Identity Theft

Attacks on Modern SSO Systems

DUO LABS

$ whoami

- Tweets - @kelbyludwig
- Blogs - https://kel.bz
- I do AppSec @ Duo
- Dabbles in cryptography, math, security engineering

Agenda

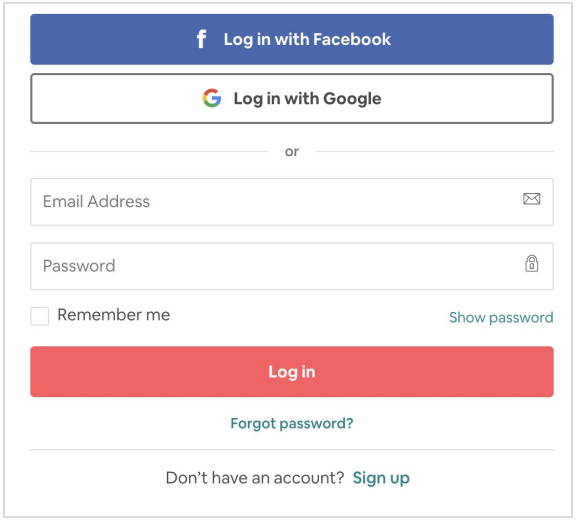
Goal: Cover new classes of attacks on single sign-on and SAML

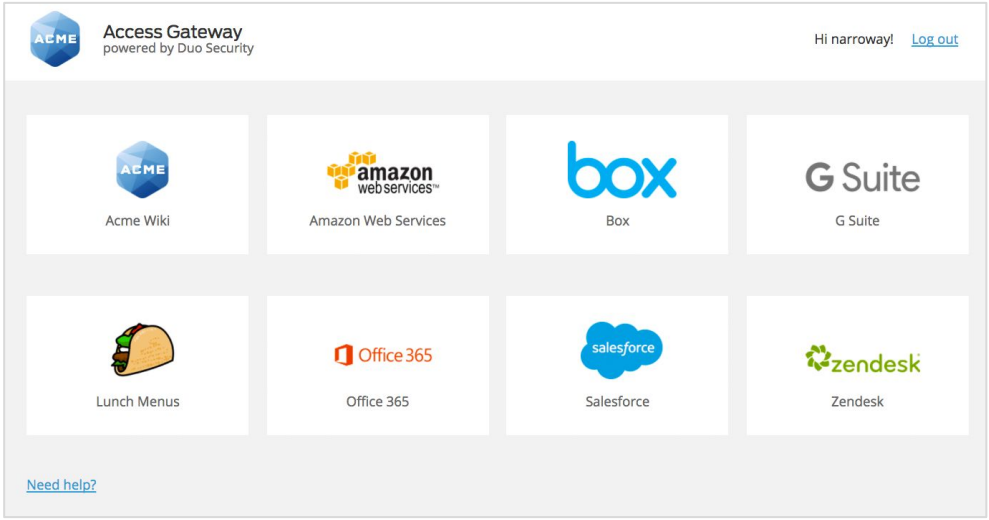
1. SAML at a high-level
2. New vulnerabilities affecting SAML and SSO systems
3. Exploiting instances of these vulnerabilities
4. Mitigation and Conclusion

SAML and SSO

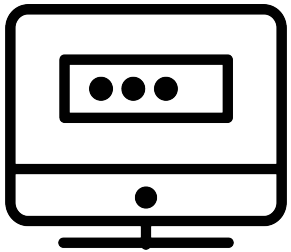
Single Sign On

- Authenticate once; Get access to multiple applications.
- UX improvement on N-passwords for N-applications.





SSO: Cast of Characters

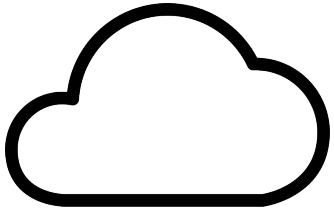


Identity Provider:

The service where you authenticate; IdP

User Agent:

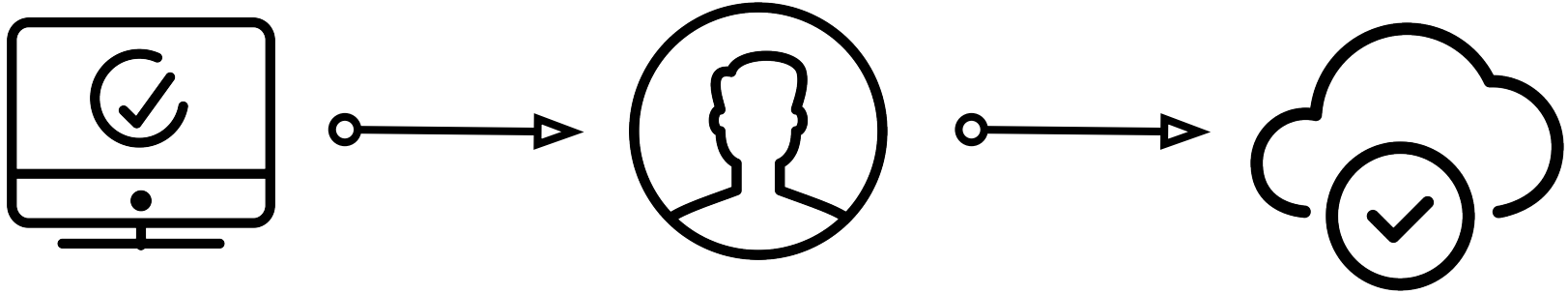
Your web browser; The message passer



Service Provider:

The service you want to access; SP

Example SSO Workflow



Once authenticated IdP passes a message through browser

Your browser takes this message and passes it to the SP

SP validates the message and then authNs you to their service

SAML

- SAML: Security Assertion Markup Language
- SAML 1.0 defined in 2002, SAML 2.0 defined in 2005
- A common language and workflow between systems that want to provide SSO
- Includes other standards that provide security controls which prevent an untrusted message passer from tampering with messages

Simplified SAML Document

```
<Response Destination="serviceprovider.com">
  <Assertion ID="thing_to_sign">
    <Subject>
      <NameID>kelbyludwig</NameID>
    </Subject>
    <AttributeStatement>
      <Attribute Name="role">
        <AttributeValue>admins</AttributeValue>
      </Attribute>
    </AttributeStatement>
  </Assertion>
  <Signature><!-- Lots of things! --></Signature>
```

Key Elements: Subject and NameID

```
<Response Destination="serviceprovider.com">
  <Assertion ID="thing_to_sign">
    <Subject>
      <NameID>kelbyludwig</NameID>
    </Subject>
    <AttributeStatement>
      <Attribute Name="role">
        <AttributeValue>admins</AttributeValue>
      </Attribute>
    </AttributeStatement>
  </Assertion>
  <Signature><!-- Lots of things! --></Signature>
```

Key Elements: Attributes

```
<Response Destination="serviceprovider.com">
  <Assertion ID="thing_to_sign">
    <Subject>
      <NameID>kelbyludwig</NameID>
    </Subject>
    <AttributeStatement>
      <Attribute Name="role">
        <AttributeValue>admins</AttributeValue>
      </Attribute>
    </AttributeStatement>
  </Assertion>
  <Signature><!-- Lots of things! --></Signature>
```

Key Elements: Signature

```
<Response Destination="serviceprovider.com">
  <Assertion ID="thing_to_sign">
    <Subject>
      <NameID>kelbyludwig</NameID>
    </Subject>
    <AttributeStatement>
      <Attribute Name="role">
        <AttributeValue>admins</AttributeValue>
      </Attribute>
    </AttributeStatement>
  </Assertion>
  <Signature><!-- Lots of things! --></Signature>
```

XMLDSig

Simplified Signature Element

```
<Assertion ID="thing_to_sign">[...]</Assertion>
<Signature Id="signature_id">
  <SignedInfo>
    <CanonicalizationMethod Alg="xml-c14n11"/>
    <SignatureMethod Algorithm="rsa-sha256"/>
    <Reference URI="thing_to_sign">
      <Transforms>
        <Transform Algorithm="xmldsig#base64"/>
      </Transforms>
      <DigestMethod Algorithm="sha1"/>
      <DigestValue>eW8=</DigestValue>
    </Reference></SignedInfo></Signature>[...]
```

XML Canonicalization (C14N)

- XML canonicalizes data prior to signature operations.
- Logically equivalent documents have the same signature.
- The following elements might be logically equivalent:

```
<Thing A="1" B="2">Hello!</Thing>

<Thing B="2" A="1">Hello!</Thing>

<Thing B="2" A="1">Hello!<!--comment!--></Thing>
```

XML Canonicalization (C14N)

- XML canonicalizes data prior to signature operations.
- Logically equivalent documents have the same signature.
- The following elements might be logically equivalent:

```
<Thing A="1" B="2">Hello!</Thing>

<Thing B="2" A="1">Hello!</Thing>

<Thing B="2" A="1">Hello!<!--comment!--></Thing>
```

XML Canonicalization (C14N)

- XML canonicalizes data prior to signature operations.
- Logically equivalent documents have the same signature.
- The following elements might be logically equivalent:

```
<Thing A="1" B="2">Hello!</Thing>

<Thing B="2" A="1">Hello!</Thing>

<Thing B="2" A="1">Hello!<!--comment!--></Thing>
```

XML C14N Algorithms

"Implementations are REQUIRED to be capable of producing canonical XML excluding all comments that may have appeared in the input document or document subset. Support for canonical XML with comments is RECOMMENDED. "

https://www.w3.org/TR/2008/REC-xml-c14n11-20080502/#DataModel

SAML Library APIs

SAML APIs

- SAML defines "language" to convey data. What you do with the data is up to implementers
- APIs often provide *non-canonical* doc to users after verification.

```
saml_response = init_saml_auth(http_request)
saml_response.process_response()
errors = saml_response.get_errors()
if len(errors) == 0:
  user_id = saml_response.get_nameid()
  authenticate(user_id)
else:
  # user is not authenticated
```

SAML APIs

- SAML defines "language" to convey data. What you do with the data is up to implementers
- APIs often provide *non-canonical* doc to users after verification.

```
saml_response = init_saml_auth(http_request)
saml_response.process_response()
errors = saml_response.get_errors()
if len(errors) == 0:
  user_id = saml_response.get_nameid()
  authenticate(user_id)
else:
  # user is not authenticated
```

SAML APIs

- SAML defines "language" to convey data. What you do with the data is up to implementers
- APIs often provide *non-canonical* doc to users after verification.

```
saml_response = init_saml_auth(http_request)
saml_response.process_response()
errors = saml_response.get_errors()
if len(errors) == 0:
  user_id = saml_response.get_nameid()
  authenticate(user_id)
else:
  # user is not authenticated
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelbyludwig</NameID>"
data = fromstring(payload)
return data.text
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelbyludwig</NameID>"
data = fromstring(payload)
return data.text
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelbyludwig</NameID>"
data = fromstring(payload)
return data.text
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelbyludwig</NameID>"
data = fromstring(payload)
return data.text
#=> kelbyludwig
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelby<!---->ludwig</NameID>"
data = fromstring(payload)
return data.text
```

XML APIs: Text Extraction

- SAML APIs need to process XML data
- In order to get e.g. the username we need to extract the inner text of a NameID

```
from defusedxml.lxml import fromstring
payload = "<NameID>kelby<!---->ludwig</NameID>"
data = fromstring(payload)
return data.text
#=> kelby
```

Why? Trees, probably.

<NameID>kelbyludwig</NameID> may be represented as:

```
ElementNode: NameID
|_ TextNode: "kelbyludwig"
```

<NameID>kelby<!---->ludwig</NameID> may be represented as:

```
ElementNode: NameID
|_ TextNode: "kelby"
|_ CommentNode: "comment!"
|_ TextNode: "ludwig"
```

Observations on Correctness

- Ruby's `REXML` also exhibits this behavior in their `.text` method, but the behavior is documented in a example.
- Non-`lxml` python libraries like `xml.etree` have `.text` methods that wouldn't be truncated
- Some libraries don't provide a method like `.text`, so text extraction would be implemented by the user.
- In short: **It's arguably technically correct to do this, but it is not very intuitive.**

Tampering with Signed Data

Putting it all together

- SAML documents…
  - Contain information about the authenticating user, within the inner text
  - Are often passed through a user's browser from the IdP to the SP
  - Are signed to prevent tampering
  - Are canonicalized, in most cases excluding comments, prior to signing
  - Because of c14n, the addition of comments would not affect a document signature
- SAML/XML APIs
  - Have unintuitive and/or inconsistent behavior when extracting inner text
  - Provide the non-canonical document representation for post-signature processing

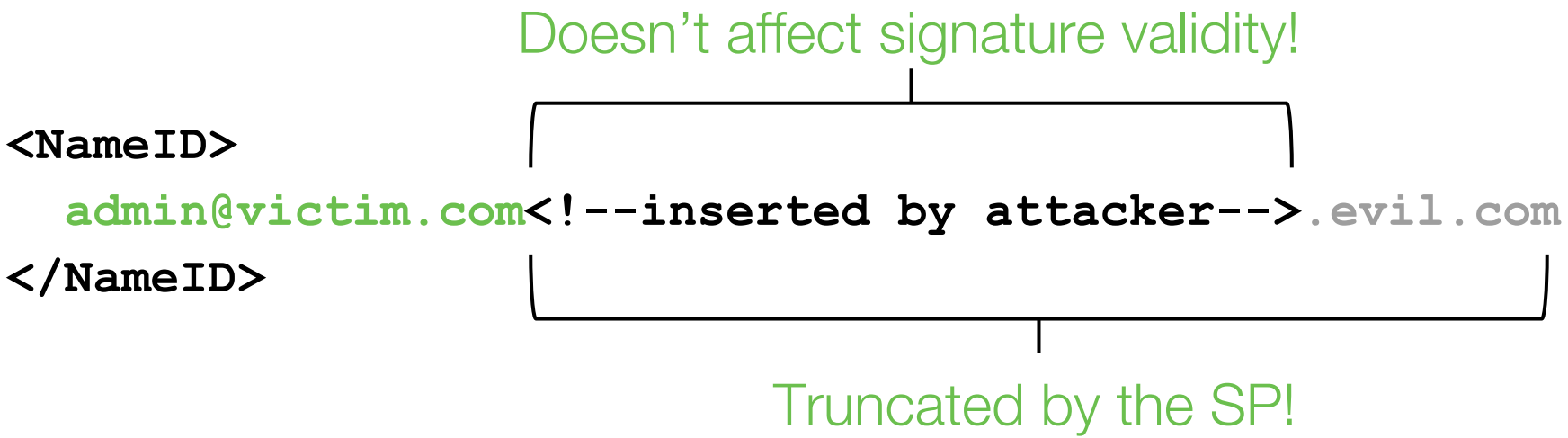
The Attack

```
<NameID>
  admin@victim.com.evil.com
</NameID>
```

The Attack

```
<NameID>
  admin@victim.com<!--inserted by attacker-->.evil.com
</NameID>
```

The Attack


Doesn’t affect signature validity!
<NameID>
admin@victim.com<!--inserted by attacker-->.evil.com
</NameID>
Truncated by the SP!

Using **my** credentials, I can pivot to **other** user accounts.

Finding Affected Systems and Disclosure

Finding Vulnerable Systems

- To detect presence of vulnerability, we often wrote or modified unit-tests that did roughly the following:

```
doc = "[...]<NameID>us<!---->er</NameID>[...]"
doc.verifySignature()
assert doc.nameID == "user"
# if assertion failed, NameID was possibly truncated
```

- If adding comments to those test cases caused a failure it was a strong indicator that the library was vulnerable.

Affected Libraries

- Initial research identified four affected open source libraries:
  - **CVE-2017-11427** - OneLogin's "python-saml"
  - **CVE-2017-11428** - OneLogin's "ruby-saml"
  - **CVE-2017-11429** - Clever's "saml2-js"
  - **CVE-2017-11430** - "OmniAuth-SAML"

- Others who self-reported vulnerablity during disclosure:
  - **CVE-2018-0489** - Shibboleth openSAML C++

Known Affected Products

- **Duo Network Gateway** (DNG) prior to 1.2.10.
- Multiple **Pulse Secure** products prior to 8.3R6 and 9.0R2.
- **Gitlab** prior to 10.7.0
- **Shibboleth** prior to 2.6.1
- **Symantec Advanced Secure Gateway** and **ProxySG** remain affected according to their advisory.

Disclosure and Research Timeline

2017-12-11 Vulnerability identified during internal audit. RCA suggests others libraries could be affected.

2017-12-14 Analysis of SAML implementations identifies three other vendors impacted by same vulnerability class.

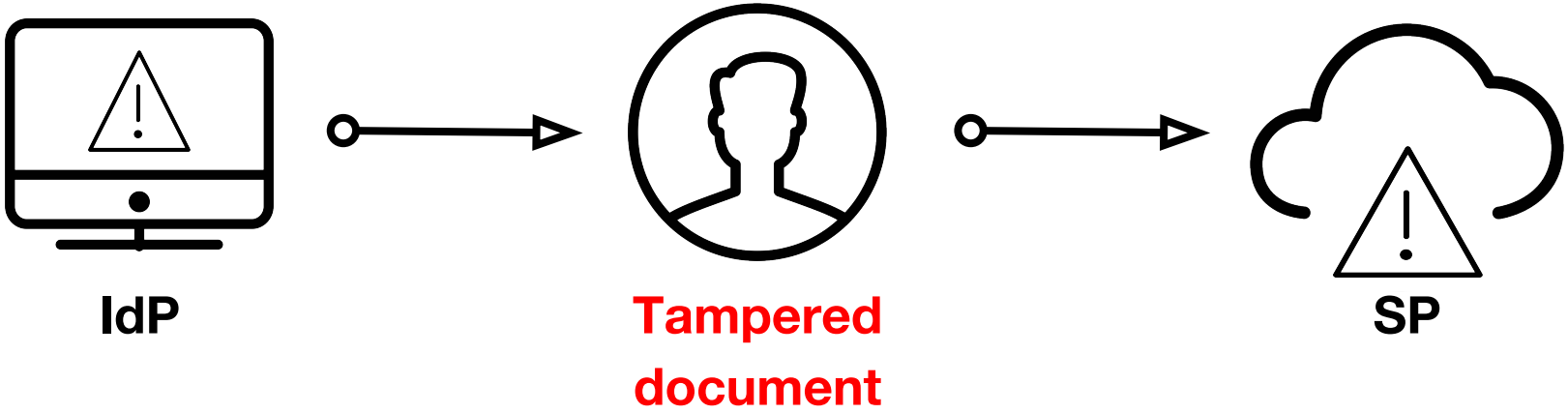
2017-12-18 CERT/CC contacted to coordinate disclosure.

2018-02-27 Coordinated public disclosure of all affected systems.

Exploitation of Service Providers

Threat Model

- Attacker has account on Identity Provider
- Attacker uses foothold to gain access to Service Providers
- Attacker uses the XML comment bug to pivot into Service Providers for *different users*



SP Exploitability: SAML Authorization Info

- Does the SAML Response convey role information?
- SAML defines the AttributeStatement element which can contain fairly generic attribute values.
- This can be used to define role information:

```
<AttributeStatement>
  <Attribute Name="Groups">
    <AttributeValue>Administrators (HR)</AttributeValue>
    <AttributeValue>Employees</AttributeValue>
  </Attribute>
</AttributeStatement>
```

SP Exploitability: User Identifiers

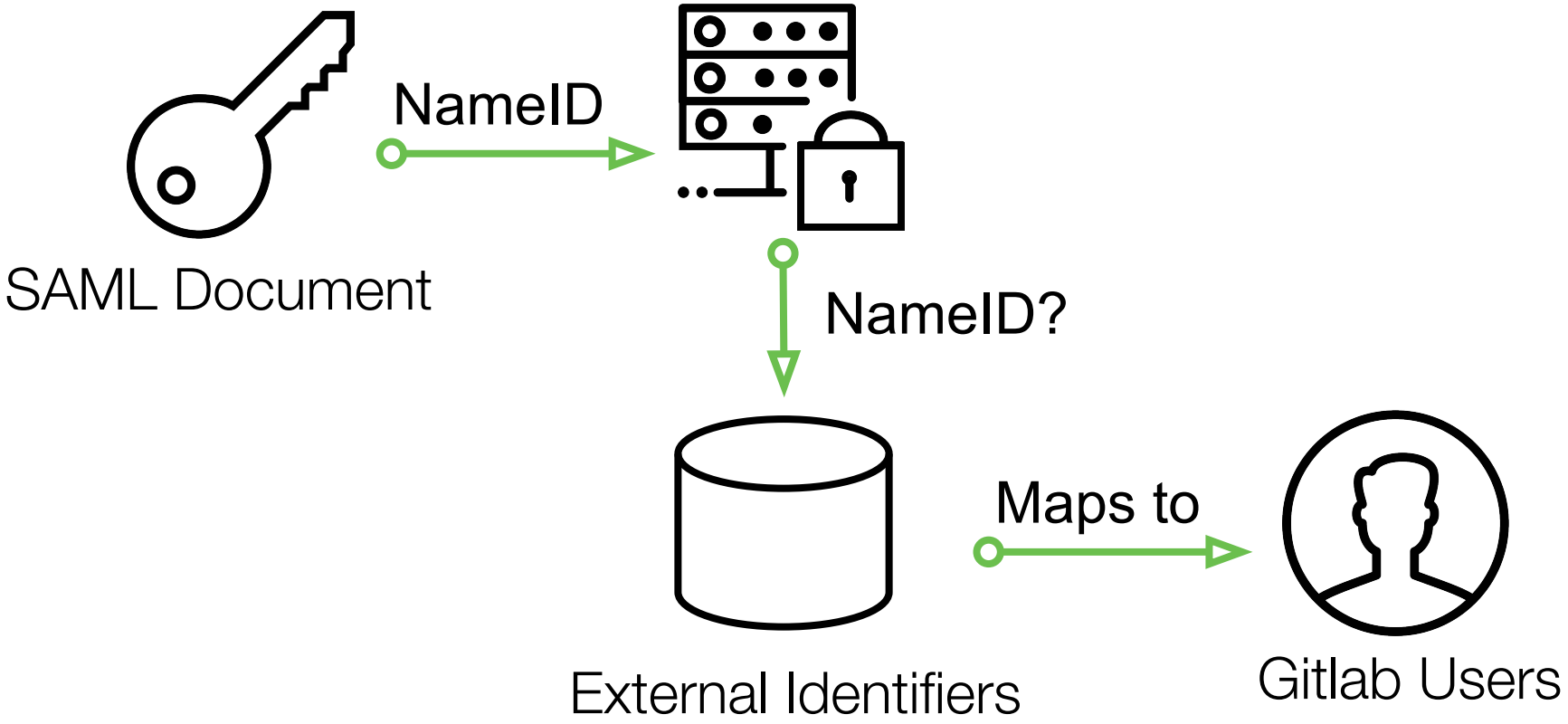
- SAML doesn't dictate the format of a user identifier.
- What format of identifiers are used by the Service Provider?

Format	Example	Opportunities for Truncation?
Random	`8f4de25fe6`	Limited
Numeric	`user_104`	Yes!
Email	`kelby@duo.com`	Registration-dependent
Usernames	`kelbz`	Registration-dependent

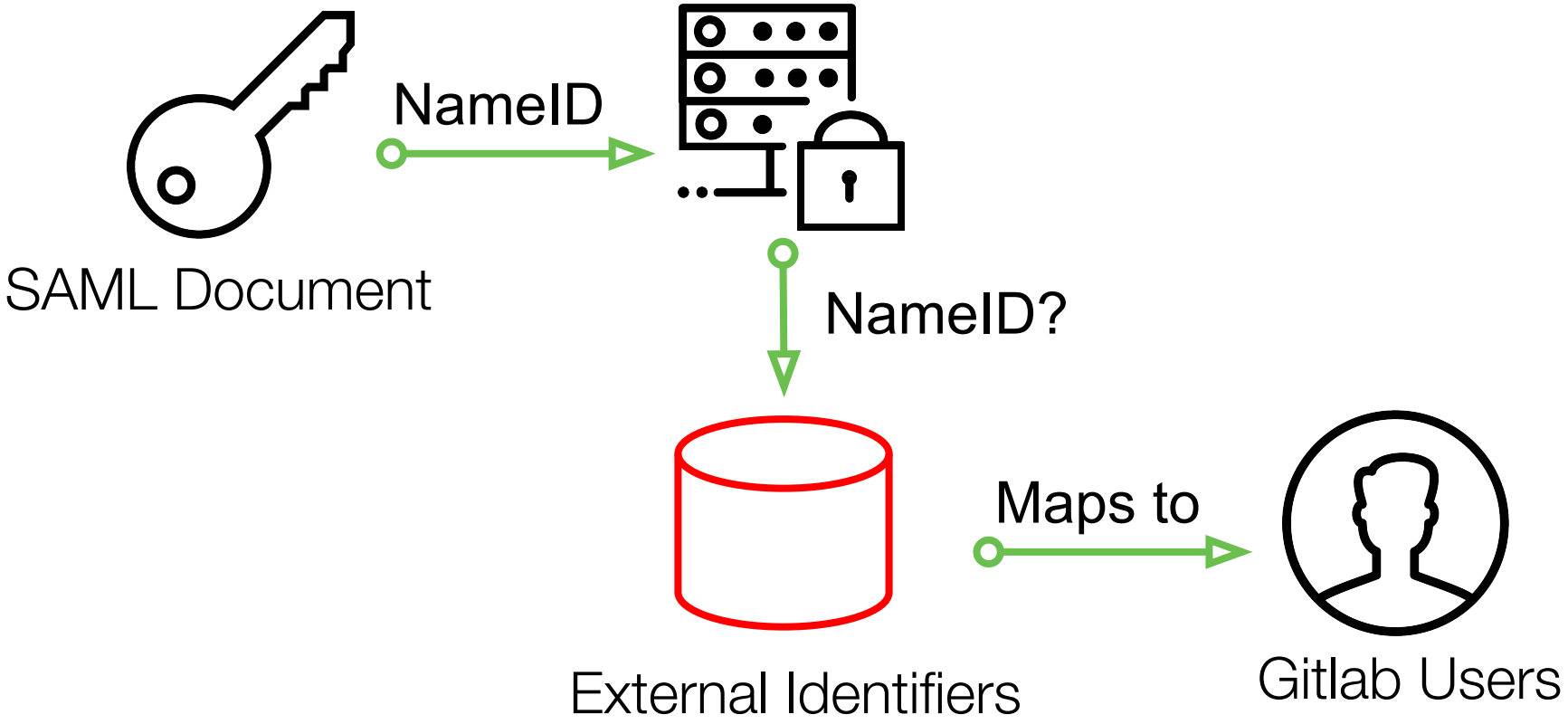
Case Study: Gitlab as an Exploitable SP

- Gitlab is an open source source code management service.
- Prior to Gitlab CE 10.7.0, Gitlab's SAML service provider implementation was affected by the comment truncation vulnerability.
- Using the comment truncation vulnerability to cause Gitlab to process one external ID as the target user's external ID allows you to authN as that user

Gitlab User Identifier Mapping for SAML



Gitlab User Identifier Mapping for SAML



Example Exploitation of Gitlab

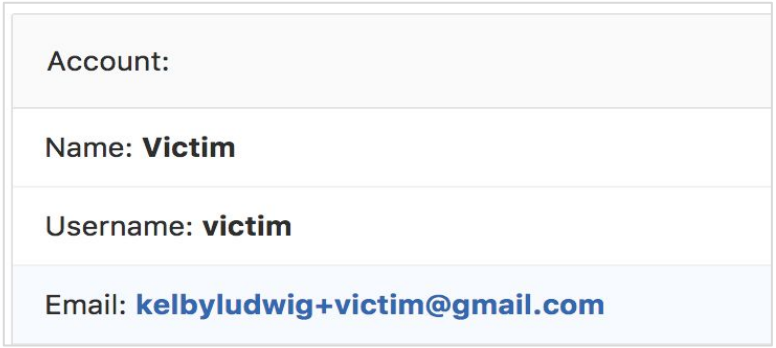
Gitlab username:

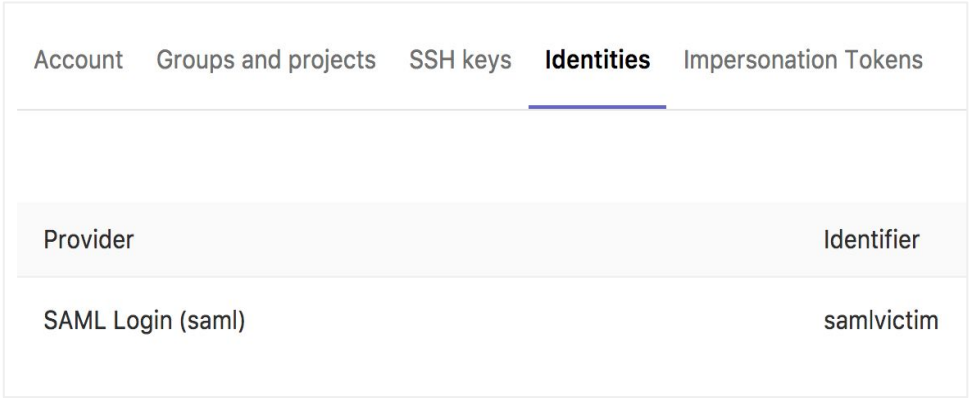
`victim`

External identifier:

`samlvictim`

Goal: Truncate our external identifier to collide with the victim user's external identifier. *Not* the Gitlab username.





Example Exploitation of Gitlab

A SAML document for the victim contains their external identity (`samlvictim`) and email address to keep it in sync with the IdP.

```
<saml2:Subject xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">samlvictim</saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
```

```
<saml2:AttributeValue
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">kelbyludwig+victim@gmail.com</saml2:AttributeValue>
```

Example Exploitation of Gitlab

<NameID>samlvictim<!---->123</NameID>

```
<saml2:Subject xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">samlvictim<!---->123</saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml2:SubjectConfirmationData
      NotOnOrAfter="2018-01-19T03:07:41.575Z" Recipient="http://gitlab.evil.com/users/auth/saml/callback"/>
  </saml2:SubjectConfirmation>
</saml2:Subject>
```

Name

Victim

User ID

10

Enter your name, so people you know can recognize you.

Email

kelbyludwig+different@gmail.com

Your email address was automatically set based on your SAML Login account.

Remediation for...

IT Organizations

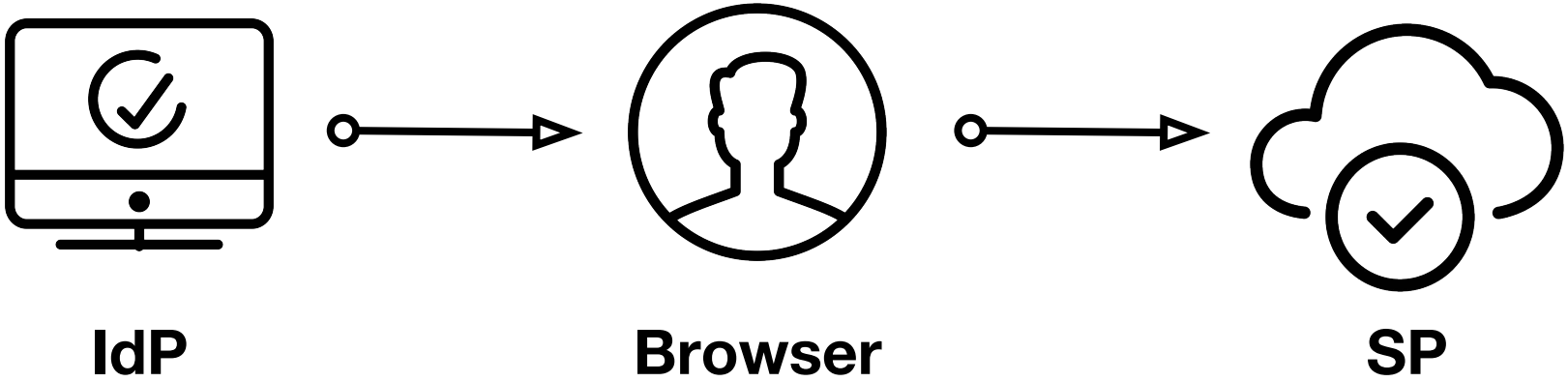
- Test your service providers for truncation vulnerabilities
- Ask service providers you rely on if they have tested themselves

SAML Developers

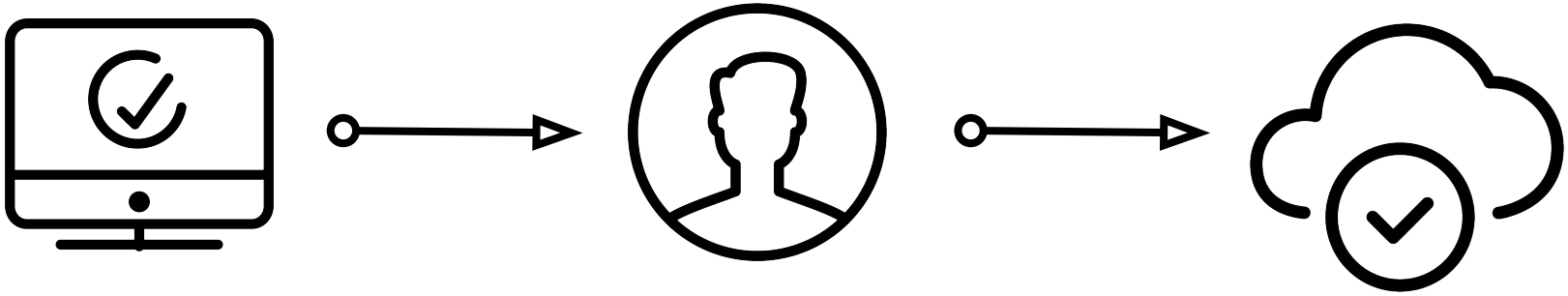
- Test your SAML code
- Reject non-c14n'd documents so further c14n is not necessary
- Reject SAML documents with comments

Exploitation of Identity Providers

IdP Exploitability: 2FA



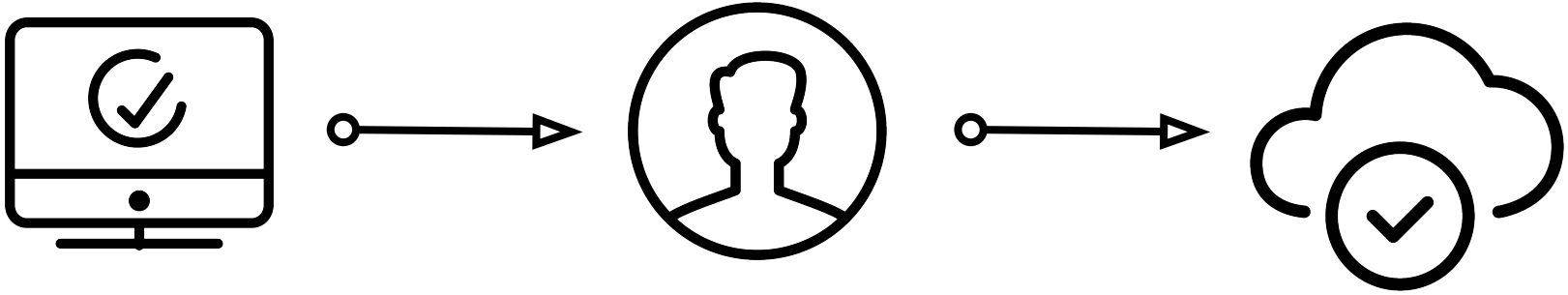
IdP Exploitability: 2FA



2FA only here?

Bypass!

IdP Exploitability: 2FA

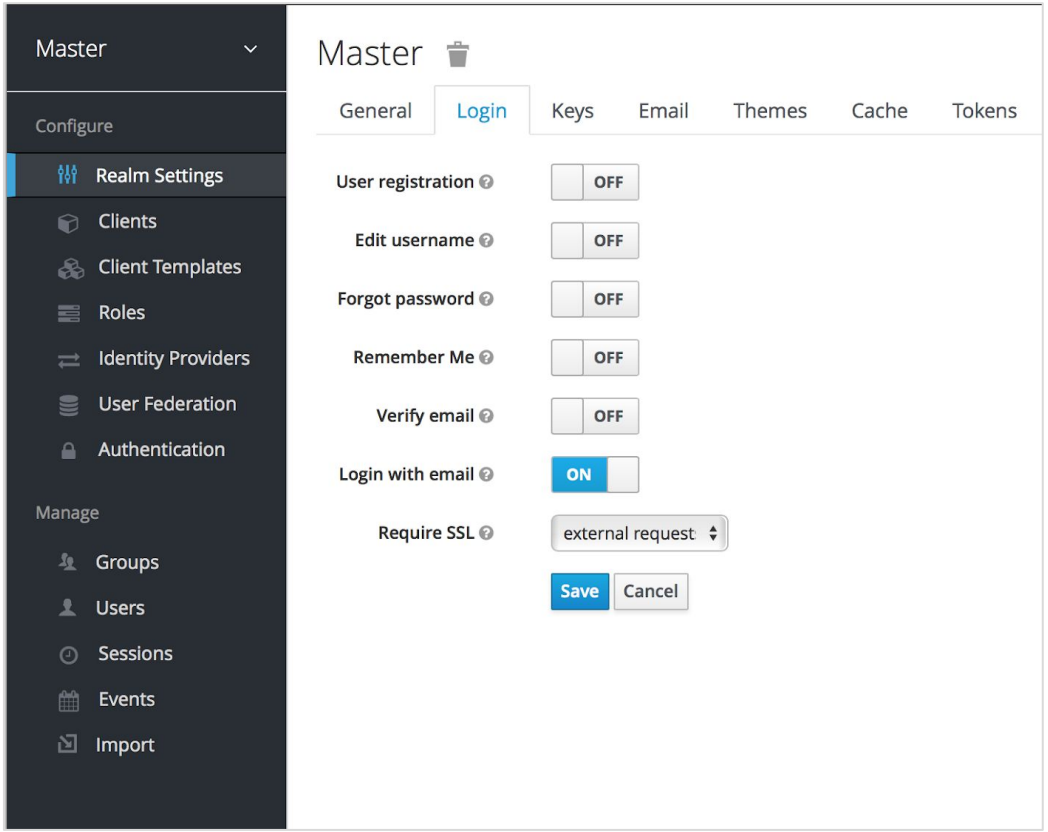


2FA only here?

No bypass!

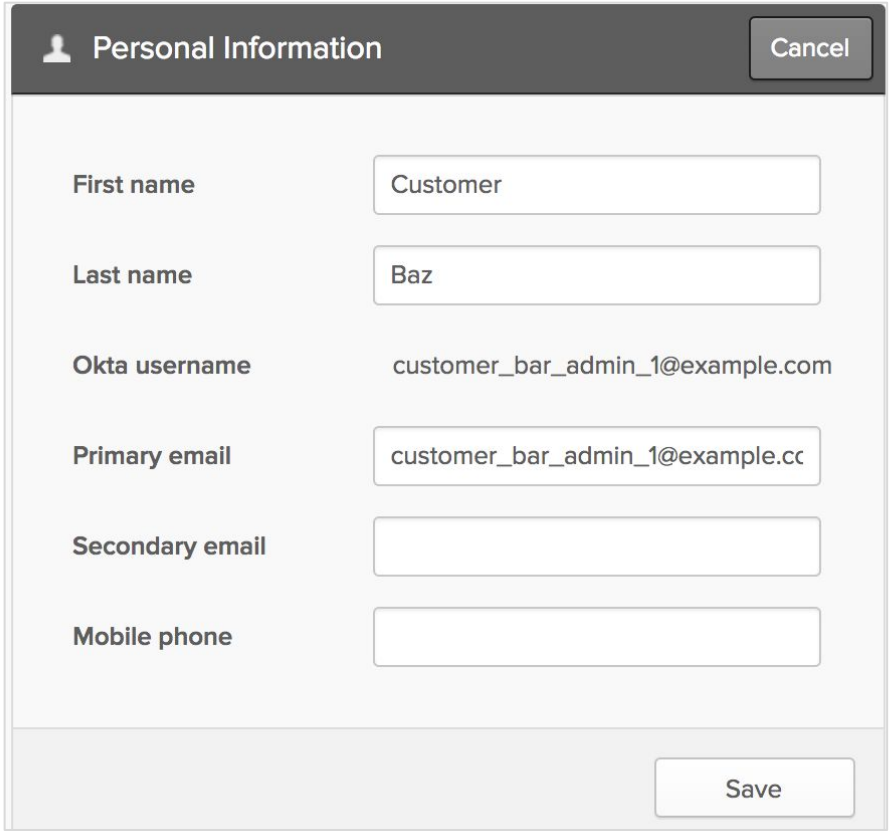
IdP Exploitability: User Registration

- Some IdPs provide self-registration portals for their IdP.
- A persistent registration link may linger in someone's inbox forever! Free user-selected identities!



IdP Exploitability: User Profile Management

- Some IdPs also function as employee-facing directories.
- Employee profiles often can be updated by the employee.
- **Mutable Identity:** A term to describe Identity Providers who conflate user-controlled directory information with SSO identity.



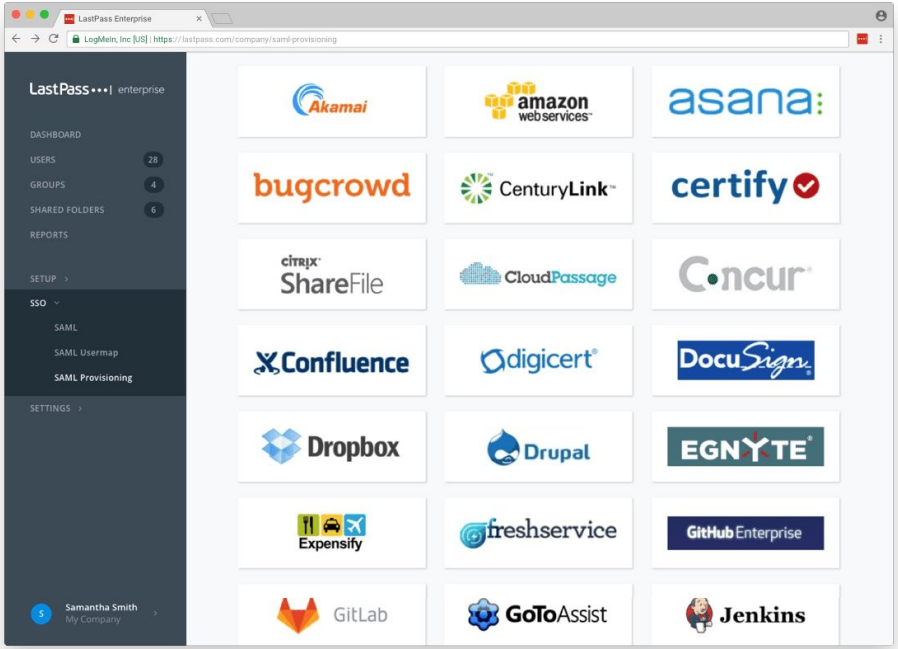
Mutable Identity

Mutable Identity means users can influence their SSO identity.

Mutable Identity increases the impact of comment truncation vulnerabilities.

Mutable Identity: LastPass

- LastPass Enterprise has a SSO feature that has mutable identity.
- NameID can be tied to email address, which can be updated by a end user.
- ***Not a vulnerability on its own!*** But very risky when combined with vulnerable SPs.



Exploiting SPs With Mutable Identity

1. Identity a SP vulnerable to comment truncation vulnerability.
2. Update my LastPass Enterprise account email to target Duo's CTO email:
   `jono@duo.com.attacker.com`
3. Re-authenticate to the SP, and insert a comment into your own NameID:

   `jono@duo.com<!---->.attacker.com`

Mutable Identity could enable authN bypasses *without a vulnerable SP*

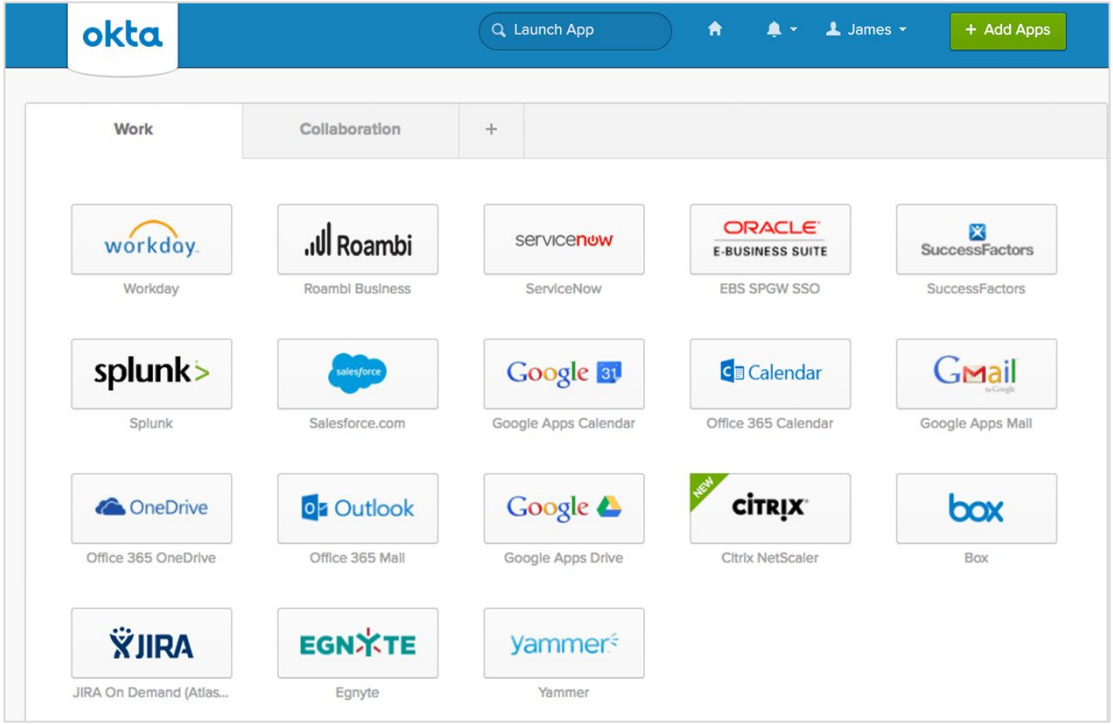
What if I could change ***my*** profile to match ***your*** profile?

What if I could change ***my*** profile to match ***your*** profile?

No truncation needed!

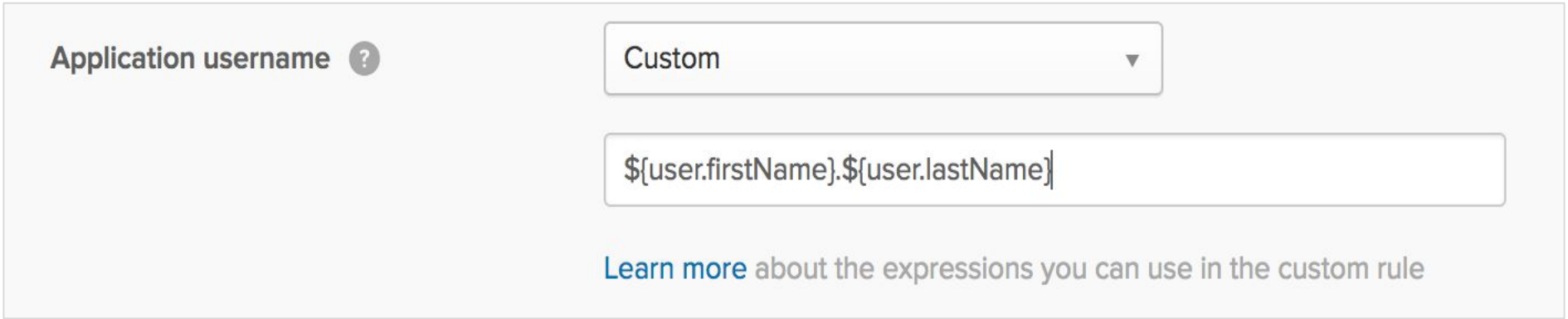
Mutable Identity: Okta

- Okta is an IdP that gives users a UI of all SPs they can access.
- Applications could be configured where user's could *completely control* their SSO identity for an application.



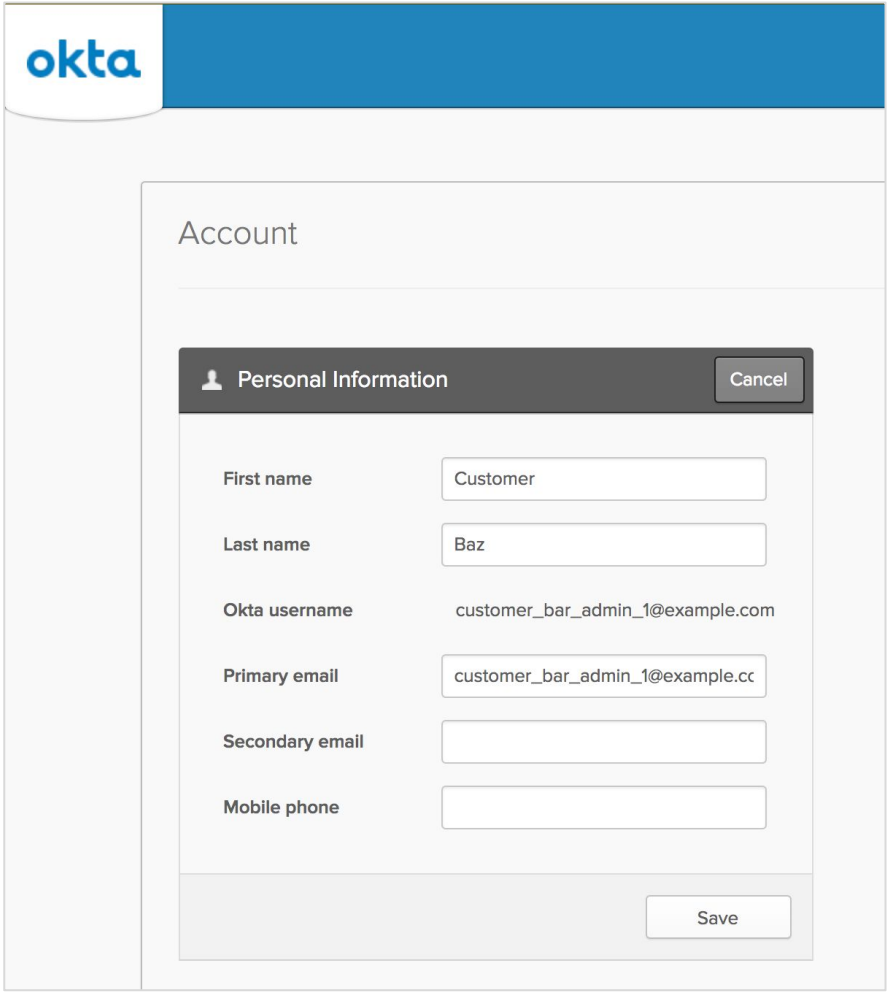
Mutable Identity: Okta

- Okta allows you to "program" identities for SAML SPs.
- In this example, my account would have SAML assertions with NameID "kelby.ludwig"



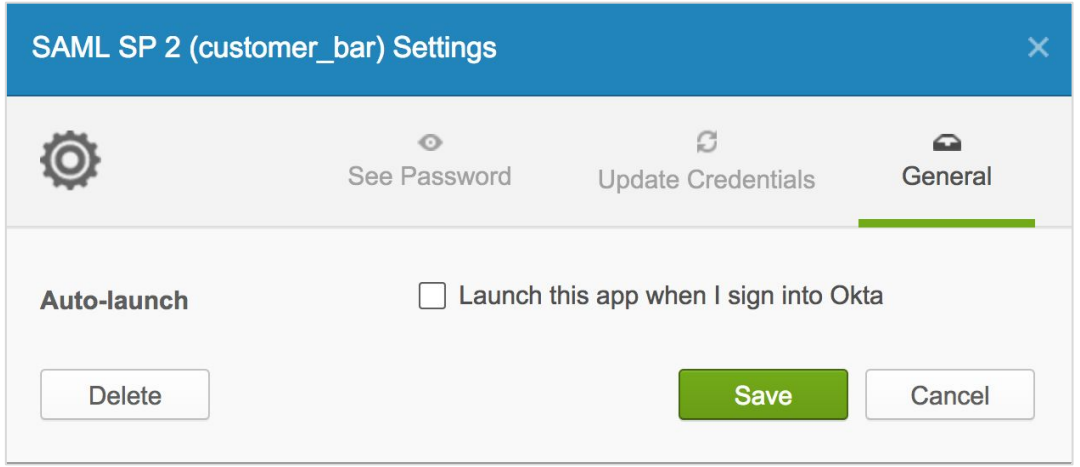
Mutable Identity: Okta

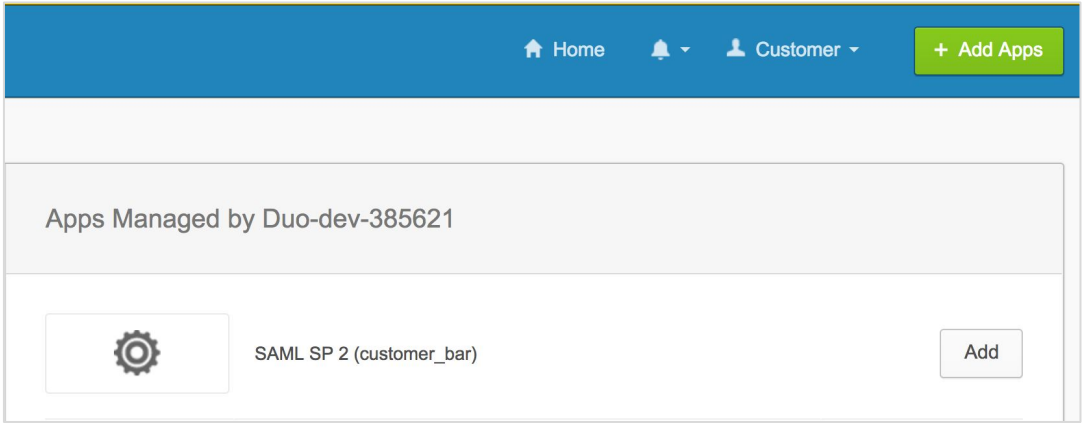
- Okta also provides user profile information.
- The editable values of first name and last name are the same values used as part of the user field mapping for SAML assertions.



Mutable Identity: Okta

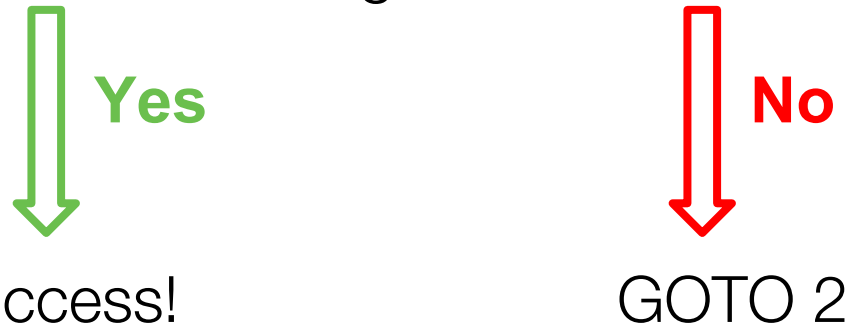
- Okta didn't calculate identities on every authentication.
- However, Okta provides a self-service application provisioning portal for users





Identifying and Exploiting Mutable Identity

1. Update profile to match a target user
2. De-provision target application using self-service portal
3. Re-provision target application using self-service portal
4. Authenticate to target application
5. SAML document uses target user information?



Remediation

- Self-remediation: confirm programmable user expressions are not using user-writable properties like first or last name.
- Okta's plan was to make these user profile attributes read-only by-default for new organizations (March 2018)
  - Existing organizations would not be changed.
- Still possible to opt-into mutable identities albeit but requires more work to configure to stumble across than before.

Conclusion

Takeaways

- Despite years of research, the complexity of SAML and related standards still makes systems built on them interesting targets
- We could only look at so many SAML implementations. Not all SAML implementations are directly accessible.

Thanks to...

- CERT/CC for coordinating disclosure with many vendors.
- Duo Labs researchers who hunted down vulnerable libraries.
- Affected vendors who provided patches to their users.

Thank you!

Twitter: @kelbyludwig
kludwig@duo.com
https://kel.bz

DUO LABS