GitHub Security Lab Meetup (Jan 2020)

Breaking SAML (.NET Edition)

Presented by Alvaro Muñoz (@pwntester)

> whoarewe

Alvaro Muñoz

- Security Researcher @ GitHub Security Lab
- Twitter: @Pwntester

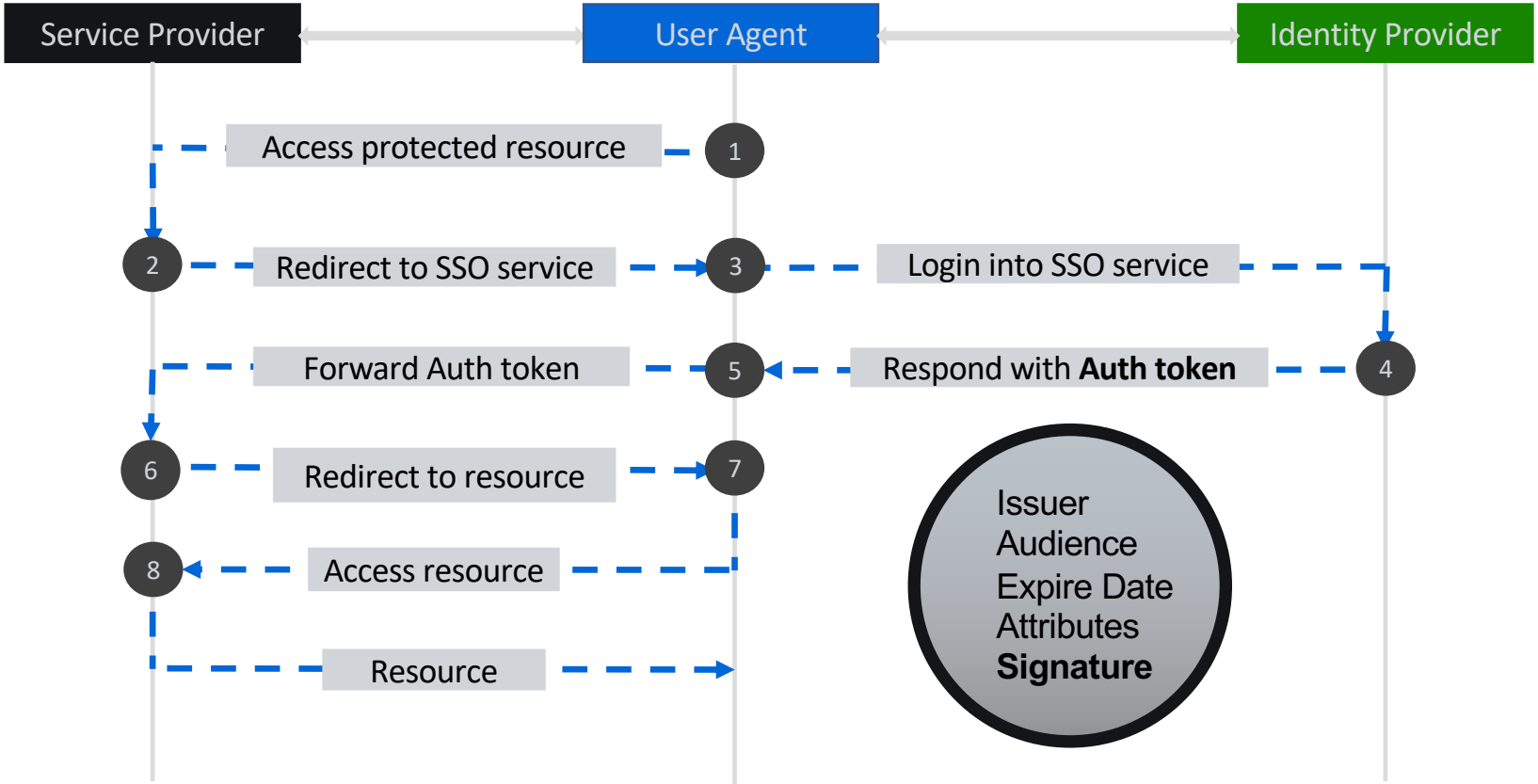
Oleksandr Mirosh (In absentia)

- Security Researcher @ Micro Focus Fortify
- Twitter: @OlekMirosh

Previously in Breaking SAML …

- **XML Signature Wrapping (XSW):**
  - Discovered in 2012 by Juraj Somorovsky, Andreas Mayer and others
  - Many implementations in different languages were affected
  - The attacker needs access to a valid token
  - The attacker modifies the contents of the token by injecting malicious data without invalidating the signature
- **Attacks with XML comments:**
  - Discovered in 2018 by Kelby Ludwig
  - The attacker needs access to a valid token
  - Uses XML comments to modify values without invalidating the signature

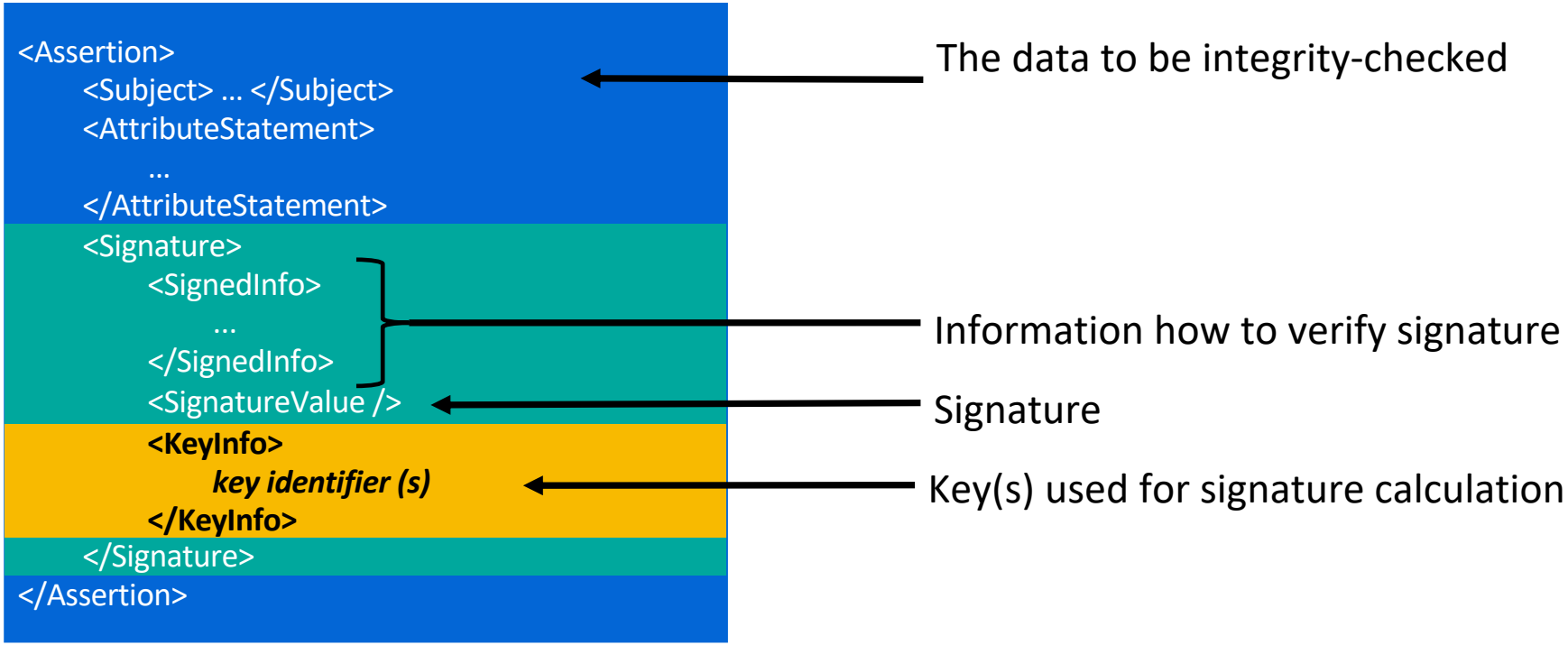
SAML 101


Service Provider
User Agent
Identity Provider
Access protected resource
1
2
Redirect to SSO service
3
Login into SSO service
4
Forward Auth token
5
Respond with **Auth token**
6
Redirect to resource
7
8
Access resource
Resource
Issuer
Audience
Expire Date
Attributes
Signature

SAML 101

- The Security Assertion Markup Language, SAML:
  - Popular standard used in single sign-on systems
  - XML-based format
  - Uses XML Signature (aka XMLDSig) standard
- XMLDSig standard (RFC 3275):
  - Used to provide payload security in SAML, SOAP, WS-Security, etc.

Simplified SAML Token



RFC 3275

4.4 The KeyInfo Element

KeyInfo is an optional element that enables the recipient(s) to obtain the key needed to validate the signature. KeyInfo may contain keys, names, certificates and other public key management information, such as in-band key distribution or key agreement data. This specification defines a few simple types but applications may extend those types or all together replace them with their own key identification and exchange semantics using the XML namespace facility. [XML-ns] However, questions of trust of such key information (e.g., its authenticity or strength) are out of scope of this specification and left to the application.

If KeyInfo is omitted, the recipient is expected to be able to identify the key based on application context. Multiple declarations within KeyInfo refer to the same key. While applications may define and use any mechanism they choose through inclusion of elements from a different namespace, compliant versions MUST implement KeyValue (section 4.4.2) and SHOULD implement RetrievalMethod (section 4.4.3).

RFC 3275

4.4 The KeyInfo Element

KeyInfo is an optional element that enables the recipient(s) to obtain the key needed to validate the signature. KeyInfo may contain keys, names, certificates and other public key management information, such as in-band key distribution or key agreement data. This specification defines a few simple types but applications may extend those types or all together replace them with their own key identification and exchange semantics using the XML namespace facility. [XML-ns] **However, questions of trust of such key information (e.g., its authenticity or strength) are out of scope of this specification and left to the application.**

If KeyInfo is omitted, the recipient is expected to be able to identify the key based on application context. Multiple declarations within KeyInfo refer to the same key. While applications may define and use any mechanism they choose through inclusion of elements from a different namespace, compliant versions MUST implement KeyValue (section 4.4.2) and SHOULD implement RetrievalMethod (section 4.4.3).

RFC 3275

4.4 The KeyInfo Element

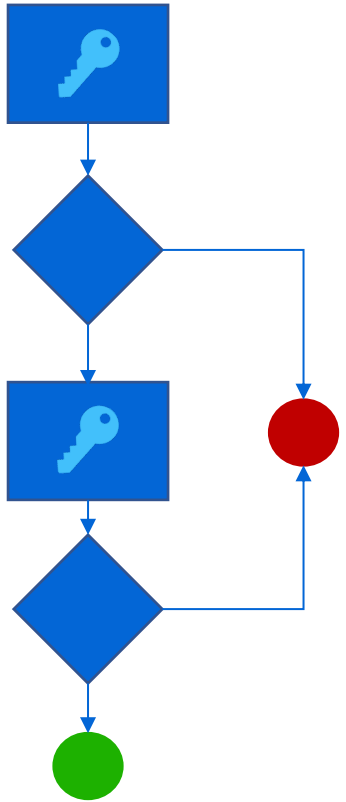
KeyInfo is an optional element that enables the recipient(s) to obtain the key needed to validate the signature. KeyInfo may contain keys, names, certificates and other public key management information, such as in-band key distribution or key agreement data. This specification defines a few simple types but applications may extend those types or all together replace them with their own key identification and exchange semantics using the XML namespace facility. [XML-ns] However, questions of trust of such key information (e.g., its authenticity or strength) are out of scope of this specification and left to the application.

If KeyInfo is omitted, the recipient is expected to be able to identify the key based on application context. **Multiple declarations within KeyInfo refer to the same key.** While applications may define and use any mechanism they choose through inclusion of elements from a different namespace, compliant versions MUST implement KeyValue (section 4.4.2) and SHOULD implement RetrievalMethod (section 4.4.3).

Token Integrity Verification

1. Verify the signature
   a) Resolve the signing key
      - obtain `SecurityKey` from `<KeyInfo/>`
      - or create it from embedded data
   b) Use key to verify signature

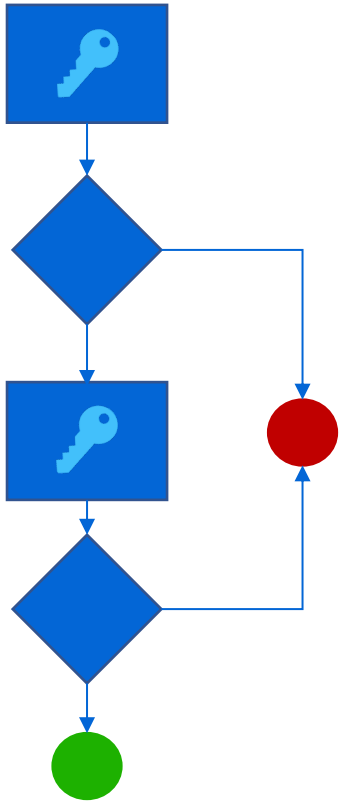
2. Verify the signing party
   a) Identify the signing party
      - obtain `SecurityToken` from `<KeyInfo/>`
   b) Authenticate the signing party
      - verify trust on `SecurityToken`



Token Integrity Verification

1. Verify the signature
   a) Resolve the signing key
      - obtain `SecurityKey` from `<KeyInfo/>`
      - or create it from embedded data
   b) Use key to verify signature

2. Verify the signing party
   a) Identify the signing party
      - obtain `SecurityToken` from `<KeyInfo/>`
   b) Authenticate the signing party
      - verify trust on `SecurityToken`



A tale of two resolvers

- `<KeyInfo/>` is processed **twice** by different methods!

- If we can get each method to return different keys, we may be able to bypass validation

Scenarios for different key resolution

1. Method *A* supports a key type that is not supported by Method *B*
2. Methods check for different subsets of keys within the `<KeyInfo/>` section
3. Both methods support same key types, but in different order

Examples of affected frameworks

Windows Communication Foundation (WCF)

- Used in Web Services
- Eg: Exchange server

Windows Identity Foundation (WIF)

- Used in claim-aware applications
- Eg: MVC application authenticating users with ADFS or Azure Active Directory

Windows Identity Foundation (WIF) + Custom configuration

- Uses custom configuration such as a custom resolver or custom certificate store
- Eg: SharePoint Server

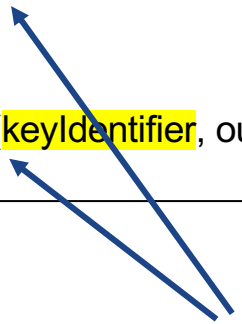
Windows Communication Foundation (WCF)

Windows Communication Foundation

- Framework for building service-oriented applications (SOA)
- Interaction between WCF endpoint and client is done using SOAP envelopes
- WCF accepts SAML tokens as Client credentials
- May use Windows Identity Foundation (WIF) or not
- XML Signature also used for proof tokens and other usages

Key & Token Resolution

```
// System.IdentityModel.Tokens.SamlAssertion
SecurityKeyIdentifier keyIdentifier = signedXml.Signature.KeyIdentifier;
this.verificationKey = SamlSerializer.ResolveSecurityKey(keyIdentifier, outOfBandTokenResolver);
if (this.verificationKey == null) throw ...
this.signature = signedXml;
this.signingToken = SamlSerializer.ResolveSecurityToken(keyIdentifier, outOfBandTokenResolver);
```



Same <keyInfo/> block is processed twice

Security Key resolution

```
// System.IdentityModel.Tokens.SamlSerializer
internal static SecurityKey ResolveSecurityKey(SecurityKeyIdentifier ski, SecurityTokenResolver
tokenResolver)
{
    if (ski == null) throw DiagnosticUtility.ExceptionUtility.ThrowHelperArgumentNull("ski");
    if (tokenResolver != null) {
        for (int i = 0; i < ski.Count; i++) {
            SecurityKey result = null;
            if (tokenResolver.TryResolveSecurityKey(ski[i], out result)) {
                return result;
            }
        }
    }
...
```

For each <KeyInfo/> element, try ALL resolvers, until one is successful

Security Key resolution

Remember, one key at a time!

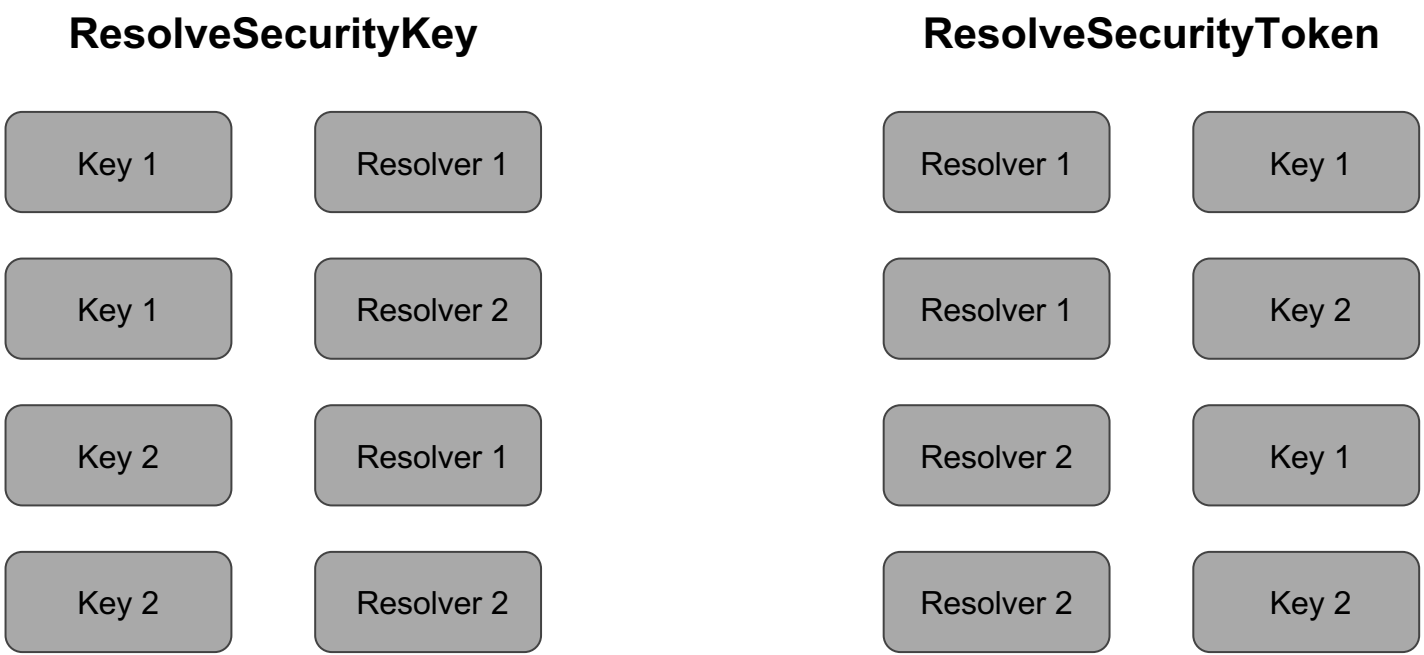
```
// System.ServiceModel.Security.AggregateSecurityHeaderTokenResolver
bool TryResolveSecurityKeyCore(SecurityKeyIdentifierClause keyIdentifierClause, out SecurityKey key) {
        ...

        resolved = this.tokenResolver.TryResolveSecurityKey(keyIdentifierClause, false, out key);
        if (!resolved)
            resolved = base.TryResolveSecurityKeyCore(keyIdentifierClause, out key);
        if (!resolved)
            resolved = SecurityUtils.TryCreateKeyFromIntrinsicKeyClause(keyIdentifierClause, this, out key);
```

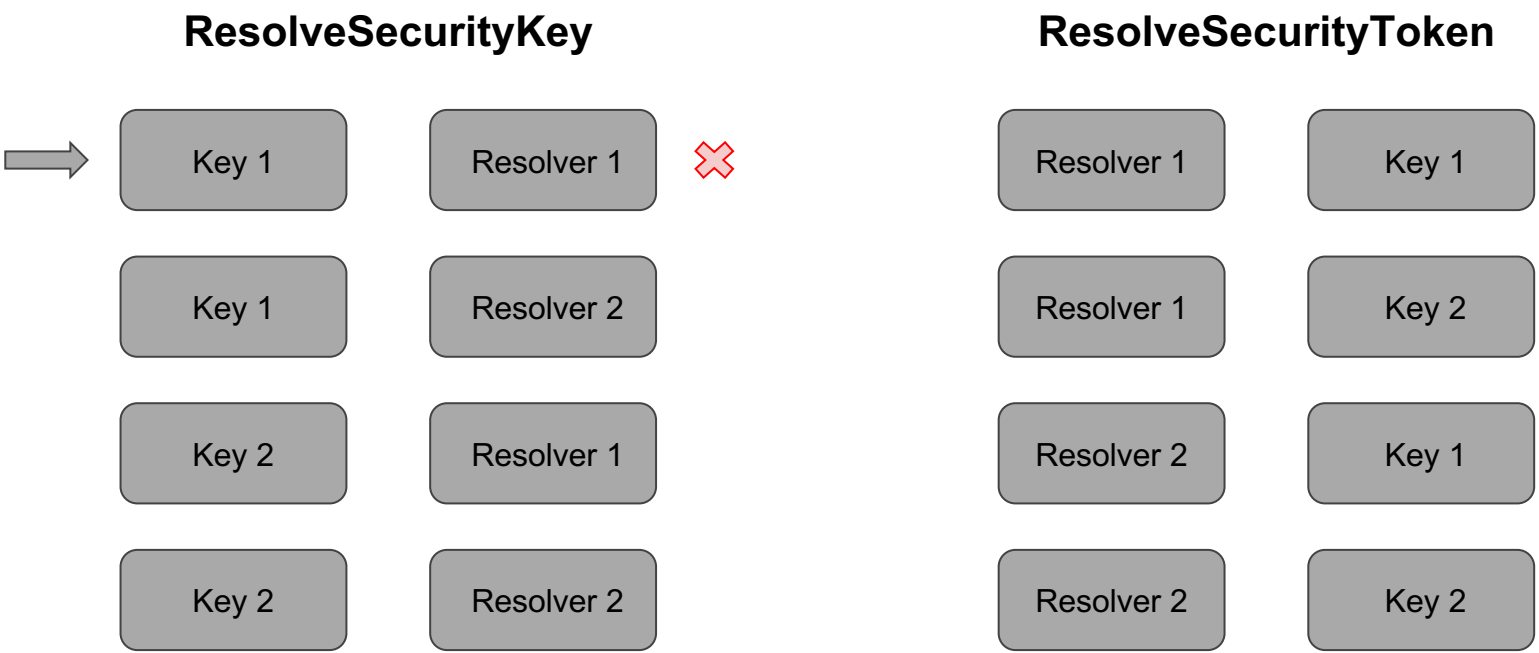
Token resolution

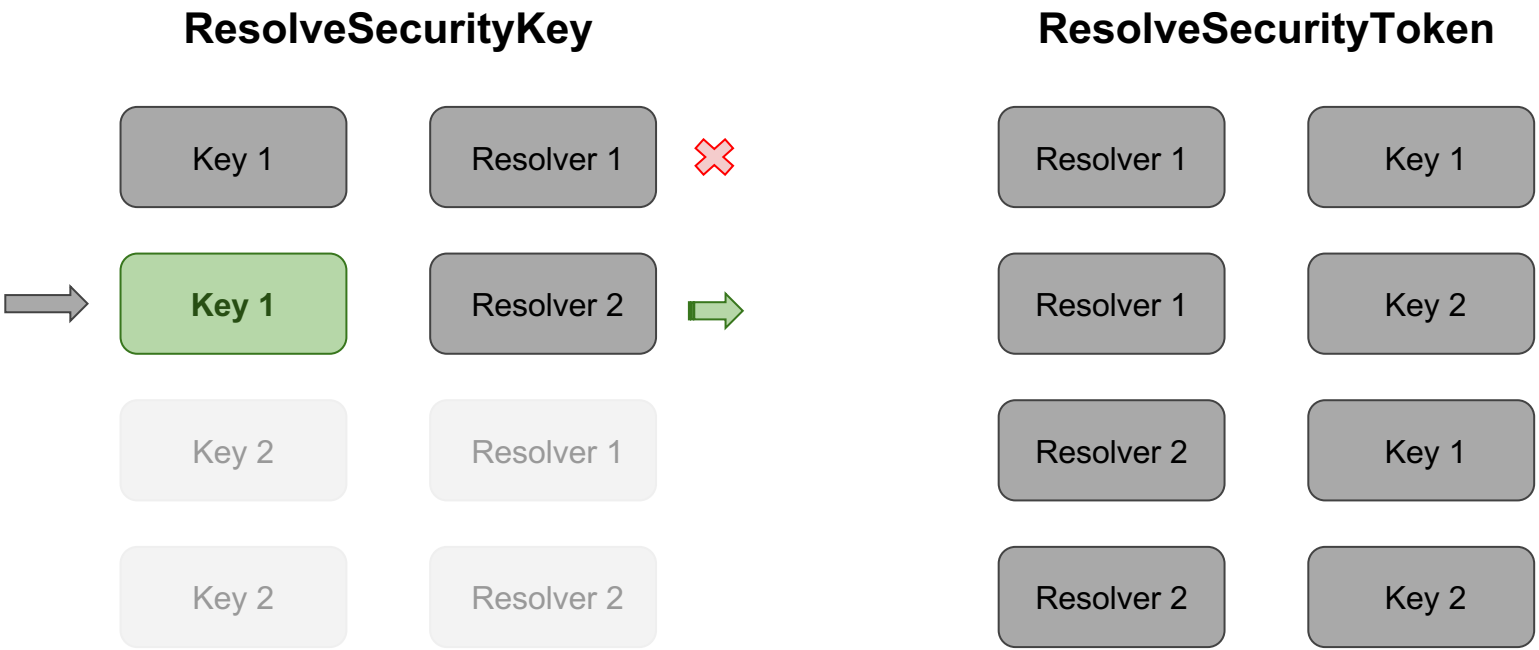
```
// System.ServiceModel.Security.AggregateSecurityHeaderTokenResolver
override bool TryResolveTokenCore(SecurityKeyIdentifier keyIdentifier, out SecurityToken token) {
    bool resolved = false;
    token = null;
    resolved = this.tokenResolver.TryResolveToken(keyIdentifier, false, false, out token);
    if (!resolved) resolved = base.TryResolveTokenCore(keyIdentifier, out token);
    if (!resolved) {
        for (int i = 0; i < keyIdentifier.Count; ++i) {
            if (this.TryResolveTokenFromIntrinsicKeyClause(keyIdentifier[i], out token)) {
                resolved = true;
                break;
            }
```

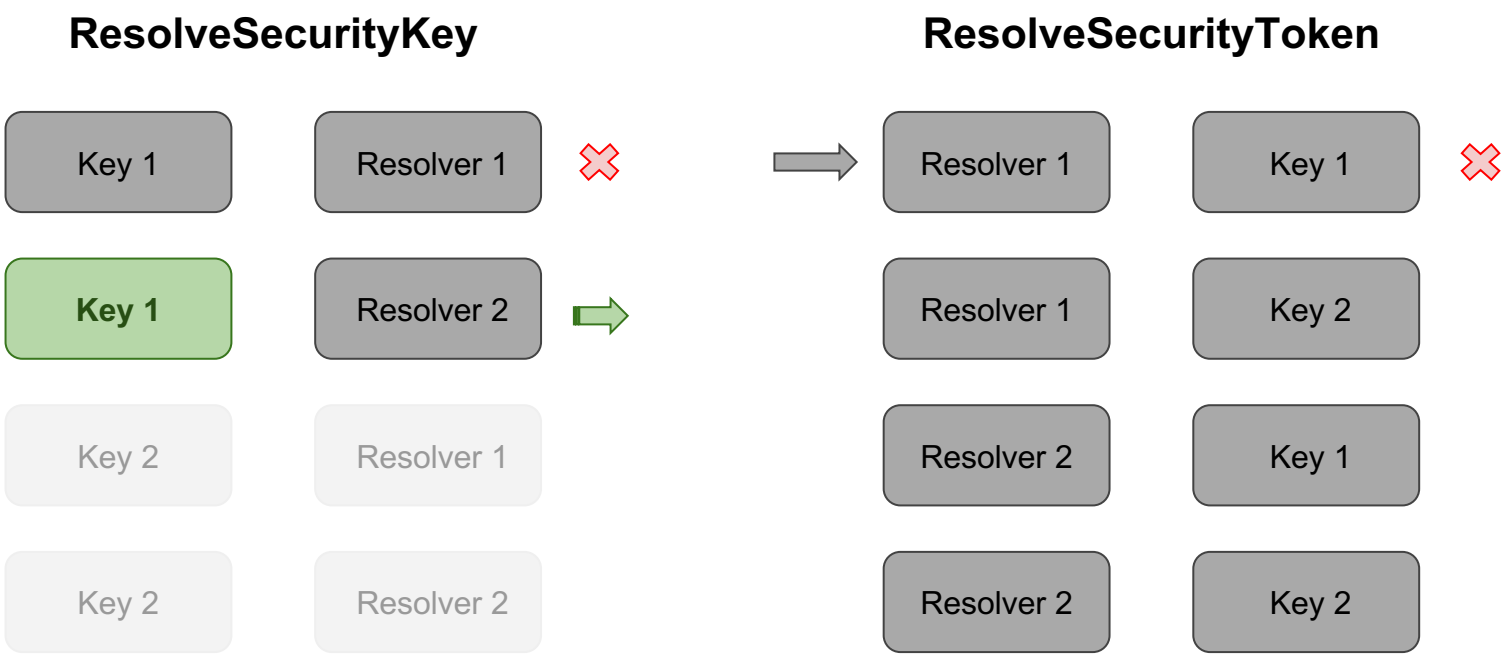
Remember, ALL keys are passed here!

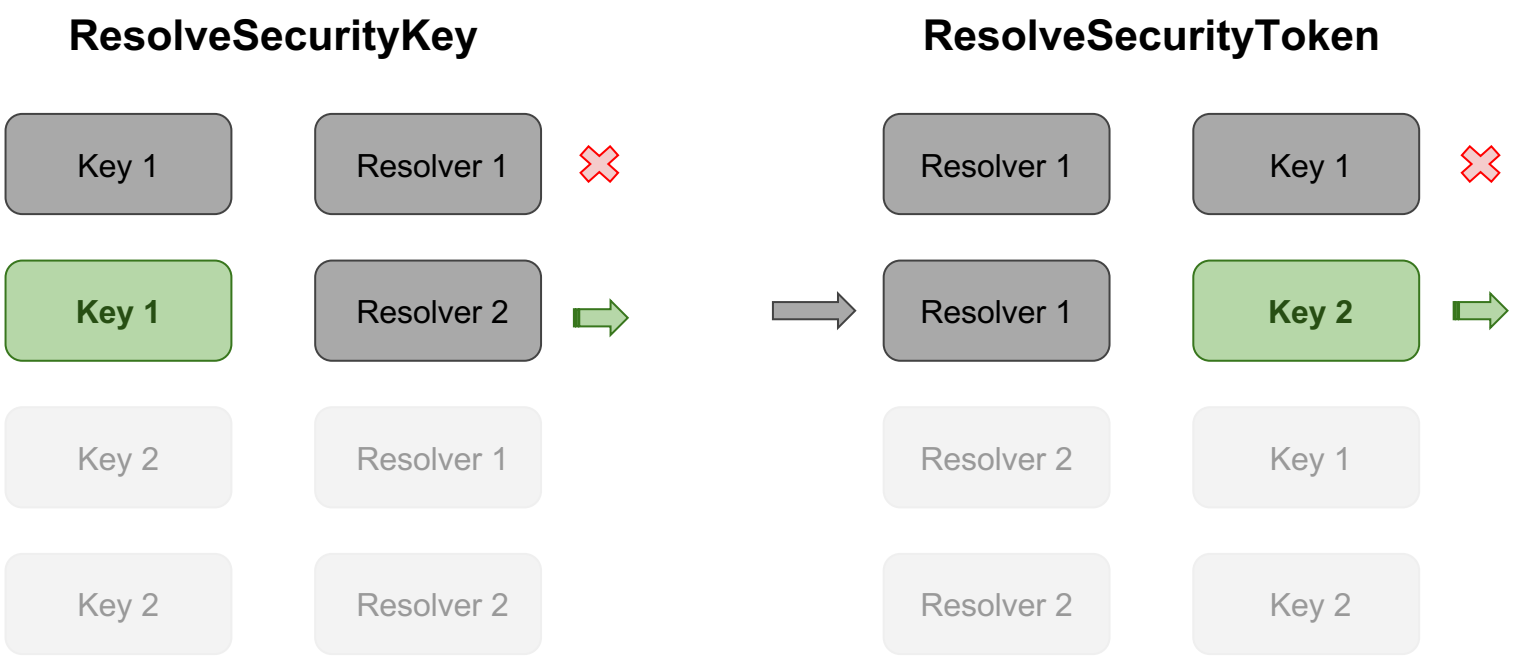
For each token resolver, try ALL <keyInfo/> elements, until one is successful


ResolveSecurityKey
Key 1
Resolver 1
Key 1
Resolver 2
Key 2
Resolver 1
Key 2
Resolver 2
ResolveSecurityToken
Resolver 1
Key 1
Resolver 1
Key 2
Resolver 2
Key 1
Resolver 2
Key 2


ResolveSecurityKey
Key 1
Resolver 1
Key 1
Resolver 2
Key 2
Resolver 1
Key 2
Resolver 2
ResolveSecurityToken
Resolver 1
Key 1
Resolver 1
Key 2
Resolver 2
Key 1
Resolver 2
Key 2

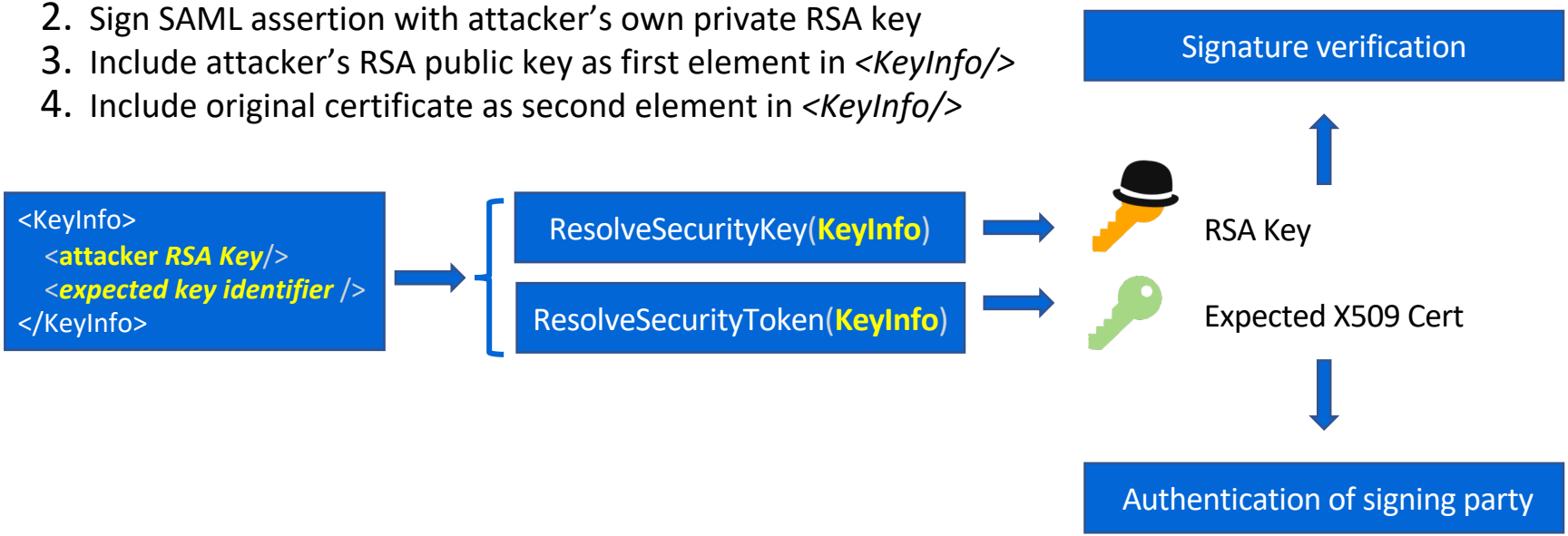

ResolveSecurityKey
Key 1
Resolver 1
Key 1
Resolver 2
Key 2
Resolver 1
Key 2
Resolver 2
ResolveSecurityToken
Resolver 1
Key 1
Resolver 1
Key 2
Resolver 2
Key 1
Resolver 2
Key 2


ResolveSecurityKey
Key 1
Resolver 1
Key 1
Resolver 2
Key 2
Resolver 1
Key 2
Resolver 2
ResolveSecurityToken
Resolver 1
Key 1
Resolver 1
Key 2
Resolver 2
Key 1
Resolver 2
Key 2


ResolveSecurityKey
Key 1
Resolver 1
Key 1
Resolver 2
Key 2
Resolver 1
Key 2
Resolver 2
ResolveSecurityToken
Resolver 1
Key 1
Resolver 1
Key 2
Resolver 2
Key 1
Resolver 2
Key 2

Dupe Key Confusion

1. Modify token at will or create token from scratch
2. Sign SAML assertion with attacker's own private RSA key
3. Include attacker's RSA public key as first element in *<KeyInfo/>*
4. Include original certificate as second element in *<KeyInfo/>*



Dupe Key Confusion

```
<ds:KeyInfo>
    <ds:KeyValue>                                              Injected Key
        <ds:RSAKeyValue>
            <ds:Modulus>irXhaxafoUZ...77kw==</ds:Modulus>
            <ds:Exponent>AQAB</ds:Exponent>
        </ds:RSAKeyValue>
    </ds:KeyValue>
    <ds:X509Data>
        <ds:X509Certificate>MIIDBTCCAe2...zzWh</ds:X509Certificate>
    </ds:X509Data>                                             Original Cert
</ds:KeyInfo>
```

https://github.com/pwntester/DupeKeyInjector

Search or jump to... / Pull requests Issues Marketplace Explore

pwntester / DupeKeyInjector

Unwatch 3 | Star 83 | Fork 15

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights | Settings

DupeKeyInjector

Edit

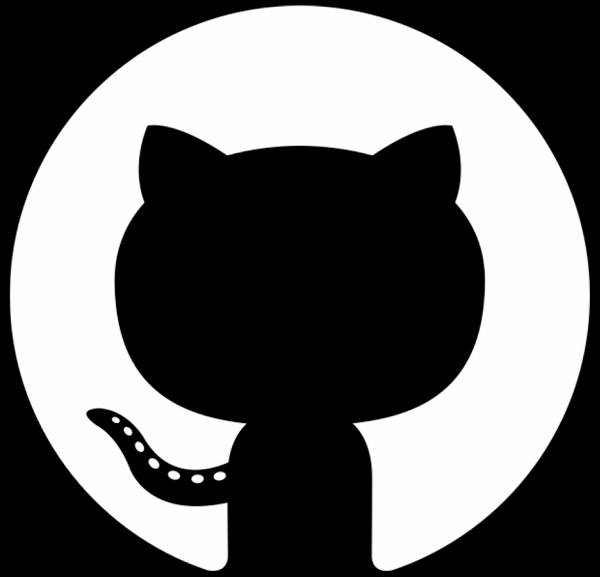
Manage topics

21 commits | 1 branch | 0 packages | 0 releases | 2 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

pwntester Merge pull request #1 from BastienFaure/patch-1 ... Latest commit 054c970 on 4 Dec 2019

resources	move screenshot to resources	5 months ago
src/main/java	Update regex to match URL encoded params	last month
.gitignore	remove target	5 months ago
LICENSE.md	Create LICENSE.md	5 months ago
README.md	Update README.md	5 months ago



@pwntester

@pwntester