Threats and Vulnerabilities in Federation Protocols and Products

Teemu Kääriäinen, CSSLP / Nixu Corporation

OWASP Helsinki Chapter Meeting #30

October 11, 2016

OWASP
The Open Web Application Security Project

Contents

- Federation Protocols: OpenID Connect and SAML 2.0
  - Basic flows, comparison between the protocols
- OAuth 2.0 and OpenID Connect Vulnerabilities and Best Practices
  - Background for OAuth 2.0 security criticism, vulnerabilities related discussion and publicly disclosed vulnerabilities, best practices, JWT, authorization bypass vulnerabilities, mobile application integration.
- SAML 2.0 Vulnerabilities and Best Practices
  - Best practices, publicly disclosed vulnerabilities
- OWASP Top Ten in Access management solutions
  - Focus on Java deserialization vulnerabilites in different commercial and open source access management products
    - Forgerock OpenAM, Gluu, CAS, PingFederate 7.3.0 Admin UI, Oracle ADF (Oracle Identity Manager)

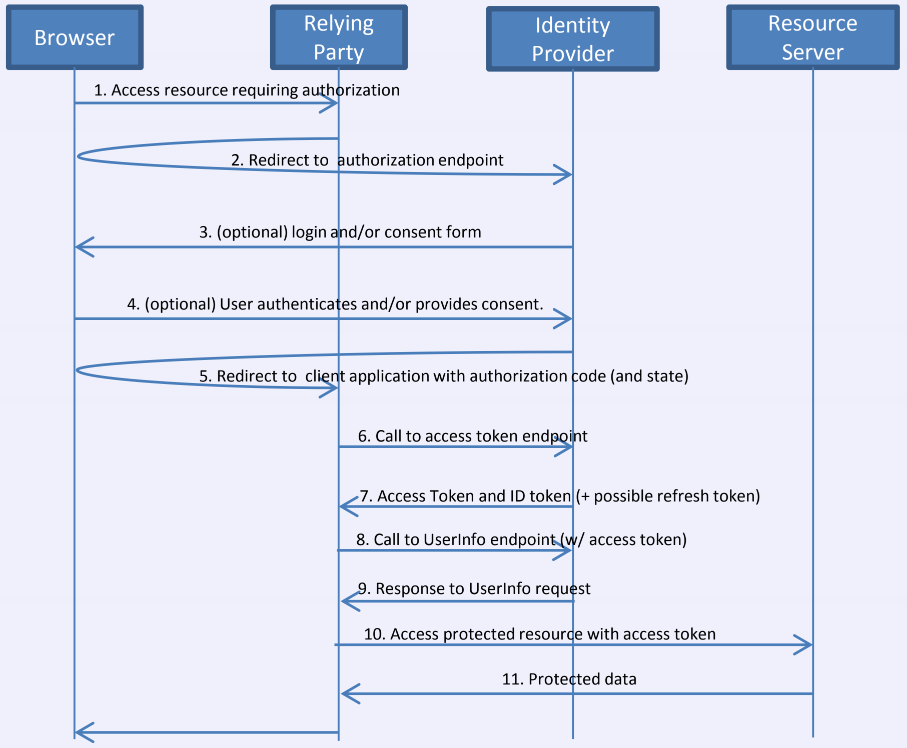
Federation Protocols: OpenID Connect and SAML 2.0

- OpenID Connect is an emerging technology built on OAuth 2.0 that enables relying parties to verify the identity of an end-user in an interoperable and REST-like manner.
- OpenID Connect is not just about authentication. It is also about authorization, delegation and API access management.
- Reasons for services to start using OpenID Connect:
  - Ease of integration.
  - Ability to integrate client applications running on different platforms: single-page app, web, backend, mobile, IoT.
  - Allowing 3rd party integrations in a secure, interoperable and scalable manner.
- OpenID Connect is proven to be secure and mature technology:
  - Solves many of the security issues that have been an issue with OAuth 2.0.
- OpenID Connect and OAuth 2.0 are used frequently in social login scenarios:
  - E.g. Google and Microsoft Account are OpenID Connect Identity Providers. Facebook is an OAuth 2.0 authorization server.

Federation Protocols: OpenID Connect and SAML 2.0, Cont..

- SAML 2.0 is an XML based protocol for exchanging authentication and authorization information between security domains.
- Most commonly used in single sign-on and federation scenarios within and between enterprise organizations:
  - Many cloud application providers (e.g. Salesforce, Google, Zendesk, Hackerone) provide mechanisms to enable single sign-on to their platforms with corporate identities using SAML 2.0.
- Solves the same problems as OpenID Connect (authentication, identity passing, federation) but the core protocol lacks the ability to provide mechanisms for API access management.

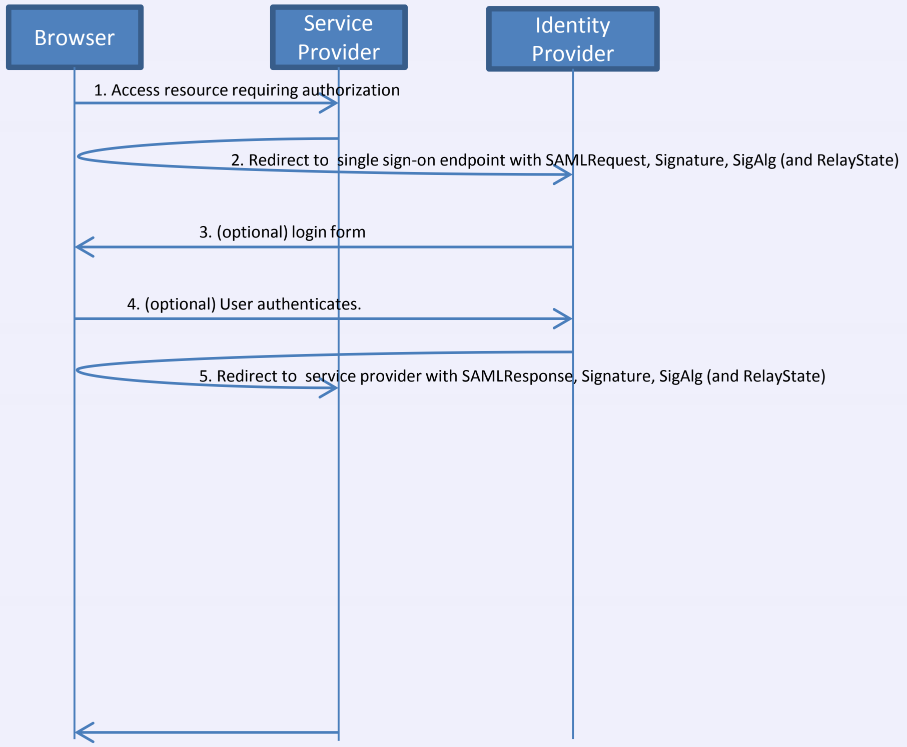
OpenID Connect High-Level Flow


Browser
Relying Party
Identity Provider
Resource Server
1. Access resource requiring authorization
2. Redirect to authorization endpoint
3. (optional) login and/or consent form
4. (optional) User authenticates and/or provides consent.
5. Redirect to client application with authorization code (and state)
6. Call to access token endpoint
7. Access Token and ID token (+ possible refresh token)
8. Call to UserInfo endpoint (w/ access token)
9. Response to UserInfo request
10. Access protected resource with access token
11. Protected data

- Diagram describes the most commonly used authorization code grant flow. Other flows include e.g. Implicit grant, resource owner password credentials grant and client credentials grant.
- Specification: http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth
- Browser-based flow
  - Relying Parties that can keep client secret secure (i.e. have a backend)
  - Suitable for desktop browser based applications and mobile applications (embedded browser)
  - For mobile applications without backend the authorization code can be output to page or written to page title in step #5.
- Security Considerations
  - Step #2 must include 'state' parameter tied to the end-user's session to prevent CSRF. Value of 'state' parameter must be validated in step #5.
  - OP must perform redirect_uri validation.
  - Access tokens must always be passed over TLS connections.
  - User consent should be asked for all provided scopes in step #3. User must have the possibility to manage authorized scopes.
  - Instead of request parameters, call to authorization endpoint can be done as JWT request object
- Preferred way of integrating mobile applications include RFC 7636: Proof Key for Code Exchange by OAuth Public Clients
  - AppAuth-iOS: http://openid.github.io/AppAuth-iOS/
  - AppAuth-Android: http://openid.github.io/AppAuth-Android/

SAML 2.0 High-Level Flow

OWASP
The Open Web Application Security Project


Browser
Service Provider
Identity Provider
1. Access resource requiring authorization
2. Redirect to single sign-on endpoint with SAMLRequest, Signature, SigAlg (and RelayState)
3. (optional) login form
4. (optional) User authenticates.
5. Redirect to service provider with SAMLResponse, Signature, SigAlg (and RelayState)

- Diagram describes the SAML 2.0 Redirect binding. Alternatively it would be possible to use the HTTP POST binding where request parameters are provided in HTTP POST payload and XML signatures are used.
- SAML 2.0 is fully related to authentication. The core specification does not take into account e.g. API access management.
- Browser-based flow
  - SAMLRequest and SAMLResponse are signed which guarantees integrity. They can also be optionally encrypted providing confidentiality.
  - Identity Providers also often provide IdP initiated flow where request to identity provider does not need to be signed.
- Security Considerations
  - SP should use RelayState to prevent CSRF indicating that authentication was really initiated by the SP.
  - Redirection URL can be overridden with the AssertionConsumerServiceuRL attribute. IdP must validate this value.
  - Proper key management and signature validation mechanisms need to be in place in SP and IdP.
  - SP should implement mechanisms to prevent SAMLResponse replay.

OpenID Connect vs. SAML 2.0

OWASP
The Open Web Application Security Project

- OpenID Connect and SAML 2.0 solve the same problems related to authentication, single sign-on and identity passing. There are however differences in the terminology.
- OpenID Connect is built on RESTful semantics and JSON whereas SAML 2.0 uses SOAP and XML.
- OpenID Connect is simpler to integrate from developer standpoint but the default specification e.g. lacks integrity in authorization endpoint.

SAML 2.0	OpenID Connect
Assertion	ID token
Attribute query	Userinfo endpoint
Authentication request	Authentication request
ForceAuthn	prompt=login
Identity Provider	OpenID Provider
IdP metadata	OpenID Provider metadata
Issuer	Issuer
Logout	End-session
NameID policy	Subject identifier type
Passive Authentication	prompt=none
Service Provider	Relying Party
SP metadata	Client metadata
Subject	Subject

Background for OAuth 2.0 Security Critisism

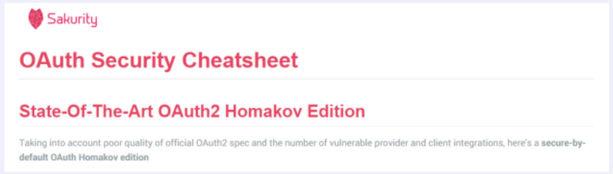
OWASP
The Open Web Application Security Project

- Fragmented authorization server implementations.
- The specification is very vague and allows basically infinite extensibility.
- No cryptographic mechanisms in place.
- Puts lot of emphasis in the secure implementation of the integrating client application.
- OpenID Connect is more explicit and contains additional security related improvements.

OAuth Security Cheatsheet

OWASP
The Open Web Application Security Project

- http://www.oauthsecurity.com/
- Maintained by Egor Homakov who has discovered large amount of OAuth 2.0 related vulnerabilities in services such as Facebook and Github
- List of best practices for implementing OAuth 2.0 authorization servers and client applications:
  - Always require using state parameter -> provides CSRF protection
  - Use static set of redirect URIs
  - Do not allow utilization of implicit grant
  - Require explicit initialization from end-user for the redirection to the authorization endpoint
  - Request only minimal set of scopes
  - Use refresh tokens only when needed
    - Facebook uses appsecret_proof
    - Google issues refresh tokens when explicitly requested



Sakurity

OAuth Security Cheatsheet

State-Of-The-Art OAuth2 Homakov Edition

Taking into account poor quality of official OAuth2 spec and the number of vulnerable provider and client integrations, here's a **secure-by-default OAuth Homakov edition**

OAuth Security Cheatsheet, Cont.

- Below list of vulnerabilities are specific to the authorization code grant flow which is the preferred authentication mechanism.
- Client account hijacking by connecting attacker's provider account.
  - Only affects the “social login” scenarios.
  - “The Most Common OAuth2 Vulnerability”
  - Mitigated by the client application always requiring the utilization of state parameter and by validating it properly.
  - State must be always generated by the server instead of allowing user-generated or predictable state parameter.
- Client account hijacking through abusing session fixation on the provider
  - Only affects the “social login” scenarios.
  - Authentication data provided by the authorization server should not be trusted.
  - Use CSRF protection for the account linking and always request explicit consent from the end-user.
- Account hijacking by leaking authorization code.
  - Authorization code may leak to external parties through the HTTP Referer header if there are third party components in the redirect URI page.
  - Use only small set of valid redirect URIs and do not allow embedding any third party components on those pages.

OAuth Security Cheatsheet, Cont.

- Leaked client credentials threat
  - Even though not much can be done with the leaked client credentials (make calls to access token endpoint), they should be stored in a secure manner in the server side.
  - If client credentials leak, they should be reset.
- Tricks to bypass redirect_uri validation
  - Only use explicit set of redirect URIs.
- Replay attack
  - It may not be guaranteed that authorization code is one-time use. Therefore it might be feasible to build mechanism to prevent replaying authorization codes.

Good news, OAuth is *almost* secure

OWASP
The Open Web Application Security Project

- http://www.theregister.co.uk/2016/01/08/good_news_oauth_is_ialmosti_secure/ (Jan 8th 2016)
- Related to the fact that some authorization servers allow the usage of HTTP 307 status code. This may leak the user's credentials from the login page. Not a threat against the integrating client applications.
- Second vulnerability is related to the case where there are multiple IdPs between which the user can choose. The RP is confused to use different IdP than originally chosen. Can be mitigated by explicitly supporting only small set of IdPs and verifying the linkage between user and IdP.

Security

Good news, OAuth is *almost* secure

Boffins turn up a couple of protocol vulns in Facebook's login standard

8 Jan 2016 at 05:03, Richard Chirgwin

67

Nasty Security Flaw In OAuth, OpenID – Covert Redirect

OWASP
The Open Web Application Security Project

- https://it.slashdot.org/story/14/05/02/2015227/nasty-security-flaw-in-oauth-openid (May 2nd 2014)
- Related to the redirect_uri validation that is already covered in the OAuth 2.0 threat model and OAuth security cheatsheet.
- E.g. Google assumes an explicit whitelist of redirect URIs (good), but Facebook allows determining only the specific domain. Facebook also allows redirecting to its internal domains (bad).
- Two such vulnerabilities (in Github and Facebook) are covered in subsequent slides.

Egor Homakov: How I hacked Github again

OWASP
The Open Web Application Security Project

- http://homakov.blogspot.fi/2014/02/how-i-hacked-github-again.html (Feb 7th 2014)
- Five different covert redirect related bugs in Github discovered by Egor Homakov:
  - Bug 1: Bypassing redirect_uri validation with /../
  - Bug 2: Lack of redirect_uri validation in OAuth 2.0 token endpoint
  - Bug 3: Leaking authorization code through Referer header
  - Bug 4: Plain-text access_token stored in cookie accessible from gist.github.com
  - Bug 5: Gist client treated as pre-approved client i.e. it receives any requested scope without user's explicit consent
- It was enough for an authenticated user to visit the attacker webpage for the attacker to take over user's Github account. Github rewarded Egor with $4000 bounty.

Nir Goldshlager: How I Hacked Facebook

OWASP
The Open Web Application Security Project

- http://www.nirgoldshlager.com/2013/02/how-i-hacked-facebook-oauth-to-get-full.html (Feb 21st 2013)
- Implicit grant was used, i.e. access_token was passed in URL fragment identifier.
- Facebook allowed redirect_uri to be pointed to some Facebook internal domains (touch.facebook.com, m.facebook.com). From there it was possible to redirect to attacker's domain and incorporate the access_token in the URL.
- The access token was issued for Facebook Messenger which meant that it was long-lived and did not require user's authorization.

OpenID Connect Security: Conclusions

OWASP
The Open Web Application Security Project

- OAuth 2.0 Threat Model (RFC 6819) and OpenID Connect Security Considerations section contain an extensive set of threats and associated mitigations to tackle the possible security issues related to OAuth 2.0 and OpenID Connect in client applications / relying parties and in authorization servers / identity providers.
- Security issues mentioned in previous slides can be tracked to threats in the threat model.
- Suggested approach would be to build security requirements from the threat model and test the implementation based on these requirements.

OpenID Connect Security: Best Practices

- Use authorization code grant even though it requires state to be stored in the backend.
  - UX can be improved with CORS or iframe. Implementation needs to reviewed carefully.
  - Use normal session management capabilities of the web application framework.
- If authorization code grant cannot be used and implicit grant needs to be used, following should be considered:
  - Allow clients to access only minimal set of API functionalities (can be controlled with scopes).
  - Make access token lifetime short. New access tokens can be created without user intervention by accessing authorization endpoint e.g. in iframe.
  - Require frequent re-authorization in case of user inactivity (checked from ID token).
  - Store access token in session storage.This guarantees that third party Javascript cannot access them.
  - Verify that tokens do not leak from fragment identifier.
  - Test thoroughly for XSS vulnerabilities.
  - Always verify that ID token is not tampered with.

OpenID Connect Security: Best Practices, Cont..

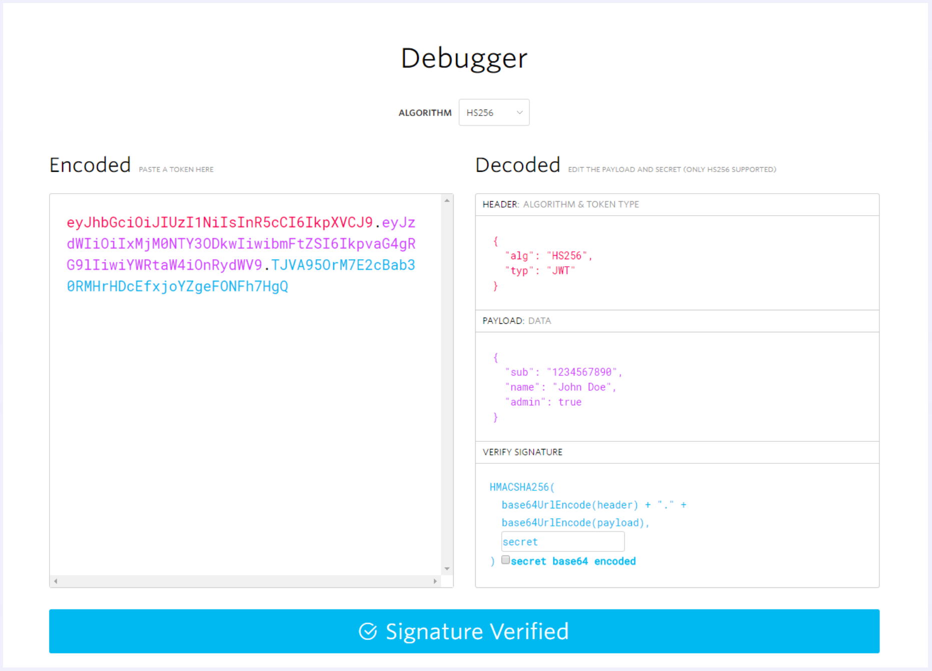
OWASP
The Open Web Application Security Project

- Require correct usage of state parameter to prevent CSRF.
- Only allow small set of valid redirect URIs. Do not incorporate external Javascript (analytics, advertisement) in landing page.
  - Always require TLS for redirect URIs.
- Do not load Javascript from e.g. external CDNs. Review carefully the need for external analytics and advertisement components.
- Use OpenID Connect end-session endpoint to end SSO session.

JSON Web Tokens

OWASP
The Open Web Application Security Project

- JSON based open standard (RFC 7519) for creating tokens to assert claims.
- Consists of three parts:
  - Header: Type of the JWT, signature algorithm and key identifier.
  - Payload: Claims of the JWT. These contain some pre-defined ones (e.g. iss – issuer of the JWT, sub – subject associated to the JWT) and custome ones defined by the issuer.
  - Signature: Signature calculated from header and payload with the algorithm defined in the header.
- Signature calculation is defined in JWS standard (RFC 7515) and encryption in JWE standard (RFC 7516). Algorithms are defined in JWA (RFC 7518) and key management in JWK (RFC 7517).
- In OpenID Connect JWTs are used e.g. to represent ID tokens which contain information about the authenticated user and about the authentication session.


Debugger
ALGORITHM HS256
Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
Signature Verified

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz
dWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR
G9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab3
0RMHrHDcEfxjoYZgeFONFh7HgQ
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

JWT Validation

OWASP
The Open Web Application Security Project

- Check that `iss` claim is known.
- `aud` claim must contain RP client_id
- Verify ID token signature. Check keys from JWKS endpoint or use your client secret (HMAC-SHA).
- Use `exp` claim to check that ID token has not expired
- `iat` claim can be used to check that ID token is recent enough
- `auth_time` claim can be used to check that authentication is recent enough

JWT Vulnerabilities

OWASP
The Open Web Application Security Project

- https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/ (March 31st 2015)
- JWT contains algorithm type 'none'. JWTs can be created with this algorithm type so that they contain no signature and RPs may erroneously accept these as valid.
- When RP is expecting asymmetric signing algorithm (RSA, ECDSA) but receives HMAC signature, it may be confused to use RSA/ECDSA public key in signature validation.

OpenID Connect / OAuth 2.0 in mobile applications

OWASP
The Open Web Application Security Project

- Until recently the way how OpenID Connect / OAuth 2.0 could be used in mobile applications was either of following:
  - Use authorization code grant: client_id and client_secret can be extracted from the application
  - Use implicit grant: no way of implementing evergreen access without re-authentications
- OpenID consortium introduced PKCE extension to enable usage of authorization code grant in mobile applications without the need to provision client secrets to the device.
- Google has open-sourced iOS and Android AppAuth SDKs to facilitate the integration of OpenID Connect to mobile applications.
  - Utilizes system browser in login flow therefore making it possible to build device side single sign-on between applications.

Microsoft OAuth 2.0 Authorization Bypass

OWASP
The Open Web Application Security Project

- https://www.synack.com/2015/10/08/how-i-hacked-hotmail/ (Oct 8th 2015)
- When allowing third party applications to be registered, they are usually allowed limited default scope and user needs to explicitly provide consent for the application to access additional API resources (scopes).
- In Microsoft Account it was possible to circumvent the additional authorization step therefore allowing any third party application to create access token with arbitrary scopes.
- This meant that if the user accessed the attacker website, it was possible for the attacker application to access full contents of user's Hotmail inbox.
- Microsoft rewarded this vulnerability with $24,000 bounty.
- Also unintended 3rd party client application creation should be prevented through OpenID Connect dynamic client registration.

SAML 2.0 Security: Best Practises

OWASP
The Open Web Application Security Project

- https://www.owasp.org/index.php/SAML_Security_Cheat_Sheet
- Protocol usage validation
  – Perform schema validation of XML
  – Securely validate digital signature
  – Avoid signature wrapping attacks
- Validate protocol processing rules for requests and responses
- Prefer short lifetimes and one time use for SAML responses

Microsoft SAML 2.0 Vulnerability

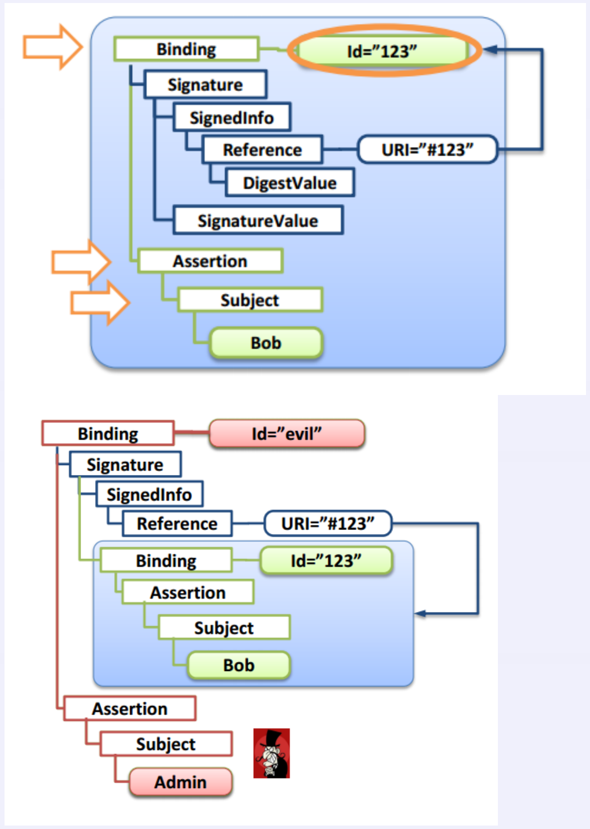
OWASP
The Open Web Application Security Project

- http://www.economyofmechanism.com/office365-authbypass.html (Apr 27th 2016)
- Microsoft allows using corporate SAML 2.0 identities to login to their Office 365 online services. Users are uniquely identified by the IDPEmail attribute in the SAML assertion.
- Microsoft SAML 2.0 SP mechanism was lacking the mapping between the Issuer and the IDPEmail therefore making it possible to use SAML IdP of e.g. someorg-attacker.com to login to accounts of e.g. someorg-victim.com.
- Microsoft fixed this issue within 7 hours of the report.

XML Signature Wrapping

OWASP
The Open Web Application Security Project

- https://www.owasp.org/images/2/28/Breaking_SAML_Be_Whoever_You_Want_to_Be_-_Juraj_Somorovsky%2BChristian_Mainka.pdf
- https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final91.pdf
- When using HTTP POST binding, XML signatures are used to guarantee the integrity of the SAML requests.
- XML Signature wrapping can be used to trick the SP or IdP to use wrong base data in the signature validation therefore making it possible to change arbitrary data in the assertion.
- SAML Extensions elements can be further used to circumvent XML schema validation countermeasures.



SAML XXE

- http://web-in-security.blogspot.fi/2014/11/detecting-and-exploiting-xxe-in-saml.html
- SAML 2.0 uses XML and is therefore vulnerable to the same XML related vulnerabilities as any other interface handling XML messages. One such attack is XXE (XML external entity reference) which may be used for e.g. information disclosure.
- More information about mitigations can be found here: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet

OWASP Top Ten in Federation Solutions

- Federation and access management solutions are usually built with same web application frameworks (e.g. JSP/Servlet, JSF, JBoss Seam, Spring) as any other web applications. This means that they are also vulnerable to the same OWASP Top Ten vulnerabilities (XSS, CSRF, SQL Injection, ..) as any other web applications.
- Access management solutions are usually complex, offering functionalities related to authentication, authorization, policy management, key management, metadata management etc. This increases the attack surface and needs to be taken into account when deploying these solutions to production environments.

Java Deserialization Based Remote Code Execution Vulnerabilities

- http://frohoff.github.io/owaspsd-deserialize-my-shorts/
  - Chris Frohoff: Deserialize My Shorts – Or How I Learned to Start Worrying and Hate Java Object Deserialization
- In January 2015 Chris Frohoff and Gabe Lawrence gave a talk in AppSec California where they introduced a set of classes that use commonly used open source libraries for attacking interfaces through which Java deserialization takes place.
- The technique was known for years, but the introduction of ysoserial provided set of practical tools how to exploit the Java deserialization vulnerabilities.
- So far different application server platforms (JBoss, Websphere, Weblogic) and middleware components (e.g. Different JMS platforms) have been proven to be vulnerable. Here the purpose is to go through vulnerabilities in commonly used commercial and open-source access management solutions in their publicly facing interfaces.
- More information about Java deserialization based vulnerabilities can be found from: https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet

Deserialize My Shorts

Or How I Learned to Start Worrying and Hate Java Object Deserialization

Chris Frohoff (@frohoff)
Gabriel Lawrence (@gebl) (in spirit)

QUALCOMM

Deserialization based RCE in Forgerock OpenAM

- OpenAM is an open-source commercially licensed Java based access management solution developed by Forgerock.
- OpenAM Security Advisory #201505: https://backstage.forgerock.com/#!/knowledge/kb/article/a68358392
- Affected versions include 9-9.5.5, 10.0.0-10.0.2, 10.1.0-Xpress, 11.0.0-11.0.3 and 12.0.0
- No exploit mechanisms have been publicly disclosed so far.
- Here we demonstrate the vulnerability which resides in the JATO web application framework which uses serialized Java objects to store page session information in the client side. Exploit uses the CommonsCollections1 gadget.

Deserialization based RCE in Gluu

OWASP
The Open Web Application Security Project

- Gluu is an open-source Java based access management solution.
- Two Java deserialization based vulnerabilities were discovered:
  - Vulnerability in Gluu oxAuth (OpenID Connect Identity Provider) application with Mojarra client side JSF ViewState. This also uses a very old CVE-2010-2087 vulnerability.
  - Vulnerability in Gluu oxAuth-RP sample RP application with JBoss Richfaces 3.3.3. This uses very old CVE-2013-2165 vulnerability.
  - In both cases CommonsCollections1 gadget was used in the exploit.
- Timeline:
  - 30/11/2015: Both vulnerabilities reported to Gluu
  - 30/11/2015: oxAuth-RP fixed by Gluu: Richfaces upgraded to 3.3.4, Commons Collections upgraded to 3.2.2.
  - 01/12/2015: First attempt to enable ViewState encryption.
  - 10/12/2015: Informed Gluu about wrong name for ClientStateSavingPassword.
  - 10/12/2015: Final fix implemented by Gluu.

gluu

Deserialization based RCE in CAS

- CAS is an open-source Java based access management solution maintained by Apereo Foundation. It is mainly used for CAS protocol based integrations, but CAS can also act as a SAML 2.0 identity provider.
- CAS uses Spring Webflow. Spring Webflow usually uses server side state saving to store the state of the login flow. From version 4.1 onwards CAS started using their own client side state storage mechanism which stored the state in client side as an encrypted Java object.
- The problematic part was that CAS did not enforce setting deployment specific encryption key. Therefore all the deployments that were evaluated were using the default encryption keys. This meant that it was possible to use these keys to create a forged state entry and execute Java deserialization based RCE.
- Timeline
  - 30/03/2016: Reported vulnerability to Apereo with CommonsCollections2 gadget.
  - 30/03/2016: Apereo verifies the vulnerability in CAS 4.1 and 4.2 and informs that fix is already in place in 4.2 master that will generate keys on the fly.
  - 31/03/2016: Apereo informs that patched versions are available for 4.1 and 4.2.
  - 31/03/2016: Confirmation that issue has been fixed.

Deserialization based RCE in PingFederate 7.3.0 Admin UI

- PingFederate is a commercial Java-based access management solution developed by Ping Identity.
- PingFederate admin UI uses Apache Tapestry 3 MVC web framework which has been in EOL for a while. Last update to Tapestry 3 has been done in 2006.
- Tapestry 3 stores form state in client side. There are different Adaptor classes for handling different data types. One such is the a class which deserializes form objects that are serialized Java objects. This can be used for the RCE using CommonsCollections1 gadget chain.
- Same vulnerability also resides in Tapestry 4 which has not been updated after 2008.
- PingFederate has patched this vulnerability in recent versions by implementing MAC validation for the serialized objects.

CVE-2016-3504: Deserialization based RCE in Oracle ADF

OWASP
The Open Web Application Security Project

- Oracle ADF is a JSF based application development framework maintained by Oracle. It is not directly related to access management, but the framework is used e.g. in Oracle Identity Manager.
- Oracle ADF uses Apache Trinidad as the underlying JSF framework. Same vulnerability has been reported in Trinidad recently (CVE-2016-5019): https://issues.apache.org/jira/browse/TRINIDAD-2542
- Also related to the client side state saving. Utilizes CommonsCollections1 gadget chain.
- Timeline:
  - 22/02/2016: Reported vulnerability to secalert_us@oracle.com
  - 23/02/2016: Security bug number assigned for the vulnerability
  - 24/02/2016: Status update: under investigation / being fixed in main codeline
  - 24/03/2016: Status update: Issue fixed in main codeline, scheduled for a future CPU
  - …
  - 19/07/2016: Fix published in July 2016 CPU: http://www.oracle.com/technetwork/security-advisory/cpujul2016-2881720.html

Deserialization based RCEs: Commonalities

OWASP
The Open Web Application Security Project

- All vulnerabilities were found in different web application frameworks used by the products (Mojarra/JSF, Richfaces, JATO, Spring Webflow, Tapestry3, Trinidad).
- All vulnerabilites were related storing state information in client side. Instead of using serialized Java objects in client side, some alternative mechanims (e.g. JSON) should be considered.
- It is not enough to only upgrade the vulnerable libraries (e.g. Commons Collections), but the underlying issue of deserializing data from untrusted sources should be fixed.

OWASP
The Open Web Application Security Project

Questions?