. B2C_1A_SignUp_SignIn

praetorian

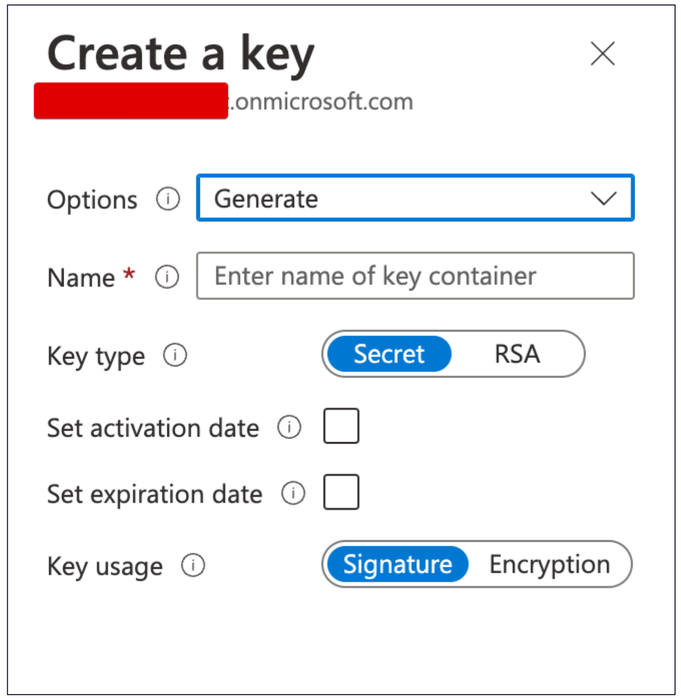
Background Info

Breadcrumbs to Crypto Misuse

Side-Channel Attack
MSRC as a Target
Recommendations & Disclosure

Documentation

- [Custom Policy Getting Started]



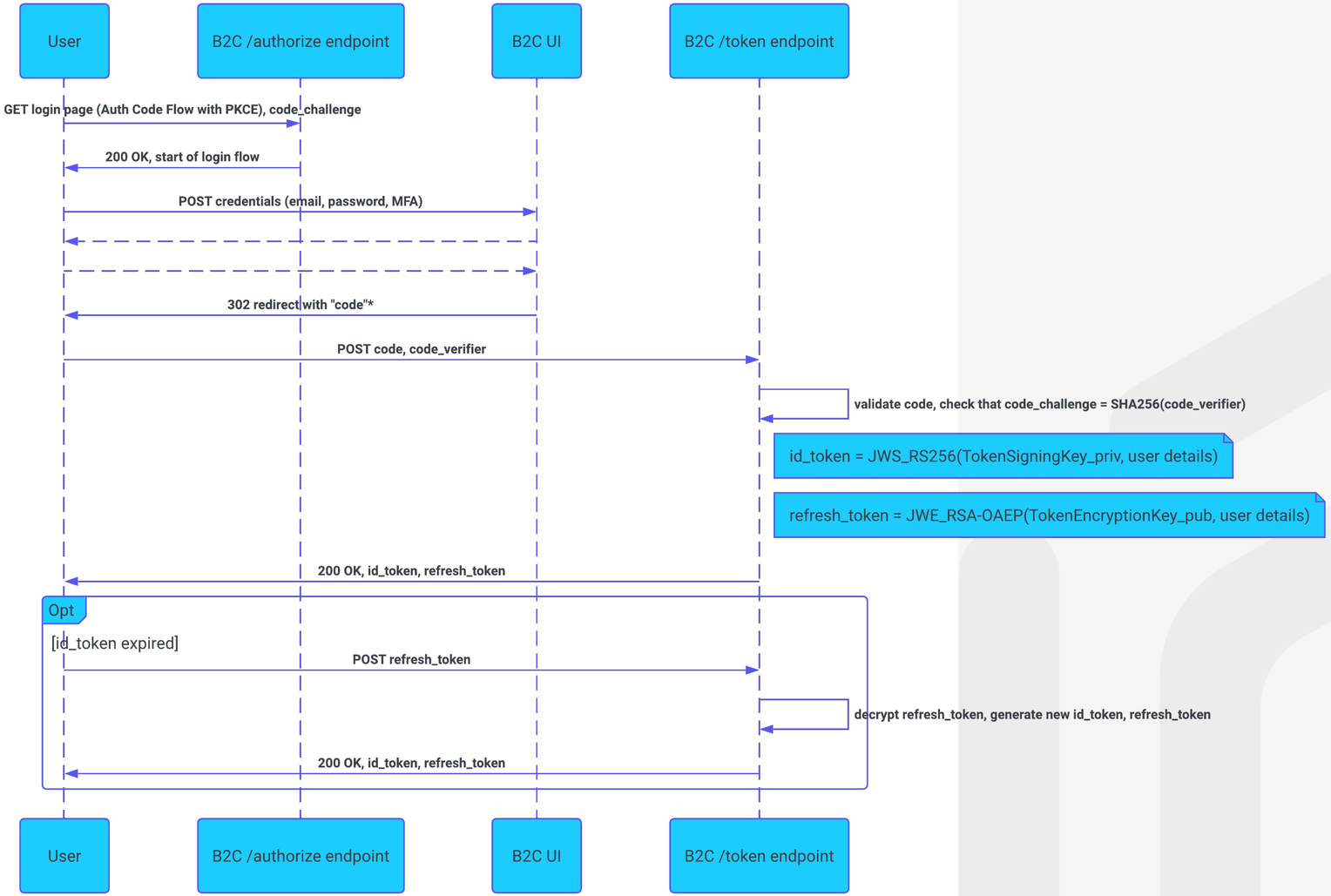
Create the signing key

1. Select **Policy Keys** and then select **Add**.
2. For **Options**, choose `Generate`.
3. In **Name**, enter `TokenSigningKeyContainer`. The prefix `B2C_1A`
4. For **Key type**, select **RSA**.
5. For **Key usage**, select **Signature**.
6. Select **Create**.

Create the encryption key

1. Select **Policy Keys** and then select **Add**.
2. For **Options**, choose `Generate`.
3. In **Name**, enter `TokenEncryptionKeyContainer`. The prefix `B2C`
4. For **Key type**, select **RSA**.
5. For **Key usage**, select **Encryption**.
6. Select **Create**.

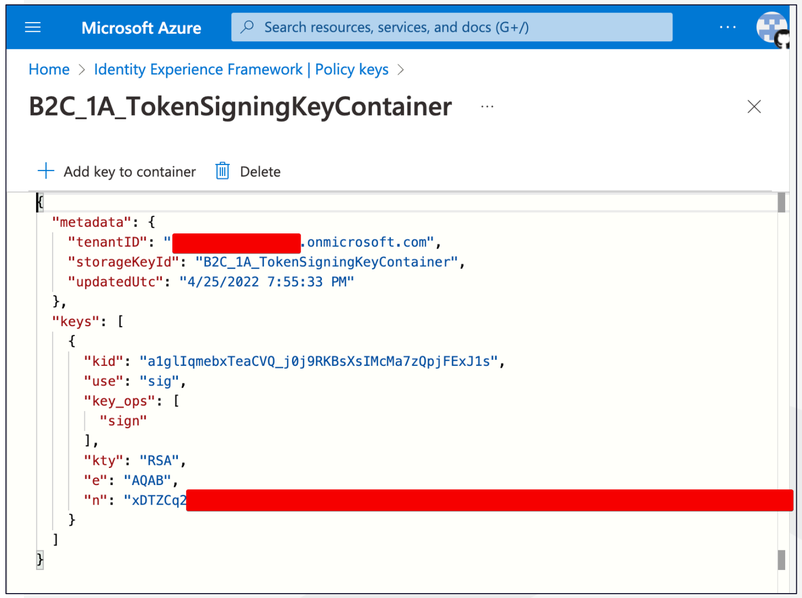
OAuth Login Flow

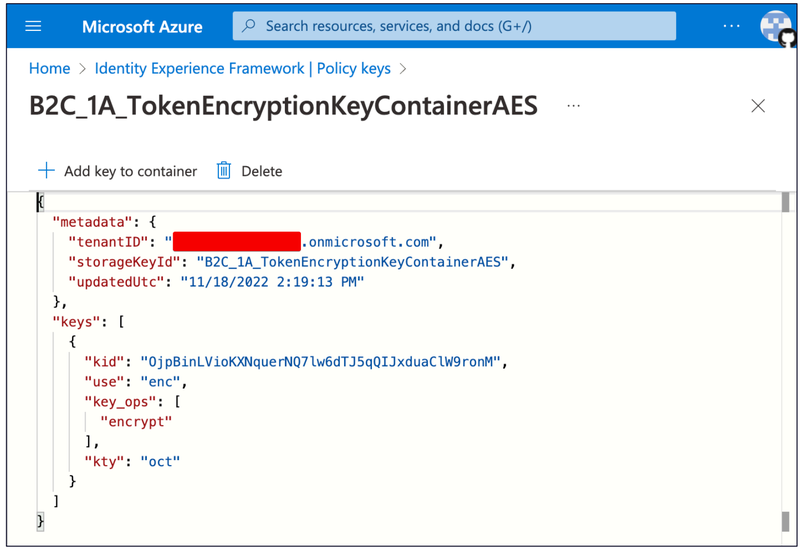


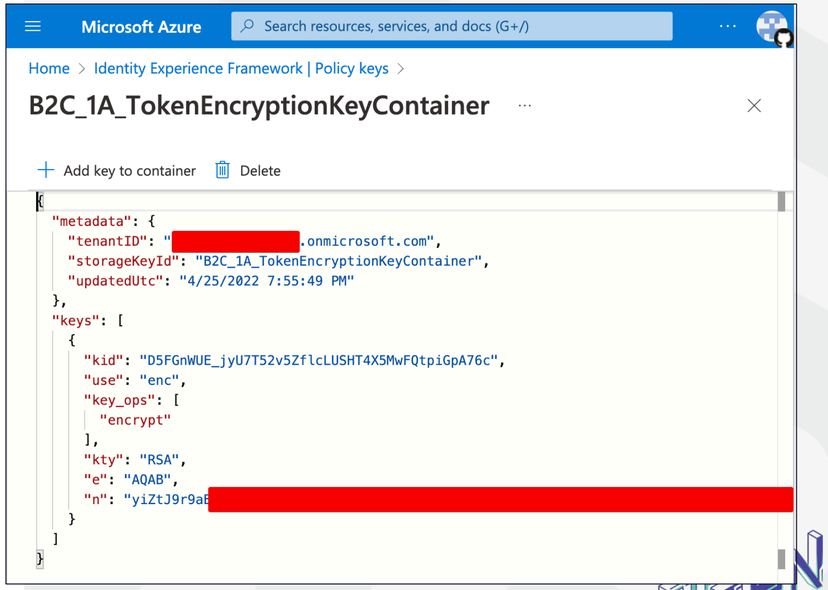
praetorian

Generating Keys

- Recommended RSA keys (right →)
- Not exportable, but can view public portions
- Can generate AES key instead (below)



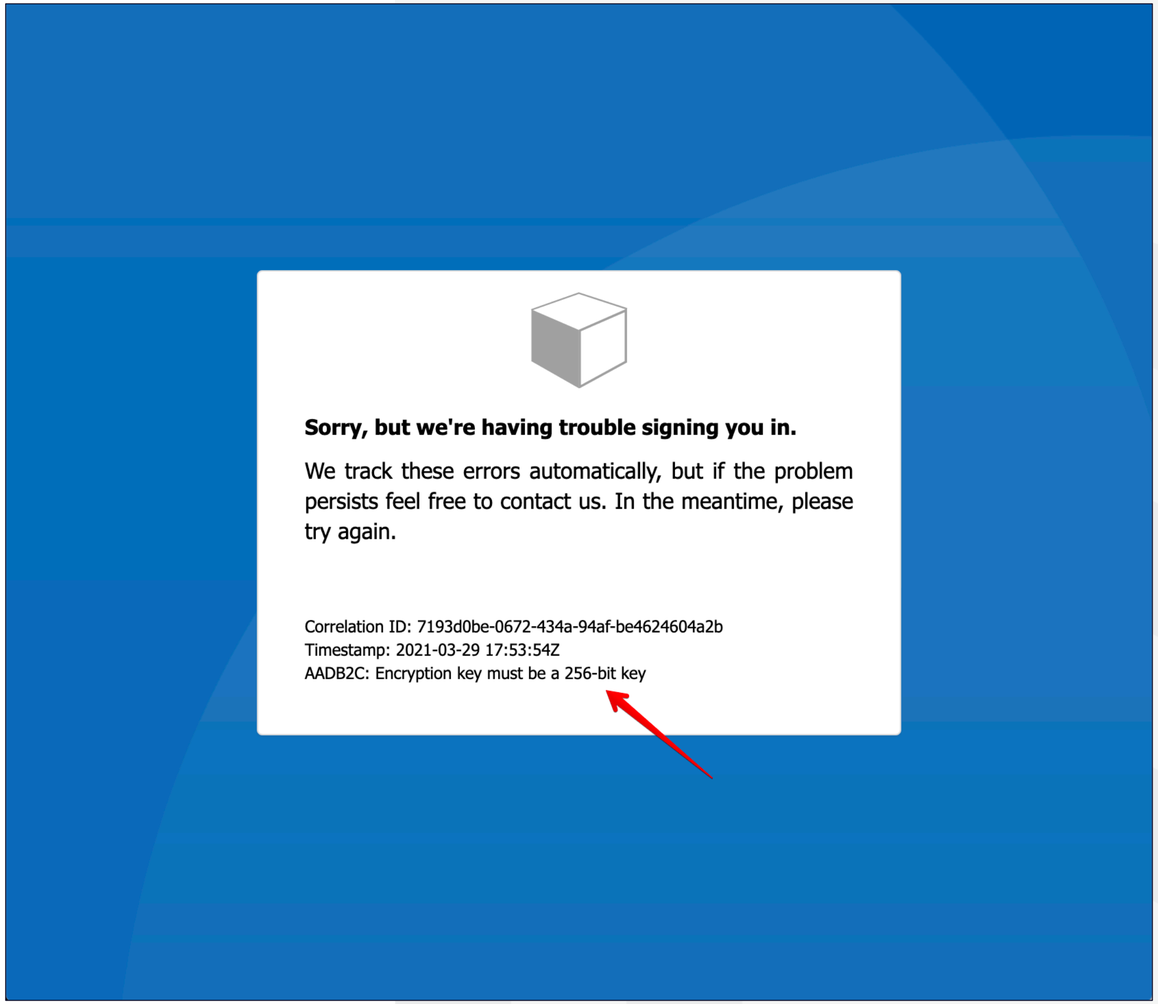




praetorian

Keys

- Choosing “Secret” key is allowed
- However, logins will fail
- Seems like 128-bit keys are generated

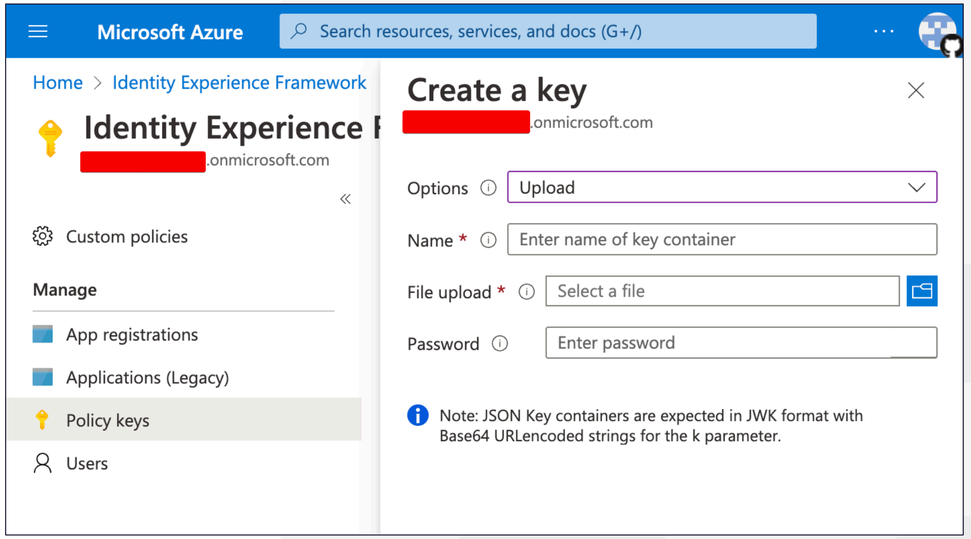


praetorian

Importing Keys

- Can upload a JWK of the right format instead
- Example format:

```
{
  "k": <base64-encoded-256-bit-key>,
  "kty": "oct",
  "use": "enc",
  "key_ops": ["encrypt"]
}
```




```
j = jwcrypto.jwk.JWK.generate(kty='oct',size=256)
js = json.loads(j.export())
js['use'] = 'enc'
js['key_ops'] = ['encrypt']
print(js)
```

praetorian

JWTs

id_token = JWS_RS256(TokenSigningKey_priv, user details)

refresh_token = JWE_RSA-OAEP(TokenEncryptionKey_pub, user details)

- Login flow with “openid” scope will return an **id_token**
  - JWS w/ “kid” corresponding to “issuer_secret”
- Auth Code login flow with “offline_access” scope will return a **refresh_token**
  - JWE w/ “kid” corresponding to “issuer_refresh_token_key”
  - JWE Header:

    `eyJraWQiOiJENUZHbldVRV9qeVU3VDUydjVaZmxjTFVTSFQ0WDVNd0ZRdHBpR3BBNzZjIiwidmVyIjoiMS4wIiwiemlwI joiRGVmbGF0ZSIsInNlciI6IjEuMCJ9`
  - Decoded:

    `{"kid":"D5FGnW…","ver":"1.0","zip":"Deflate","ser":"1.0"} ...`
  - Missing “alg” & “enc”; “zip” should be “DEF”

praetorian

Decrypting the Refresh Token

id_token = JWS_RS256(TokenSigningKey_priv, user details)

refresh_token = JWE_RSA-OAEP(TokenEncryptionKey_pub, user details)

- Generate policy key and upload it
- Trial-and-error algorithm types
  - RSA -> “RSA-OAEP” unwraps a 256-bit key
  - A256GCM -> “AES 256-bit GCM” decrypts compressed data
  - Deflate (zlib) to decompress claims
- Not a nested JWT

praetorian

Token Contents

id_token = JWS_RS256(TokenSigningKey_priv, user details)

refresh_token = JWE_RSA-OAEP(TokenEncryptionKey_pub, user details)

Refresh Token Claims:


```
{
 "tid": "<omitted>.onmicrosoft.com",
 "pid": "B2C_1A_signup_signin",
 "t": 2,
 "cls": {
  "$id": "1",
  "$values": [
   {
    "claimTypeId": "sub",
    "value": "9d8<omitted>"
   },
   {
    "claimTypeId": "name",
    "value": "unknown"
   },
   {
    "claimTypeId": "given_name",
    "value": "John"
   },
   {
    "claimTypeId": "family_name",
    "value": "Doe"
   },
   {
    "claimTypeId": "tid",
    "value": "a4e<omitted>"
   },
   {
    "claimTypeId": "isAdmin",
    "value": true
   },
   {
    "claimTypeId": "extraParam",
    "value": "extraValue"
   }
  ]
 },
 "o_aud": "7b1<omitted>",
 "o_iat": 1617027992,
 "iat": 1617027993,
 "exp": 1618237593,
 "avm": "V2.0",
 "rcc": false,
 "scp": [
  "offline_access",
  "openid"
 ],
 "uid": "9d8<omitted>",
 "sa": [
  "offline_access openid"
 ],
 "l": "0c4<omitted>"
}
```


ID Token Claims:


```
{
  "exp": 1617045712,
  "nbf": 1617042112,
  "ver": "1.0",
  "iss": "https://<omitted>.b2clogin.com/a4e<omitted>/v2.0/",
  "sub": "9d8<omitted>",
  "aud": "7b1<omitted>",
  "acr": "b2c_1a_signup_signin",
  "iat": 1617042112,
  "auth_time": 1617038508,
  "name": "unknown",
  "given_name": "John",
  "family_name": "Doe",
  "tid": "a4e<omitted>",
  "isAdmin": true,
  "extraParam": "extraValue"
}
```


praetorian

Creating Tokens

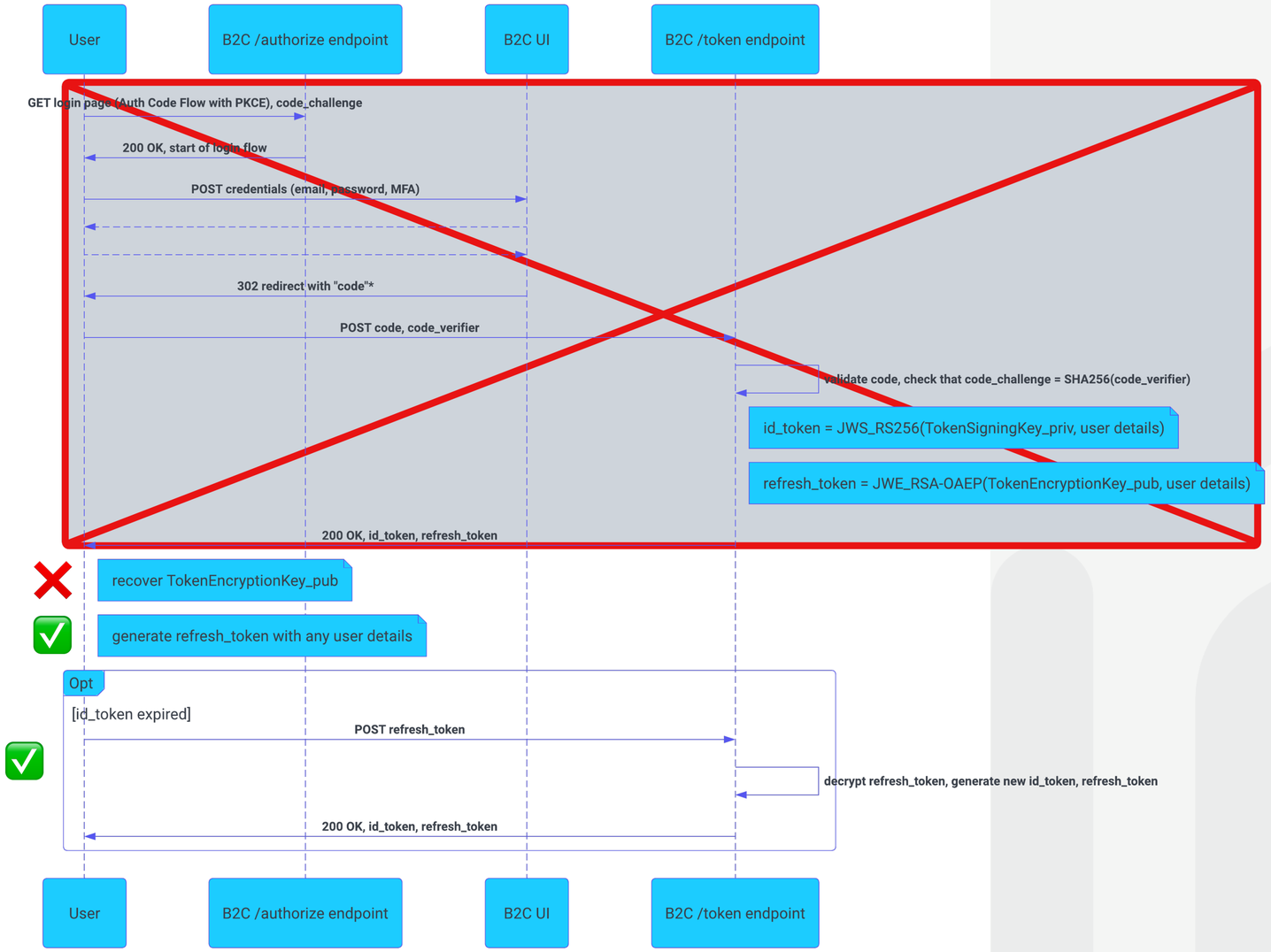
id_token = JWS_RS256(TokenSigningKey_priv, user details)

refresh_token = JWE_RSA-OAEP(TokenEncryptionKey_pub, user details)

- With format known, a user can create the claims in a refresh token
- If configured with RSA, the public key can be used to encrypt/generate a refresh token
- Public key is exposed in Azure Portal to:
  - Global Administrator
  - Global Reader
  - IEF Key Set Administrator

praetorian

Creating Tokens



Disclosure (Part 1)

- Made to MSRC in March 2021
- Knowledge of the public key for refresh tokens can be used to compromise any user
- Public key is not published outside Azure Portal
- Issue closed with no action taken in April 2021

- I’d argue that if security depends on hiding a *public key* then it isn’t secure

Read-Only Access – My 2C

- If an attacker has read-only access to your source code, this should not introduce any new vulnerabilities
  - ex. no hardcoded keys, backdoors, or configuration options
- Read-only cloud access should be similar
  - ex. Global Reader access should not allow an attacker to elevate privileges, read key information, access database entries

praetorian

Background Info
Breadcrumbs to Crypto Misuse

Side-Channel Attack

MSRC as a Target
Recommendations & Disclosure

Public Keys

- Asymmetric encryption is not the right choice for tokens because they are generated with a public key
- RSA public components:
  - N – modulus, 2048 bits, “unknown”
  - e – exponent, typically fixed to 65537
- $cipher < N$
  - can get a lower bound for N with enough samples
  - Likely only recover $\log_2(\#samples)$ bits of N

Public Keys

- RSA components:
  - N – modulus, 2048 bits, “unknown”
  - e – exponent, typically fixed to 65537
  - d – private exponent
- Submitting a refresh token to B2C effectively calls:

  decrypt_rsa($cipher, N, d$)

  with a $cipher$ of our choice

Asymmetric Decryption

- def decrypt_rsa($cipher, N, d$):
  - $plain = cipher^d \ mod \ N$
  - Verify OAEP of $plain$ and remove padding
    - involves SHA256 hash
  - If verified, return $plain$ (w/o padding), else return error
- Computationally expensive – modular exponentiation and SHA256 hash

Asymmetric Decryption

- def decrypt_rsa($cipher, N, d$):
  - If $cipher \geq N$, return error
  - $plain = cipher^d \ mod \ N$
  - Verify OAEP of $plain$ and remove padding
    - involves SHA256 hash
  - If verified, return $plain$ (w/o padding), else return error
- Suppose a crypto library tries to save time for obviously wrong ciphertext

Timing Attack

- def decrypt_rsa($cipher, N, d$):
  - If $cipher \geq N$, return error } $time1$
  - $plain = cipher^d \ mod \ N$
  - Verify OAEP of $plain$ and remove padding
    - involves SHA256 hash
  - If verified, return $plain$ (w/o padding), else return error } $time2$
- The time to return reveals if $cipher \geq N$ or $< N$

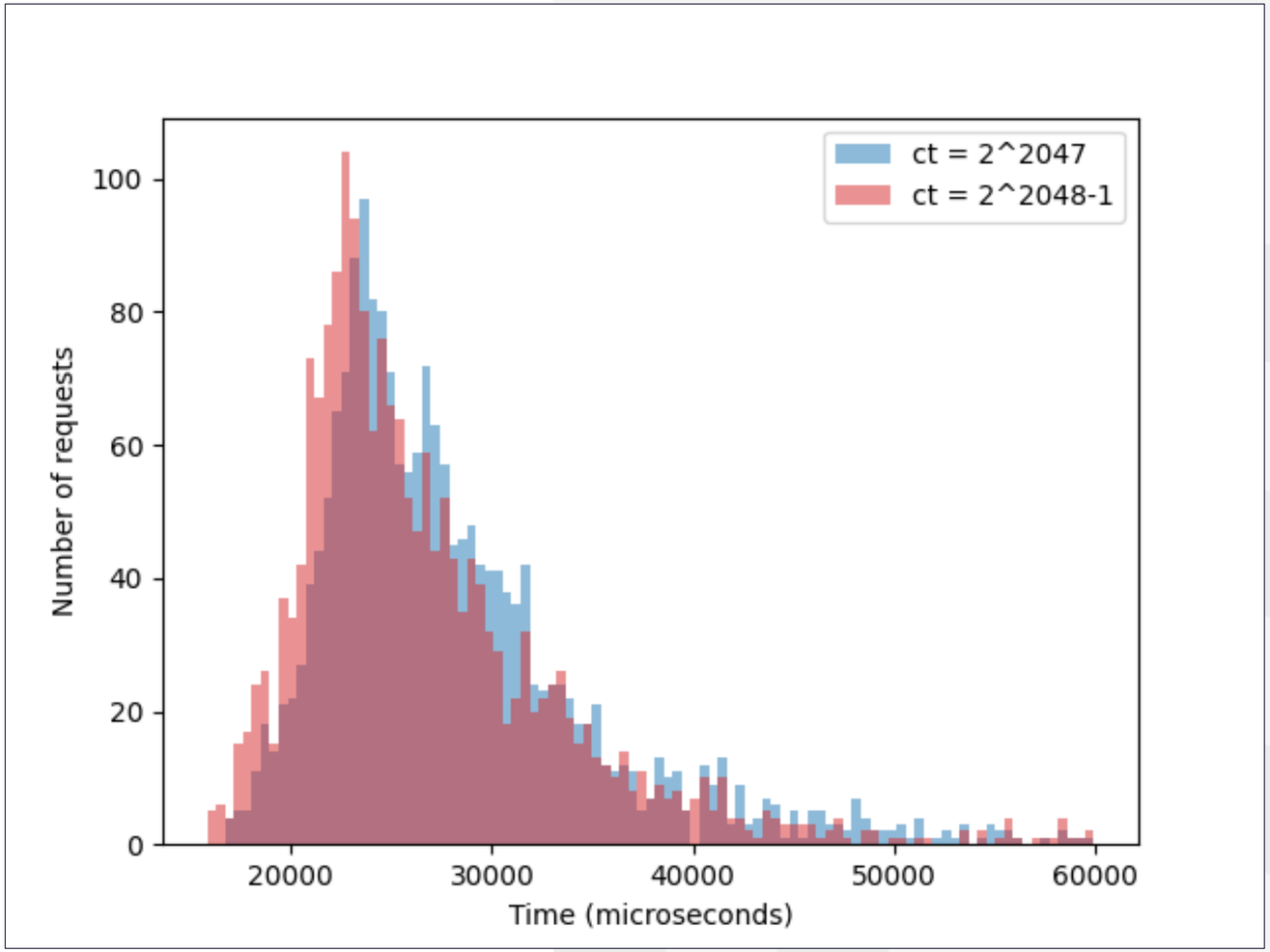
praetorian

Timing Attack

- JWE with cek of 2^{2047} = 0x8000… and $2^{2048} - 1$ = 0xffff…
  - `(header).gAAA(…)AA.(iv).(ct).(tag)`
  - `(header).___(…)__w.(iv).(ct).(tag)`
- Send “grant_type=refresh_token” request to `/token`
- Observe response time and compare
- Reduce latency
  - Run on cloud infrastructure (Azure VM) in same region (East US)
  - Avoid load balancing by fixing IP address (hardcode in /etc/hosts)

Timing Attack

- Submit 2000 requests
  - ~ 98.5% valid
- Lower values have → longer response time
- Used test B2C environment
- Average (mean):
  - 28.1 ms for 2^{2047}
  - 26.8 ms for 2^{2048} -1



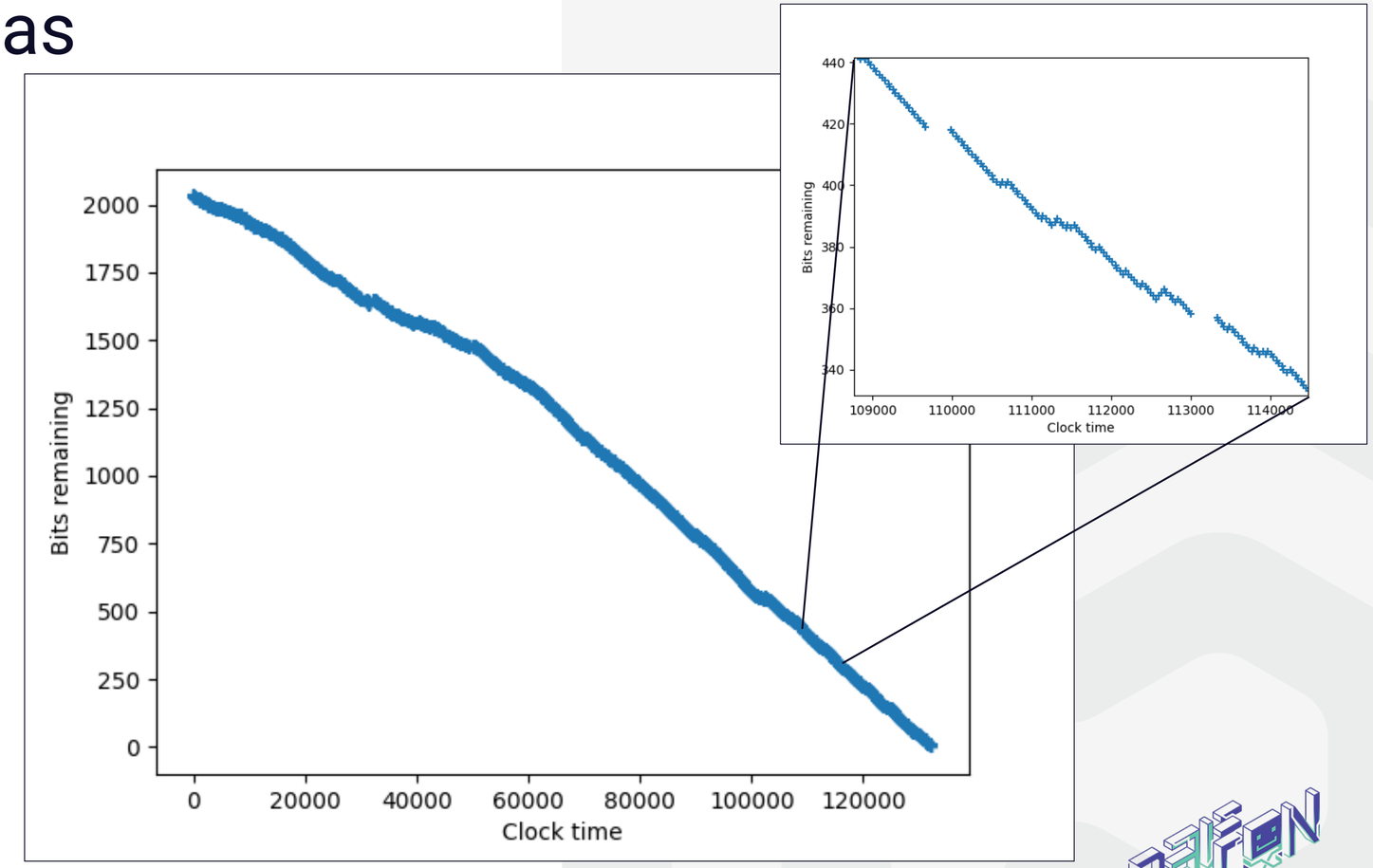
praetorian

Timing Attack

- Generalize one round:
  - Start with upper & lower values
  - Submit *n* samples of cek = $\{upper, mid = \frac{upper+lower}{2}, lower\}$
  - If mid has response time closer to upper or lower, then set that value to mid
  - Repeat (binary search)
- Recovers “1 bit” of public key each round

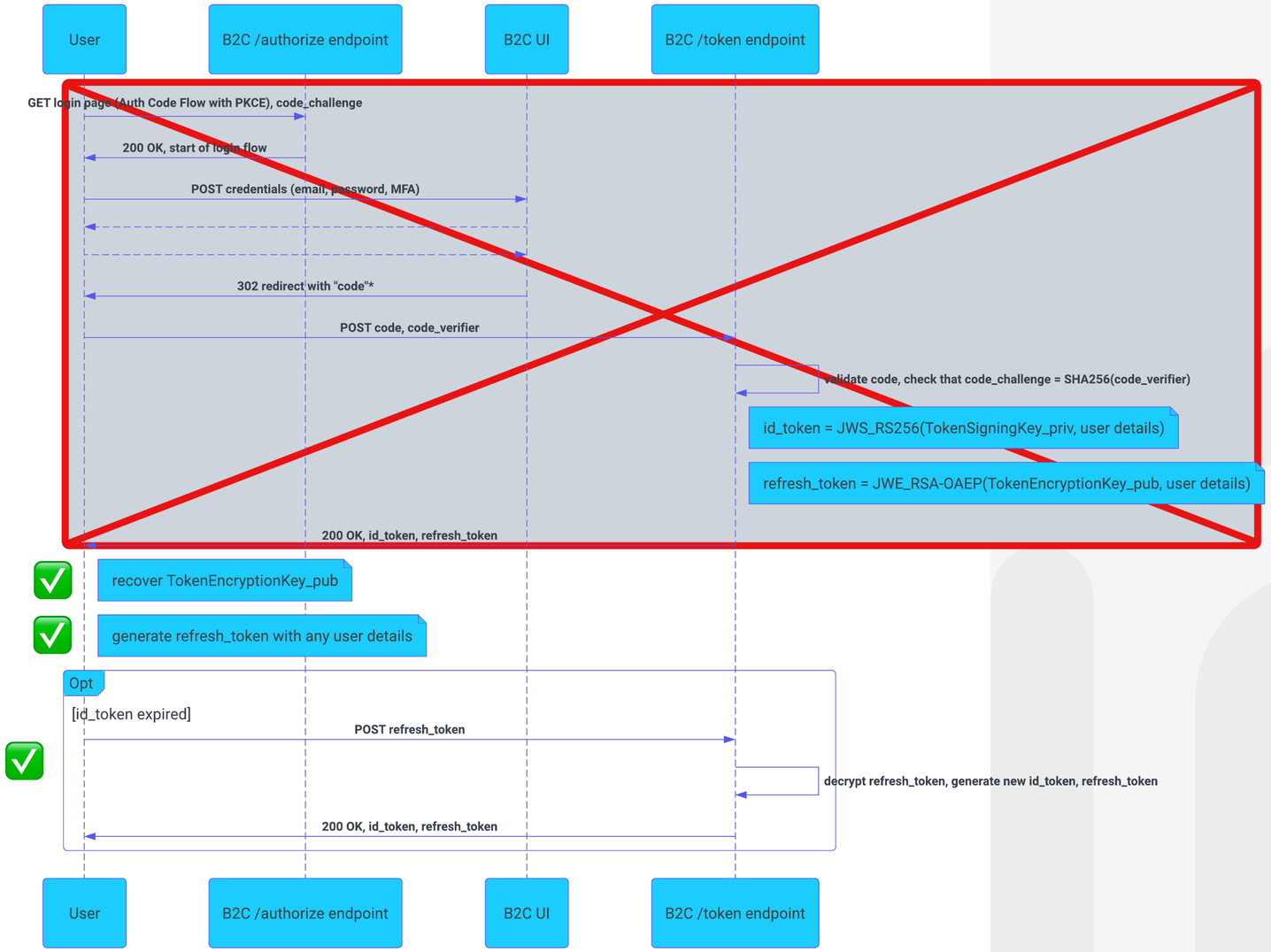
Timing Attack

- Each round is not 100% reliable, requires backtracking
- Azure token endpoint has rate limiting
- Implemented attack recovers ~55 bits/hour
- Total time: 37 hours



praetorian

Creating Tokens

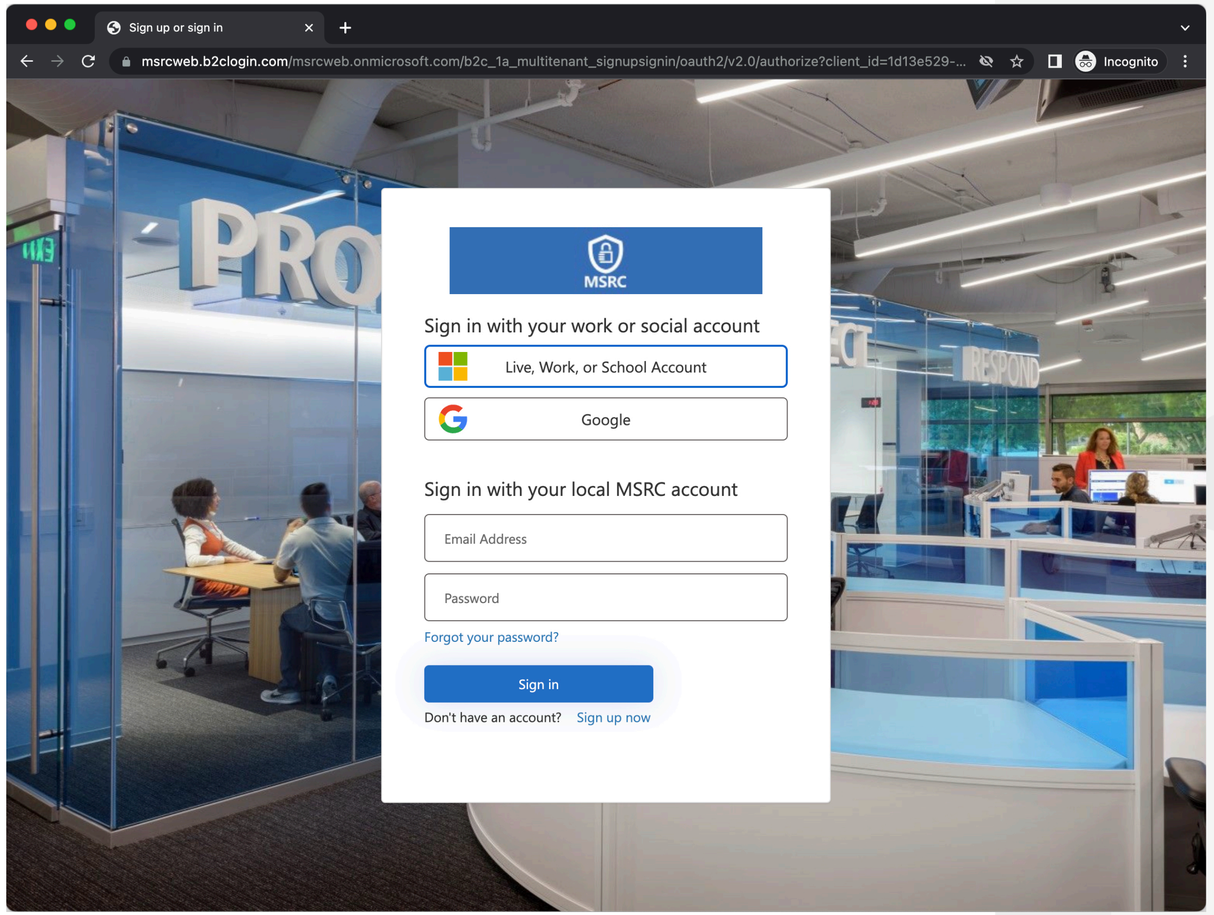


Background Info
Breadcrumbs to Crypto Misuse
Side-Channel Attack

MSRC as a Target

Recommendations & Disclosure

Microsoft Security Response Center (MSRC)



praetorian

Microsoft Security Response Center (MSRC)

- Used to report security vulnerabilities to Microsoft
- SPA which uses Azure B2C for session management
  - https://msrcweb.b2clogin.com
- Self-signup with username/password or social account
  - b2c_1a_multitenant_signupsignin
- Authentication with default IEF flow returns refresh_token and id_token to the user

praetorian

Microsoft Security Response Center (MSRC)

- Recovered public modulus for MSRC kid
  `VLR0YVyvbVMaew_iHKXBWbJDIozbnstb9Mi4rAkuzcg`
- Used known format of refresh token to craft a token for a (fake) victim user
- Encrypted the contents
- Sent to the msrcweb B2C /token endpoint
- Got back an id_token for the (fake) victim user

MSRC – Proof of Work

- ID token for msrcweb.b2clogin.com →
- Claims:

```
{
  "exp": 1659843146,
  "nbf": 1659839546,
  "iss": "https://msrcweb.b2clogin.com/9f449d34-93e9-437f-8082-7127fc8ab474
  …
  "email": "alice.bob@example.com",
  ”defcon": 31,
  …
}
```

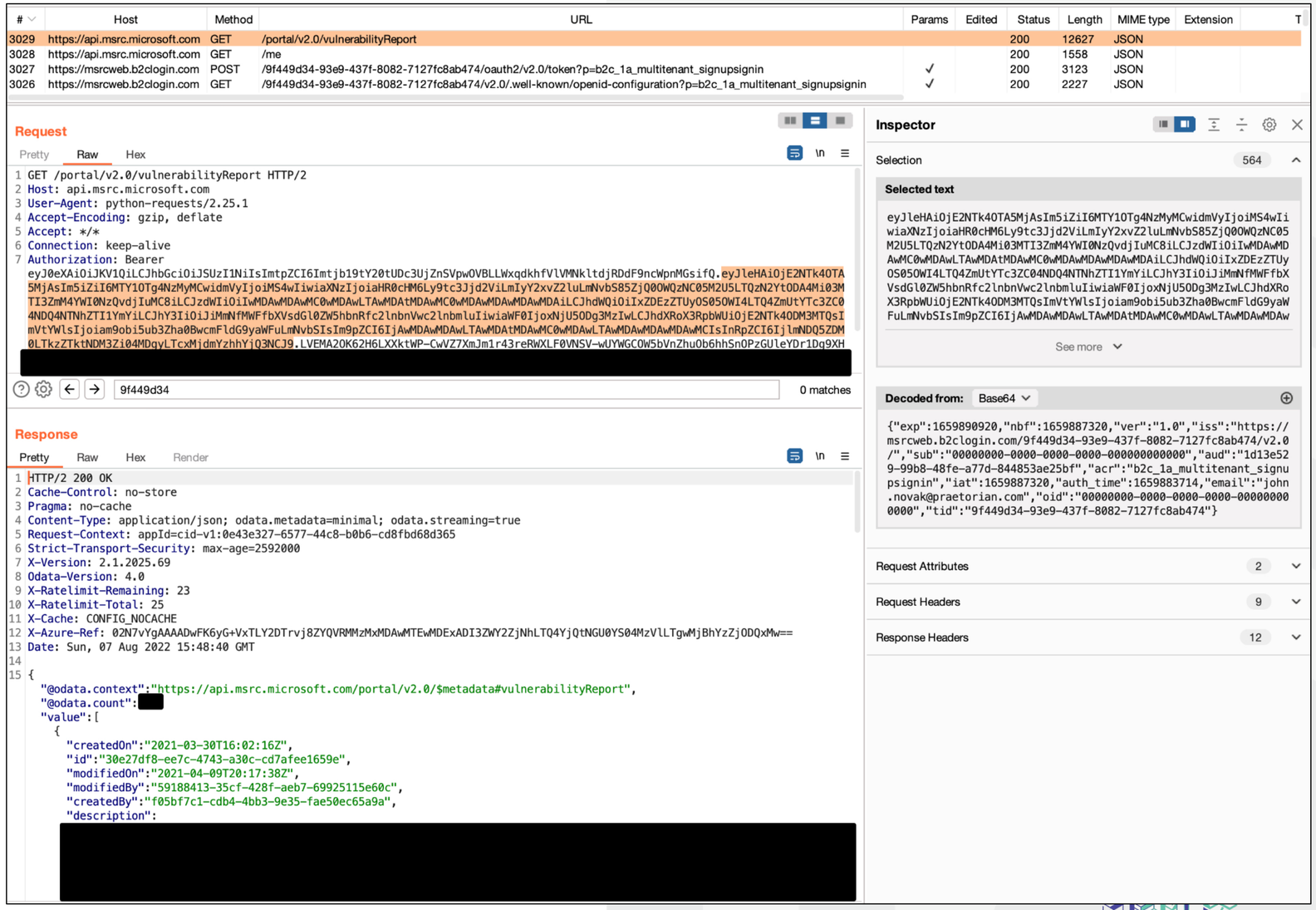
- Verify Token:
  - OpenID Key for msrcweb
    (“iss” + “/discovery/v2.0/keys?p=b2c_1a_multitenant_signupsignin”)

```
from jwcrypto import jwt, jwk
tok="eyJ0..." (QR code)
k={"kid":"kco…"} (from OpenID Key link)
key=jwk.JWK(**k)
jwt.JWT(key=key,jwt=tok,check_claims=False)
```

praetorian

MSRC – Listing Vulnerability Reports

- Access /vulnerabilityReport
- Dummy sub, oid
- Requires:
  - Audience “aud”
  - Tenantid “tid”



praetorian

Impact

- Can list vulnerability submissions knowing only a user's email
- Contents of vuln reports are likely:
  - Windows, Azure, Exchange, etc. zero-days
  - Junk submissions
- Likely could change user information or bug bounty payment processor IDs

Background Info
Breadcrumbs to Crypto Misuse
Side-Channel Attack
MSRC as a Target

Recommendations & Disclosure

Crypto Confusion

- Encryption is for confidentiality; Signing/MAC is for integrity
  - Access or refresh tokens need integrity – you don’t want an attacker to modify an existing one or create a new one
- Crypto misuse can be misunderstood without a clear attack/impact

Recommendations

- Discussed on Praetorian Blog
- Option 1: Key Rotation – new RSA key
- Option 2: Key Type Change – RSA to Secret
- Incident Response
- To Microsoft:
  - Use nested JWTs
  - Fix “Secret” key generation function

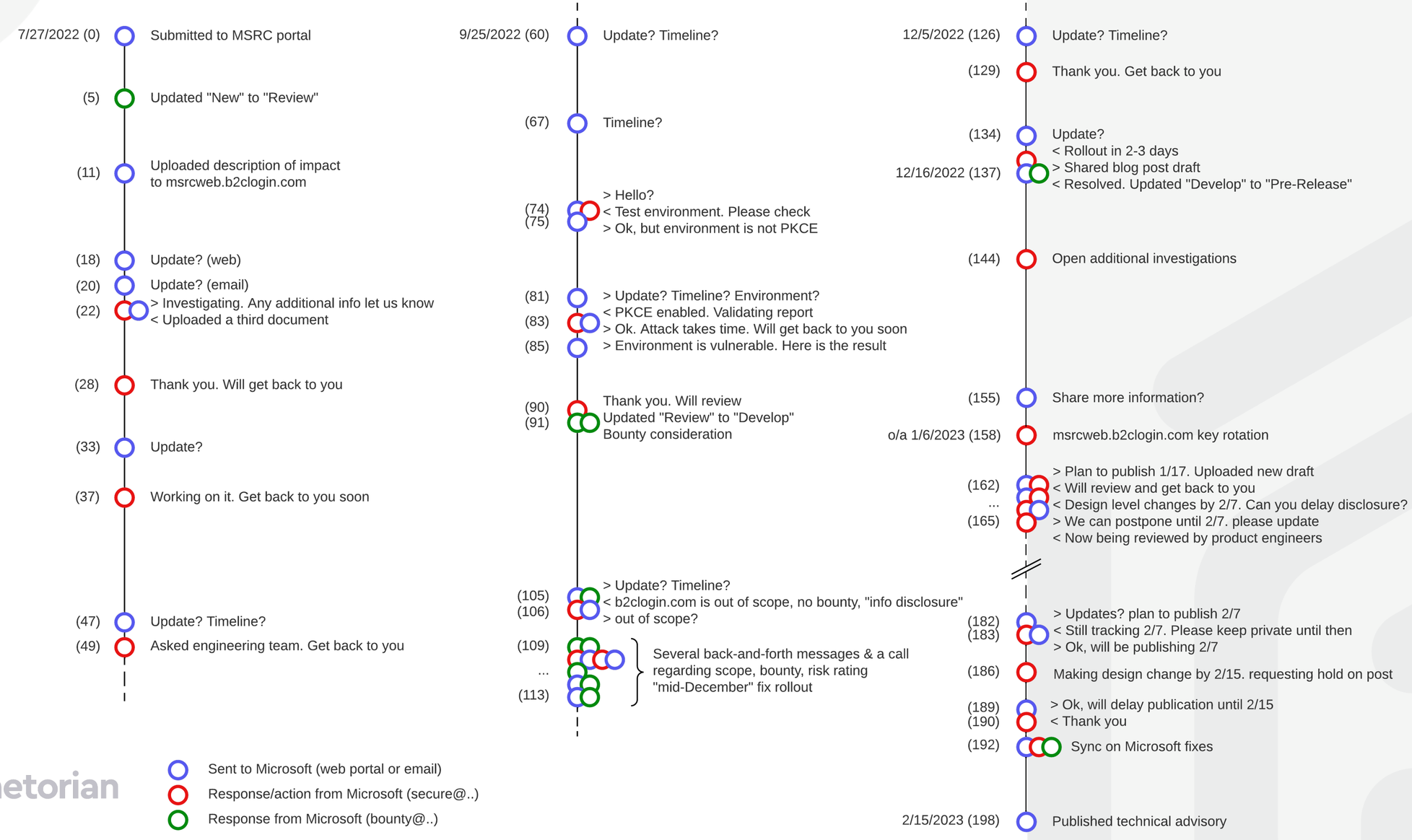
praetorian

Disclosure (Part 2)

- March 2021 – “Crypto Misuse” and prior details submitted to MSRC
- April 2021 – Microsoft closed without remediation citing only privileged users could access public key information

- July 2022 – discovery of side channel & disclosure
- December 2022 – resolved with “no response” on /token endpoint
- ~~February 2023 – added new element to refresh tokens signed with Microsoft key~~

Disclosure



praetorian

- Sent to Microsoft (web portal or email)
- Response/action from Microsoft (secure@..)
- Response from Microsoft (bounty@..)

Risk Rating

- Assigned Important, not Critical
  - Pre-requisites: B2C tenant details, victim email address
  - Impact: Full compromise of victim’s account
- Categorized as “Information Disclosure” not “Elevation of Privilege”
  - A public key is disclosed
  - That key is used to effectively “grant privileges” to any victim account

Bug Bounty Consideration

- At disclosure, two programs which might apply:
  - Microsoft Azure Bounty Program
  - Microsoft Identity Bounty Program
- Not eligible for Azure since this is an Identity service
- Not eligible for Identity since “*.b2clogin.com” is not in scope

- Microsoft subsequently added Azure B2C to its Identity program in June 2023

Lingering Remediations

- Microsoft’s first fix was narrow against the info disclosure
  - The /token endpoint doesn’t return for invalid refresh tokens
  - Unclear if further attacks to recover a public key are possible
- No release notes or recommendation from Microsoft to move away from public key cryptography
  - “Secret Key” generation still broken
- Key rotation not forced on Microsoft customers
  - Completed on msrcweb.b2clogin.com o/a 1/6/23
- New signed element in refresh tokens not yet implemented

praetorian

* as of July 13, 2023

Background Info
Breadcrumbs to Crypto Misuse
Side-Channel Attack
MSRC as a Target
Recommendations & Disclosure

One More Thing…

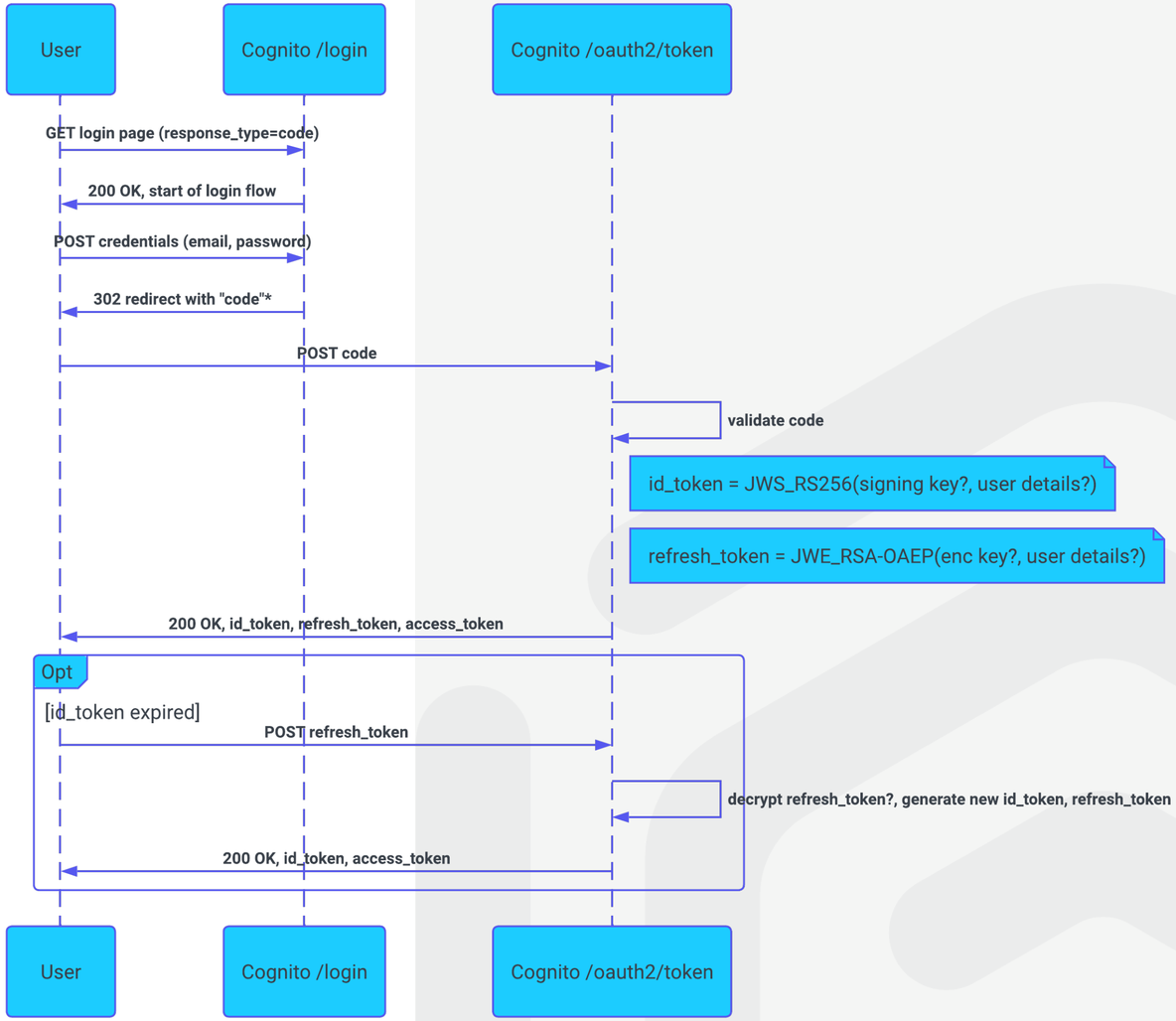
AWS Cognito

- Basic setup following developer guides
- Encrypted refresh tokens with header:

```
eyJjdHkiOiJKV1QiLCJlbmMiOiJBMjU2R0N
NIiwiYWxnIjoiUlNBLU9BRVAifQ …
```

- Decoded:

```
{"cty":"JWT","enc":"A256GCM","alg":
"RSA-OAEP"} ...
```



Thank you!

praetorian