Reading Security Protocol Specifications is Difficult and Error Prone

COINS summer school 2021

June 18, 2021

Dieter Gollmann

Security in Distributed Applications

TUHH

Hamburg University of Technology

About Myself

- PhD on a topic in cryptography (Linz, Austria, 1984)
- Postdoc at University of Karlsruhe, Germany (1986–1990)
- Information Security Group, Royal Holloway, University of London (1984–1985, 1990–1997)
    - Course director of the MSc in Information Security
- Microsoft Research Cambridge, 1998–2003
- Chair for Security in Distributed Applications, Hamburg University of Technology, 9/2003 - 3/2021 (retired!)
- Visiting positions at Royal Holloway, QUT, Technical University of Denmark, Tsinghua University, Nanyang Technological University

Going Walkabout

- Talk about the difficult things you shouldn't do if you have to complete your PhD research within three years
- OAuth 2.0 – return on experience
- Protocols for the German eHealth card
- Observation on sources (in the web)

Security Protocols

Needham, Roger M., and Michael D. Schroeder.
"Using encryption for authentication in large networks of computers", Communications of the ACM 21 (12), 1978

1. $A \rightarrow B : \{I_A, A\}^{PKB}$
2. $B \rightarrow A : \{I_A, I_B\}^{PKA}$
3. $A \rightarrow B : \{I_B\}^{PKB}$

"We made it socially acceptable to write academic papers on three-line protocols"

[Roger Needham]

Designing Security Protocols

Designing security protocols is difficult and error prone

[Popular line in many papers on formal protocol analysis]

Proving Protocol Security

- Research on formal protocol analysis started in the 1980s
- Success story: we have a range of analysis tools
  - AVISPA, ProVerif, Scyther, Tamarin, ...
- Help to avoid embarrassing mistakes [Tom Berson, former president, IACR]
- ‘Serious’ security protocol proposals are expected to come with formal proofs today
- How can it be that protocols succumb to attacks in practice, although they had been formally verified?
- Has all this great research been in vain?

From Specification to Deployment

- Starting point: protocol specification, typically a public standard
- Programmers develop an implementation that should comply with the standard
  - Might not work immediately; it took years to get compatible implementations of software handling X.509 certificates
  - For this presentation, let's assume that such issues have been resolved
- Programmers make design decisions on matters required but not detailed in the specification
  - For example, counter or pseudo-random number generator for creating nonces, parser for comparing URLs, . . .
- Sysadmins configure the protocol when it is deployed
  - Choice of ciphersuites, security policy rules, . . .

OAuth 2.0

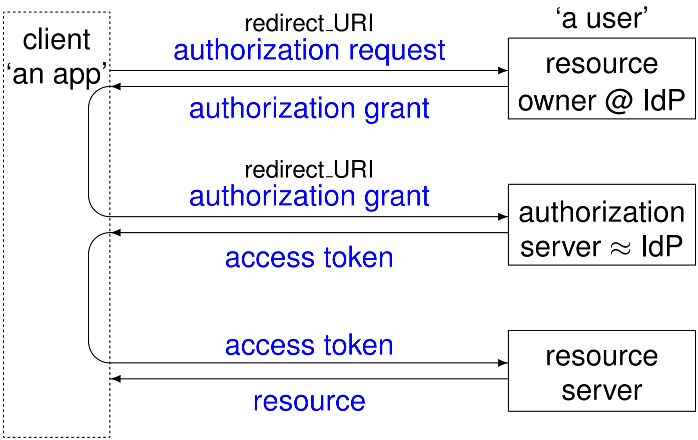
Use Case

- A user has an account with an identity provider (IdP)
  - Facebook is an illustrative example for such an IdP
- The user stores personal data at a resource server
  - Facebook is an illustrative example for such a service
  - The user is the resource owner
  - Resource server and IdP may be the same party
- An app running on the user's device wants to access certain user data
- The user has to authorize this request
  - User decides whether to 'delegate' access rights to the app
- The resource server checks that a request is authorized before giving access to the data

OAuth 2.0

- OAuth 2.0 addresses this use case
- Specified in RFC 6749, October 2012
- OAuth 2.0 is an authorization protocol, not an authentication protocol
  - This is an important paradigm change
- OpenID Connect adds user authentication to OAuth 2.0
- Tenuous relation to OAuth 1.0, OpenID 1.0, OpenID 2.0
  - Not earlier versions in the usual meaning of this term
  - Marketing wanting to keep established ‘trademarks’?
  - These protocols are now deprecated in favour of OAuth 2.0
- https://developers.google.com/identity/protocols/oauth2/openid-connect#update-to-plus

OAuth 2.0 – Message Flow



OAuth 2.0

- This diagram is often the first contact with OAuth 2.0
  - Top four images in my Google search on ‘OAuth 2.0’
- High level specification from RFC 6749
- Timeline of messages runs from top to bottom
- The authentication server is the IdP mentioned earlier
- Further security protocols that are expected to be in place stay under the radar
- Trust relationships that are expected to be in place stay under the radar

Protocol Endpoints [RFC 6749, Sec. 3]

- The authorization process utilizes two authorization server endpoints (HTTP resources):
  - Authorization endpoint – used by the client to obtain authorization from the resource owner via user-agent redirection
  - Token endpoint – used by the client to exchange an authorization grant for an access token, typically with client authentication
- One client endpoint:
  - Redirection endpoint – used by the authorization server to return responses containing authorization credentials to the client via the resource owner user-agent.
- Extension grant types MAY define additional endpoints as needed

OAuth 2.0 – Client Registration [RFC 6749]

- Before initiating the protocol, the client registers with the authorization server.
- The means through which the client registers with the authorization server are beyond the scope of this specification but typically involve end-user interaction with an HTML registration form.
- When registering a client, the client developer SHALL:
  - Specify the client type (confidential or public)
  - Provide its client redirection URIs
  - Include any other information required by the authz server
- Confidential clients can be authenticated by the authorization server; the RFC does not specify how, but ...
- The authorization server MUST require the use of TLS when sending requests using password authentication

Authorization Grant

- Authorization Grant: expresses authorization to access a resource; granted by resource owner; used by client to obtain access token
- Access token: credential granting access to resource
- Authorization code: authorization grant obtained from authorization server, authenticates client and resource owner
  - Three more grant types, but they are not relevant for this presentation
- What is a ‘credential’?

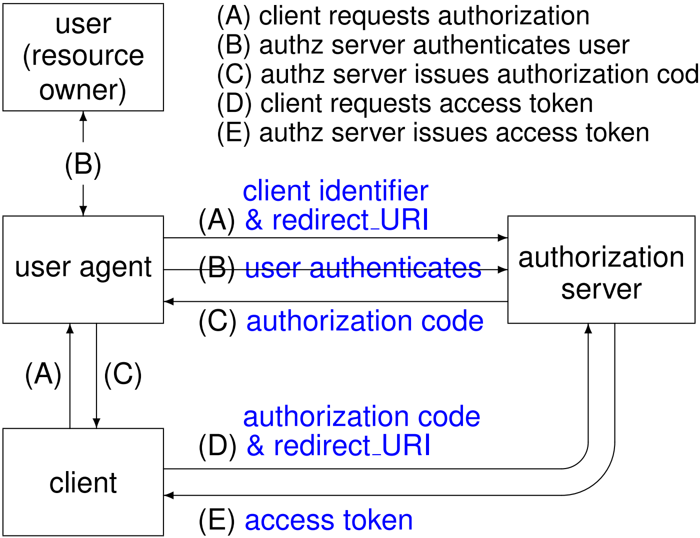
Requesting an Authorization Code

- User navigates to client's web page in browser (user agent)
- User clicks on a 'Connect' / 'Sign in with' button shown on that web page; triggers a GET request to client
- Client now redirects the user agent to the authorization server / IdP using the following query parameters:
  - response_type: code
  - client_id: id issued to the client
  - redirect_uri (optional): URI where the authorization server should redirect its response to
  - scope (optional): scope to be requested
  - state (recommended): opaque value for maintaining state between request and response
- Authorization code (and access token) returned to redirect_uri

Requesting an Authorization Code ctd.

- User must be authenticated at server and then may authorize the client to access the requested resources
- Authorization server then redirects the user agent to the redirect_uri using the following query parameters:
  - code: the authorization code
  - state: value passed in the previous request (*recommended*)
- Client can use the authorization code to request an access token (with appropriate client authentication), passes the request parameters:
  - grant_type: authorization_code
  - code: authorization code received earlier
  - redirect_uri: redirect_uri passed in the first request

OAuth 2.0 – Requesting an Authorization Code



(A) client requests authorization
(B) authz server authenticates user
(C) authz server issues authorization code
(D) client requests access token
(E) authz server issues access token

OAuth 2.0 – Messages

- Authorization Request

```
GET /authorize?response type=code&
client id=s6BhdRkqt3&state=xyz&
redirect uri=https://client.example.com HTTP/1.1
Host: server.example.com
```

- Authorization Response

```
HTTP/1.1 302 Found Location:
https://client.example.com/cb?
code=SplxlOBeZQQYbYS6WxSbIA&state=xyz
```

- The “state” parameter links request and response

On Cross-Site Request Forgery (CSRF)

- The client MUST implement CSRF protection for its redirection URI
- The client SHOULD utilize the "state" request parameter to deliver this value to the authorization server when making an authorization request [RFC 6749, Sec. 10.12]
- RFC 6749 gives a security reason for using "state"
- RFC 6749 does not insist on the use of "state", but some other CSRF defence has to be present
  - Spec of the CSRF defence is out of scope for RFC 6749
  - Limiting the scope of a standard is a reasonable design decision
- OAuth 2.0 security depends on aspects beyond RFC 6749

OAuth 2.0 for Native Apps [RFC 8252]

- Many mobile and desktop computing platforms support inter-app communication via URIs by allowing apps to register private-use URI schemes (sometimes referred to as “custom URL schemes”) like “`com.example.app`”.
- When the browser or another app attempts to load a URI with a private-use URI scheme, the app that registered it is launched to handle the request.
- To perform an OAuth 2.0 authorization request with a private-use URI scheme redirect, the native app launches the browser with a standard authorization request, but one where the redirection URI utilizes a private-use URI scheme it registered with the operating system

Redirect URLs for Native Apps

- In order to support a wide range of types of native apps, your server will need to support registering three types of redirect URLs, each to support a slightly different use case
- Some platforms, . . . , allow apps to register a custom URL scheme which will launch the app whenever a URL with that scheme is opened in a browser or another app
- Supporting redirect URLs with a custom URL scheme allows clients to launch an external browser to complete the authorization flow, and then be redirected back to the application after the authorization is complete
- App devs should choose a URL scheme that is globally unique, and one which they can assert control over
- `https://www.oauth.com/oauth2-servers/oauth-native-apps/redirect-urls-for-native-apps/`

Reverse Domain Name Patterns

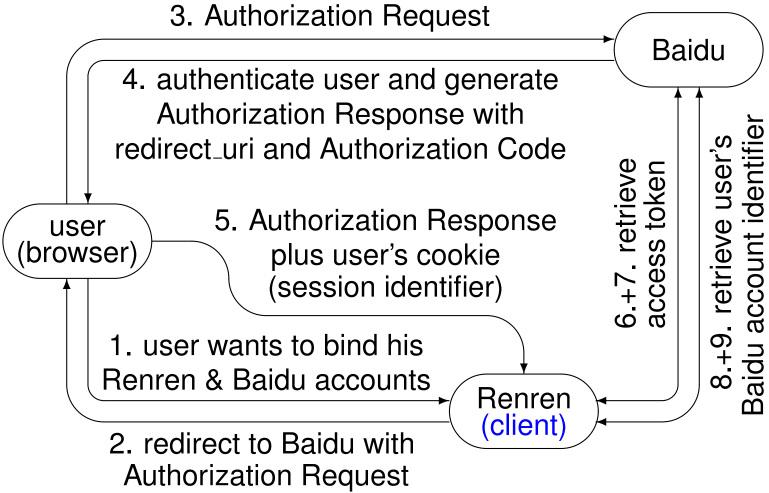
- For example, if an app has a corresponding website called `photoprintr.example.org`, the reverse domain name that can be used as their URL scheme would be `org.example.photoprintr`
- The redirect URL that the developer would register would then begin with `org.example.photoprintr://`
- By enforcing this, you can help encourage developers to choose explicit URL schemes that won't conflict with other installed applications
- https://www.oauth.com/oauth2-servers/oauth-native-apps/redirect-urls-for-native-apps/

OAuth 2.0 – Summary

- OAuth is an authorization system
  - Authorization server acts an intermediary between clients and resource owners
  - Resource owners grant access, one-time authorization
  - Clients that have been granted access get access token (longer lifetime) from authorization server
- Policy Information Points: resource owner, client (sets redirect_uri)
- Policy Decision Points: resource owner, authorization server
- How secure is OAuth in practice?

OAuth 2.0 – Return on Experience

OAuth – Renren to Baidu Binding



Renren-Baidu Account Binding

- User logged in at Renren (RP) wants to bind Renren account to Baidu (IdP) account
- Renren sends an Auth Request via User Agent (browser) to Baidu without including state in Auth Request
  - No way of binding request to subsequent Auth Response
- Baidu authenticates user and returns an Auth Response containing a code via the UA to Renren; UA adds cookies containing user's session ID
- Renren uses the code to get access token from Baidu; then retrieves the Baidu account ID using the access token
- Finally Renren binds user's Renren account ID to user's Baidu account ID

Renren-Baidu Binding Attack

1. Attacker has an account with Baidu and runs OAuth to get an Auth Response for the attacker's ID
2. Attacker posts response as a link to a forum
3. A user with an active Renren session clicks on that link
4. User Agent (browser) will follow redirect_uri and redirect response to Renren
5. Renren retrieves attacker's Baidu ID and binds it to user's Renren ID (steps 6-9 of the binding protocol)

Wanpeng Li and Chris J Mitchell. Security issues in OAuth 2.0 SSO implementations, ISC 2014

Renren-China Mobile Binding Attack

- In this implementation, Auth Request and Auth Response did contain a state value
- But the same value (always ‘9’) was used for multiple requests and responses
- An attack similar to the Renren-Baidu attack works, binding attacker’s China Mobile account to victim’s Renren account

- Lesson: the implementation matters
- Lesson: nonces are a very useful security primitive

Wanpeng Li and Chris J Mitchell. Security issues in OAuth 2.0 SSO implementations, ISC 2014

Generic Ctrip Binding Attack

- Ctrip is a travel agency with a focus on China
- Renren among the supported OAuth 2.0-based IdPs
- In the Renren-Ctrip binding process (Renren acting as IdP not RP as before), Auth Request and Auth Response did contain a state value, but previous attack did not work
- Initial HTTP request contained a Ctrip generated user ID
- Replacing the UID in attacker's request with victim's UID caused Ctrip to bind attacker's IdP account to victim's Ctrip account (worked not only for Renren)
- Cause: logic flaws in Ctrip's implementation

Wanpeng Li and Chris J Mitchell. Security issues in OAuth 2.0 SSO implementations, ISC 2014

CSRF Vulnerabilities – 2015

- "Mobile application developers have struggled to develop secure implementations of OAuth 2.0 because their use in non-web applications conflicts with assumptions made in the standard"
- Alexa Top 10,000 domains: "25% of websites using OAuth appear vulnerable to CSRF attacks"
- Major IdPs have published OAuth code samples that omit the state parameter

E. Shernan et al., More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations, DIMVA 2015

CSRF Vulnerabilities – Remedies

- Specification explicitly asks for CSRF protection (slide 20)
- Developers need guidance to get it right
- Authorization Servers / Identity Providers have leverage
- Advice on good implementations:
  - Identity providers should provide correct and complete developer tools for implementing OAuth
- Do not accept bad implementations
  - Identity providers should reject OAuth requests that do not contain all necessary authenticating tokens

Open Redirectors [RFC 6819]

- An open redirector is an endpoint using a parameter to automatically redirect a user agent to the location specified by the parameter value without any validation
- If the client may register only part of the redirect URI, an attacker can use an open redirector at the client to construct a redirect URI that will pass authorization server validation but will send the authorization code or access token to an endpoint under the control of the attacker
- Impact: Attacker could gain access to authorization codes or access tokens
- Countermeasure: Clients must register full redirect URI

RFC 6819. OAuth 2.0 Threat Model and Security Considerations, January 2013

Open Redirectors in the Media, 2014

- "Serious security flaw in OAuth, OpenID discovered"
  - http://www.cnet.com/news/serious-security-flaw-in-oauth-and-openid-discovered/
- "In terms of severity, Covert Redirect ranks fairly low"
- "It turns out that Covert Redirect has been known about for some time"
- "The cynic in me suggests that we're going to see a lot more flashy "new" vulnerabilities discovered by upstart security firms and researchers aiming to attract attention to themselves and their research"
  - https://www.mcafee.com/blogs/consumer/consumer-threat-reports/what-is-covert-redirect/

Attacks via Redirect_URI

- When an attacker manages to set the redirect_uri sent with the authorization grant, the access token will be sent to the URI specified by the attacker
- Defences: authorization server checks redirect_uri against
  - a redirect_uri white list created when the client registered at the authorization server
  - the redirect_uri used in the first request (assumes the authorization server handles both requests; requires the authorization server to keep state)

Lassie Come Home, 2013

- www.thecloudcompany.biz offers the possibility to register your own client
- One of the clients of www.thecloudcompany.biz is a department of www.thecloudcompany.biz itself
- This client, named example here, runs under the domain of www.example.thecloudcompany.biz
- Assume this client registers the redirect_uri *.thecloudcompany.biz
- The bad guy registers a client with client id 'Bad' and the redirect_uri www.bad.com at www.thecloudcompany.biz

http://blog.intothesymmetry.com/2013/05/oauth-2-attacks-introducing-devil-wears.html

Lassie Come Home – The Attack, 2013

- Bad guy sends this link to a resource owner: `https://www.thecloudcompany.biz/oauth/authorize?client_id=example&response_type=token&redirect_uri=https://www.thecloudcompany.biz%2Foauth/authorize?%2Fauthorize%3Fclient_id=Bad%26response_type=token%26redirect_uri=http://www.bad.com`
- When the resource owner clicks on this link, the access token is sent to `www.bad.com` because the redirect_uri `https://www.thecloudcompany.biz%2Foauth%2Fauthorize%3Fclient_id=Bad%26response_type=token%26redirect_uri=http://www.bad.com` matches the redirect_uri `*.thecloudcompany.biz` of client 'example'

http://blog.intothesymmetry.com/2013/05/oauth-2-attacks-introducing-devil-wears.html

Another Redirect_URI Bug, 2014

- The redirect_uri in `https://graph.facebook.com/oauth/authorize` is not validated correctly
- One can bypass the redirect_uri validation with `/.\.\../`, which might result in stealing the authorization code of a Facebook registered OAuth client
- For example, in `https://parse.com/account` there is the option to link an account with Facebook via `https://www.facebook.com/dialog/oauth?response_type=code&client_id=506576959379594&redirect_uri=https%3A%2F%2Fparse.com%2Fauth%2Ffacebook%2Fcallback&state=420c2f177072bc328309aab640fa0e9141b0f7de2c1f7d81&scope=email`

http://blog.intothesymmetry.com/2014/04/oauth-2-how-i-have-hacked-facebook.html

Exploit, 2014

- Exploit uses the request
  `https://www.facebook.com/dialog/oauth?response_type =code&client_id=506576959379594&redirect_uri= https%3A%2F%2Fparse.com%2Fauth%2Ffacebook %2Fcallback%2F.\.\../.\.\../asanso& state=420c2f177072bc328309aab640fa0e91 41b0f7de2c1f7d81&scope=email`
- If this redirect_uri is wrongly accepted, code and state are passed to
  `https://parse.com/auth/asanso?code=CODE#_=_`
- From the blog: “ ”

http://blog.intothesymmetry.com/2014/04/oauth-2-how-i-have-hacked-facebook.html

From the Blog

Q. Why is the browser changing "`https://gist.github.com/auth/facebook/callback/.\.\../.\.\../.\.\../asanso/a2f05bb7e38ba6af88f8`" to `https://gist.github.com/asanso/a2f05bb7e38ba6af88f8`

A. Is this a question :) ?

C. I can only guess that the answer has something to do with the way illegal backslash characters in the URI are handled

https://hackerone.com/reports/405100

Path Separators

- Most browsers treat both / and \ as path separators
- When a URL is entered in the address bar, most browsers automatically convert \ to /
  - Desired behaviour according to the URL standard
- However, URL validator and browser may disagree

The Evil Slash Trick

- Example: `https://attacker.com\@benign.com`
- Parser does not treat \ as path separator but browser does
  - Parser extracts domain `benign.com`
  - Browser converts \ to / and gets domain `attacker.com` from `https://attacker.com/@benign.com`,
- Parser treats \ as path separator, but the browser does not
  - Parser converts \ to / and gets the domain `attacker.com`
  - Browser extracts domain `benign.com`
- Bug in Safari, 2019:
  - When handling a redirection, Safari allows \ in user-info and does not treat it as a path separator
  - When parser treats \ as path separator and retains it in the output, Safari will be redirected to `attacker.com`

Xianbo Wang et al., Make Redirection Evil Again: URL Parser Issues in OAuth

https://www.blackhat.com/asia-19/briefings/schedule/#make-redirection-evil-again---url-parser-issues-in-oauth-13704

Leaking Tokens with Redirect_URI, 2016

- As the login with Facebook does not have a dedicated directory like `gratipay.com/facebook/callback` it is possible to still steal access tokens:
`https://www.facebook.com/dialog/oauth?response_type=code&client_id=1441249023904047&redirect_uri=https://gratipay.com/~attacka/&scope=public_profile%2Cemail%2Cuser_friends&state=mjemgKNb0s24lbEqBcyVqDEVNoYDYs`
- The token will be sent to the attacker's profile `/~attacka`, which in turn may point to `example.com`
- If a user clicks on that link the referrer header will send the tokens
- Fix: add a redirect_uri like `https://www.gratipay.com/facebook/callback`

https://hackerone.com/reports/140432

Stealing Users OAUTH Tokens via Redirect_uri, 2018

- On `https://accounts.bistudio.com` the redirect_uri given is not properly validated; it is hence possible to bypass the filter and to exfiltrate users' OAUTH tokens
- On clicking on Login on `https://xbox.dayz.com` an OAUTH request is triggered to `accounts.bistudio.com`; the endpoint checks if the redirect_uri starts with `https://xbox.dayz.com` but does not check the ending bits; it is thus possible to inject anything after that
- For example, `https://xbox.dayz.comtest.com` will pass the server's filter and a redirect with the code and state values is performed to this URL

`https://hackerone.com/reports/405100`

Redirect_uri Bypass Using IDN Homograph Attack, 2020

- SEMrush OAuth implementation fails to properly validate the value of redirect_uri parameter which was bypassed using IDN homograph attack which results in leaking the user's access token to an attacker-controlled domain name
- IDN homograph attack exploits the fact that different characters look alike, e.g. the Cyrillic letter e and Latin e
- Attacker registers a homograph for `semrush.com` as its domain
- Attack succeeds when a liberal matching algorithm checking the redirect_uri treats homographs as equivalent
- A liberal matching algorithm may accept `sémrush.com`, `sêmrush.com`, `sèmrûsh.com`, `šemrush.com` for `semrush.com`

https://hackerone.com/reports/861940

Proof of Concept

- Authenticate to your account then browse to `https://oauth.semrush.com/oauth2/authorize?response_type=code&scope=user.info,projects.info,siteaudit.info&client_id=seoquake&redirect_uri=https://oauth.šemrush.com/oauth2/success`
- Once you approve the SEMrush application, your OAuth code will be sent to `oauth.šemrush.com`, i.e. to `oauth.xn--emrush-9jb.com` since the browser will translate it to the punycode version
    - Punycode used in DNS to represent IDNs with characters (A-Z, 0-9)
- Attacker just needs to register the domain name `xn--emrush-9jb.com`

https://hackerone.com/reports/861940

Spring Security – Attacking OAuth, 2020

- Redirection attacks rely on the fact that the OAuth standard does not fully describe the extent to which the redirect_uri must be specified; this is by design
- Thus some implementations of OAuth allow for a partial redirect_uri
- Let an application developer registers the redirect_uri `*.cloudapp.net` with the authorization server
- This would be valid for `app.cloudapp.net` but also for `evil.cloudapp.net`
- `cloudapp.net` is a part of Microsoft's Windows Azure platform and allows any developer to host a subdomain under it to test an application

https://www.baeldung.com/spring-security-oauth-attack-redirect

Spring Security – Attacking OAuth, 2020

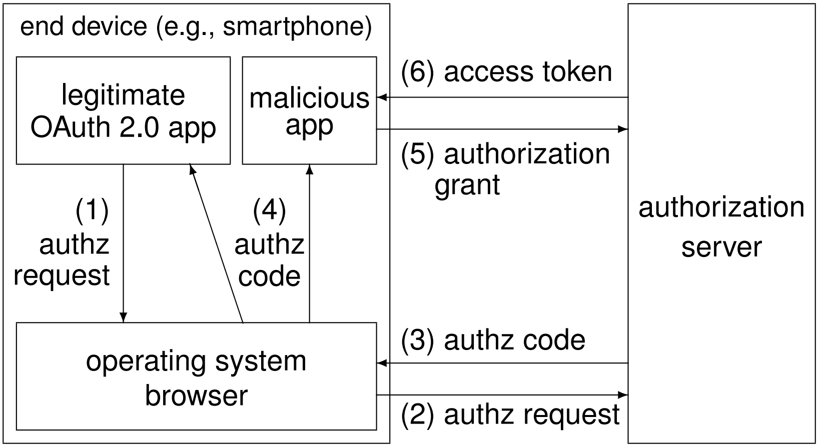
- The attacker creates the link
  `GET /authorize?response_type=code&client_id={apps-client-id}&state={state}&redirect_uri=https%3A%2F%2Fevil.cloudapp.net%2Fcb HTTP/1.1`
- When a user clicks on this link, the authorization server receives a URL with the app's Client ID and a redirect_uri pointing back to the attacker's endpoint
- The server will accept the redirect_uri as valid, authenticate the user and ask for consent w.r.t. the client app
- It will finally redirect back into the `evil.cloudapp.net` subdomain, passing the authorization code to the attacker
- The attacker can use this authorization code to receive an access token for the resource owner's protected resources

https://www.baeldung.com/spring-security-oauth-attack-redirect

Public Clients & Redirection Endpoints

- Clients incapable of maintaining the confidentiality of their credentials (e.g., clients executing on the device used by the resource owner, such as an installed native application or a web browser-based application), incapable of secure client authentication via any other means. [RFC 6749]
- Step 1: native application running on the end device, such as a smartphone, issues an OAuth 2.0 Authorization Request via the browser/operating system
  - The redirect_uri registered by the native app typically uses a custom URI scheme
  - Request happens through a secure API that cannot be intercepted, though it may potentially be observed in advanced attack scenarios

Authorization Code Interception Attack [RFC 7636]



Authorization Code Interception Attack

- Step 2: the request is forwarded to the OAuth 2.0 authorization server
  - OAuth 2.0 requires the use of TLS, so this communication is protected by TLS and cannot be intercepted
- Step 3: the authorization server returns the authorization code to the resource owner's device
- Step 4: the authorization code is returned to the native app via the redirect_uri that was provided in step (1)
- A malicious app may register itself as a handler for the custom scheme registered by the legitimate app
- The malicious app may then receive the authorization code in step (4); the attacker can then request and obtain an access token

Lessons

Problem Spots

- Understanding and applying proper CSRF protection
- Understanding the use of state parameter in a challenge-response authentication pattern
- Setting policies: wildcard in redirect_uri, open redirectors, custom URI schemes?
- Definition of ‘match’ when comparing the redirect_uri in a request with a registered redirect_uri
- Parsing strings

Remedies

- Problem spots 1 & 2 relate to general security knowledge
- It is not the task of a standard to give tutorials on security basics
- Good code samples are a service to the community
  - Identity providers can set good examples
  - Copy-and-paste will raise security levels, even when programmers are security-unaware
- Ultimately, it would be better to have some security expertise when implementing security features

Custom URIs

- Custom URIs must be locally unique on the end device
- If an app developer wants to use the same URI on all end devices, custom URIs must be globally unique
- Achieved by the reverse domain name pattern, as long as all app developers are honest
- Malicious app developers may cheat and pick a custom URI already used by someone else
- Defences?
  - Operating system on end device could check for duplicates
  - User might be asked to decide in case of a conflict
- Can this work in practice?

Remedies

- Problem spots 3 & 4 relate to access control
- When registering a redirect_uri, the client sets a policy to be enforced by the authorization server
- Wildcards in the redirect_uri were included as a feature
  - Gives clients some flexibility in their choice of endpoints
  - Wildcards in TLS certificates, e.g. for `*.google.com`
  - Use of wildcards is discouraged today
- IdPs may enforce policies on redirect_uri's at registration time or warn clients about dangerous practices

"A knife sharp enough to cut meat is sharp enough to cut your finger"

[Fred Schneider quoting David Parnas]

On URL Matching [RFC 6749]

- When a redirection URI is included in an authorization request, the authorization server MUST compare and match the value received against at least one of the registered redirection URIs
- If the client registration included the full redirection URI, the authorization server MUST compare the two URIs using simple string comparison

- Design decisions on “match”:
  - Matching prefix or identical strings?
  - How far to take Postel’s Law ”be liberal in what you accept”?
- OAuth 2.0 matching has become less liberal over time

The Full Picture

- Problem spot 5 relates to software security
  - Writing parsers can be “difficult and error-prone”
  - Momot et al., The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them, SecDev 2016
- Vulnerabilities can be product specific and technical knowledge may have a short time-to-live
- Challenge for developers: up-to-date view on threat landscape
- “In order to have a safe implementation it is important to understand what is OAuth about and to be involved in the “OAuthsphere” (OAuth mailing list, blogs, etc)” `http://blog.intothesymmetry.com/2013/05/oauth-2-attacks-introducing-devil-wears.html`
- Challenge for researchers: recognizing threat patterns

eCard Protocols

Background – eHealth

- German eCard strategy for the health sector in the 2000s
- Based on authentication, qualified electronic signatures, and the use of smart card based tokens
- Privacy issues are addressed in eCard applications but remain a key concern for citizens
- Politically charged atmosphere where certificational weaknesses have to be taken very seriously
- We had analyzed the security protocols within a project assessing the security of the German health card

Jan Meier, DG. Caught in the Maze of Security Standards. ESORICS 2010

Background – Architecture

- A security architecture relates the overall security goals of an application to the specific security services provided by the individual components and security protocols
- In eCard applications, authentication is one goal; it can be reached in multiple ways
- The architecture typically references standards to specify the exact method; these standards define protocols and cryptographic parameters
- Having multiple standards frequently causes cross-dependencies, gaps, or conflicts between requirements

Intentional Underspecifications

- A standard should not constrain possible implementations when a desired behaviour can be achieved in multiple ways
- Implementation details are therefore left open
- Protocol analysis at the level of the specification in the standard the may then flag a failure, although simple defences are available at the implementation level
- Protocol analysis therefore needs more than the protocol specification and also information on the implementation
  - Dual of the situation where an abstract protocol has been verified to be secure, but an implementation introduces vulnerabilities
  - Here, the abstract protocol would be insecure and the implementation plugs the gaps

Smart Cards

- Smart cards do not have a user interface and cannot initiate an action; they can only react
- A card reader has to send commands to the smart card and wait for a response
- Protocols can be built from such request/response pairs
- Commands can manipulate the internal state of a card
- Based on the internal state, the card may reject commands sent by the reader
- We will examine authentication between smart card and reader

Security Protocols

- The following five standards or standard related documents are relevant for eCard projects
  - Each document resides on its own abstraction layer and addresses different issues
- ISO/IEC 9798 series
- BSI TR-03116 Technische Richtlinie für eCard-Projekte der Bundesregierung
- ISO/IEC 7816 series
- CWA14890-1
- Common Criteria

ISO/IEC 9798 Series

- Describes entity authentication protocols for symmetric key cryptography and asymmetric key cryptography
- Application and technology independent, protocols are described on an abstract level.
- Gives detailed descriptions of the actions communication partners have to perform
- Does not specify message formats, cryptographic algorithms, and key lengths
- Hence, these standards do not define direct blueprints for implementation.
- Referenced in a standard for smart cards as secure signature creation devices and in the eCard guideline documentation

BSI TR-03116

- Recommendations on the strength of cryptographic algorithms and key lengths published by the German Federal Office for Information Security
- E.g., life-spans for encryption mechanisms (April 2009!)
  - Two key triple DES (2KTDES) prohibited in eCard projects after 2009
  - Three key triple DES could be used until the end of 2013
- No migration strategies or ways to adapt existing protocol to eCard requirements
- Does not assist application designers in adapting protocols from ISO/IEC 9798 to smart cards

ISO/IEC 7816 Series

- Standardizes interactions between smart cards and their environment
- Includes electrical interfaces, position of connectors, dimensions and commands
- ISO/IEC 7816-4 specifies byte sequences to invoke commands, transmission of command data (parameters), and the status flags a command could possibly return
- Command processing is left unspecified
- Tied to smart card technology but independent of the applications realized with the help of smart cards

ISO/IEC 7816-4

- Commands relevant for authentication protocols:
- `Manage Security Environment (MSE)`: sets information about the cryptographic algorithms and keys for later use
- `Get Challenge (GetChall)`: requests a challenge from the smart card; card stores the last challenge requested
- `Read Binary (ReadB)`: requests the content of a file given in the command data
- `Internal Authenticate`: generates an authentication token from the command data
- `External Authenticate (ExAuth)`: transfers an authentication token from an external party
- `Mutual Authenticate (MutAuth)` combines `Internal Authenticate` and `External Authenticate`

CWA14890-1

- Primarily an interoperability standard maintained by the European Committee for Standardization
- Builds on ISO/IEC 7816-4 as it uses smart card commands when specifying protocols
- Not intended for a specific application, but refers specifically to secure signature creation devices
- Does not include a security argument, nor does it give a reference to such a security argument
- The protocol specification for mutual authentication defines 2KTDES as the encryption algorithm
  - But 2KTDES is prohibited in BSI TR-03116
- Unlike BSI TR-03116, no statements on the security or lifespan of the protocols

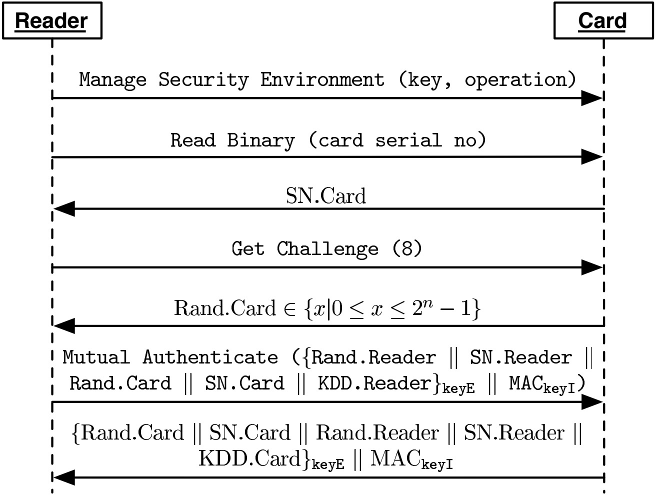
Common Criteria

- Common Criteria protection profiles are relevant for eCard applications
- Indicate which tests eCard components have to pass in order to get certified
- However, designers cannot extract requirements that, say, smart card commands have to fulfil
- They must trust smart card producers that cards are suitable for the intended task

Authentication with Key Establishment

- Card and card reader share two long term symmetric keys for encryption, decryption and integrity protection
- Goal #1: Provide evidence that smart card and reader know previously shared secret keys and thus are legitimate communication partners
- Goal #2: Smart card and reader agree on two session keys to protect subsequent communications
  - One key for encryption/decryption operations, the second key for computing message authentication codes (MAC)
- Encryption method in CWA 14890-1: 2KTDES in cipher block chaining mode with fixed initialization vector 0
- Length of challenge: 64 bit
- The procedure to establish session keys is defined and the length of the key derivation data is set to 256 bit
- Gives all the information needed to integrate the protocol into an application

Authentication with Key Establishment



From CWA14890-1, message order from top to bottom

Key Derivation

- Smart card and reader have stored both their own and received key derivation data, KDD.c and KDD.r
  - KDD.c and KDD.r are random 256 bit strings
- Both parties compute KDD.rc = KDD.r $\oplus$ KDD.c
- Then, two 32 bit counters are appended to KDD.rc; this gives KDD.rc1 and KDD.rc2
  - First counter has value 1, the second has value 2
- Both parties compute SHA-1(KDD.rc1) and SHA-1(KDD.rc2)
- Session keys are derived from these two hash values

Don’t Trust Your Inputs

- CWA 14890-1 includes two mandatory checks:
  - Does the challenge received equal the challenge stored?
  - Does the message received include the correspondent’s serial number (“self”)?
- Let a rogue card respond with the serial number SN.r of the reader she is communicating with
- This would facilitate a reflection attack

Reflection Attack

- Attacker wants to reply to `GetChall` with the Rand.r the reader will use in `MutAuth`
- Attacker uses the command data from the reader's `MutAuth` command as its final message
- Reader will accept its own command data as a valid response if Rand.c = Rand.r
  - In this case: Rand.c || SN.c = Rand.r || SN.r
- Attacker does not know KDD.r but has reflected the reader's data so KDD.c = KDD.r and KDD.c $\oplus$ KDD.r = 0
- For unpredictable 8 byte challenges, on average 2^{63} attempts for an attack to succeed
- Attacker would then learn all four keys 56 bit keys (224 bits) needed for encryption and MAC computations

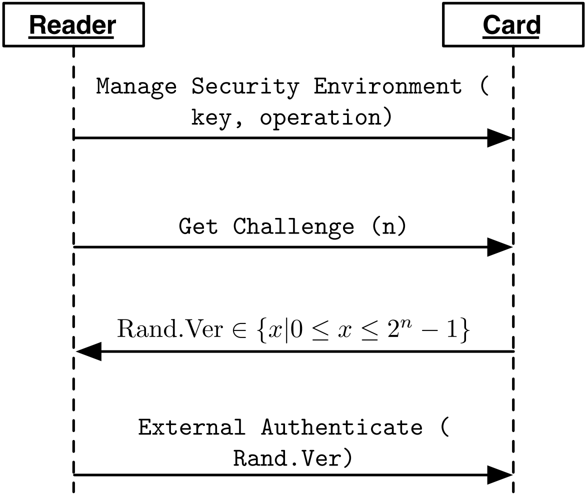
On the Use of XOR

- The rationale for using XOR would be mistrust of the communication partner's random number generator.
- However, when the attacker can reflect (unknown) random data back to its creator, XORing these values can result in a loss of security.
- The XOR operation does not only weaken protocol security it is also unnecessary
- The key derivation data KDD.r and KDD.c could be fed into a hash function and still both parties would not have to trust their partner's random number generator

Challenge-Response Authentication Protocol

- A device may have to authenticate itself to a smart card, e.g. by proving knowledge of certain cryptographic keys
- Authentication success is stored in the smart card's state as long as the card is connected to the reader or the card explicitly removes this information from its state
- A simple challenge-response protocol will do
- Can be obtained by removing the Read Binary command pair and replacing the Mutual Authentication with External Authenticate in the previous protocol

Challenge-Response Authentication Protocol



Adapted from ISO/IEC 9798-2, message order from top to bottom

Differences to ISO/IEC 9798-2

- This protocol is related to the unilateral two pass authentication protocol from ISO/IEC 9798-2
- The protocol specification in ISO/IEC 9798-2 includes an optional identifier to prevent reflection attacks
- In scenarios where reflection attacks cannot occur, the identifier can be omitted
- None of the standards introduced earlier discusses in which situations reflection attacks are impossible

Short Challenges

- In the protocol from ISO/IEC 9798, the verifier starts the protocol by sending a random challenge
- In the smart card protocol above, the card reacts to a command sent by the reader and has to protect itself from attackers that alter command sequences
- The attacker could set the length of the challenge requested in `GetChall` command to one byte, limiting the smart card to choose the challenge from 256 possibilities
- Smart cards could reject requests for random numbers that are too short, but checking `GetChall` command data is not part of the design pattern in CWA 14890-1

Reflection Attack

- Attacker sends a `MSE` command to the smart card setting the shared key for encryption in an `IntAuth` command
- Then, requests Rand.c from the smart card with `GetChall`
- Rand.c is then included as command data in an `IntAuth` command, to which the smart card replies with the encrypted challenge
- Next, the attacker sends a `MSE` command to the smart card setting the same symmetric key for use with `ExtAuth`
- Then, the attacker adversary sends the encrypted challenge with `ExtAuth` to the smart card
- The card will accept the encrypted challenge and thus believe that the attacker is a legitimate reader

Reflection Attack – Defences

- The smart card cannot trust the reader until successful completion of the authentication protocol run
- Smart cards must thus not accept extra commands from the reader before authentication is assured
- Defences in the smart card operating system without changing the protocol
  - Erase random values stored in the card's internal state whenever a `MSE` command arrives
  - Restrict use of symmetric keys so that they could either be used for encryption or decryption but not for both
  - A protocol automaton could detect commands arriving out of sequence in a protocol run
- Slightly change the protocol and include the serial number of the encrypting device when encrypting Rand.c

eCard – Conclusions

- Several standards are crucial for protocol security in eCard applications
- Each standard has its own remit and abstraction layer
- These standards hardly address restrictions or requirements they impose on other standards
- As a result, application designers can take all the right turns and still get lost in the maze of security standards
- Given our observed protocol design vulnerabilities, application designers would require security expertise to successfully negotiate this maze
- Adhering to standards will not automatically result in secure applications

Security Sources in the Web

A Grumpy Old Man?

- Security research needs a lot of very diverse sources, most of which can be found in the web
- I have been noting that students struggle to critically evaluate the sources they find in the web
- The problem seems to have become worse
- May be due to the distance learning mode of the past year
- May be specific to security
- May be my own distorted view of the world
- ...but I am not the only making this observation

Horses for Courses

- The first question to ask yourself: “For whom has this article been written?”
- For the general public?
  - Explanation of a national e-identity scheme
- For potential investors?
  - Be very optimistic about potential applications
- For developers?
  - Explain the world via APIs
- For a project application/review?
  - Strong on motivation and significance
- For security researchers?

Don’t Trust Your Inputs

- Statements that were true once may no longer be true
  - Science / technology / applications may have moved on
- Datasets used for ML research may no longer reflect current use patterns
- An academic idea may never have left the lab
- Anticipated applications many not have materialized
- Peer review is as good as the quality of the peers
  - Be doubly cautious with security papers published in non-security venues
- The analysis of the state-of-the art may be too limited
  - Security is a fashion industry; a literature search based on search terms may not get you very far

Don’t trust what I told you. Verify!