OAuth2.0 协议形式化验证: 使用 AVISPA①

郭丹青

(中国科学院软件研究所, 北京 100190)
(中国科学院大学, 北京 100049)

摘 要: OAuth 协议是一套用于在不同的服务中进行身份认证并且实现资源互访一套协议. 由于关系到用户隐私, 所以OAuth协议的安全性非常重要. 这篇文章的主要贡献是研究OAuth2.0协议文本, 对协议进行抽象, 并且使用验证工具 AVISPA 对抽象后的协议进行建模与验证, 找到协议中会导致隐私泄露的一种攻击模式. 我们在建模过程中提出需将要验证的消息作为双方的对称密码这样一种创新思路. 这种对协议的抽象和验证的方法可以推广到其他安全协议上, 例如在线支付协议等等.

关键词: OAuth2.0; AVISPA; 抽象; 建模; 验证; 攻击模式

Formal Verification on Oauth2.0: Using AVISPA

GUO Dan-Qing

(Institute of software, Chinese Academy of Sciences, Beijing 100190, China)
(University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: OAuth is an protocol used to identify the client, and control resource access. Because it concerns about the privacy of the private resource owner, the security of OAuth protocol is fairly important. This paper mainly research on the OAuth2.0 protocol text, and make an abstraction on it, build an model in AVISPA, an formal verification tool for security protocols , and then verify the model in AVISPA. Finally, we find there is an attack mode that may result in leaking the private resource to attackers. We suggest a way to model the message to be authenticated as a symmetric key, which is innovative. This modelling and verification method we used on analyzing OAuth2.0 can be used in the verification of other security protocols, like the online payment protocol.

Key words: OAuth2.0; AVISPA; abstraction; modelling method; verification; attack mode

1 前言

随着分布式网络服务和云计算的使用越来越多, 服务之间的交互越来越频繁, 第三方应用需要有一套方便的机制能访问到其他服务器的托管资源. 传统的基于客户端-服务器的身份验证模型中[1], 客户端需要使用资源拥有者的私有证书去访问服务器资源, 但是第三方应用得到了私有证书之后会带来很多安全性问题. 而 OAuth[2]协议通过将客户端和资源拥有者的角色进行分离来解决这些问题, 这套流程让第三方可以访问托管资源, 同时不会得到资源拥有者的私有证书, 保证了资源拥有者的账号安全.

OAuth 协议在很多网站上都有使用, 例如我们在一些网站上使用 QQ 账号, 新浪微博账号登陆, 在一些网站导入微博好友关系, 或者将一些信息分享到微博. OAuth 的广泛使用也必然引来大量的攻击, 因此针对 OAuth 的安全性的分析对于寻找 OAuth 的安全隐患, 提升 OAuth 协议的安全性很有必要.

通过形式化[3]的方法, 可以对一个协议进行建模、验证[4], 利用形式分析找出协议的漏洞, 并且提出改进的措施. 本文中我们将使用 AVISPA[2]协议分析形式化工具对 OAuth2.0 协议进行分析, 建模和验证.

2 研究现状

针对 OAuth2.0 的验证工作并不多[5], 对其安全问题

① 收稿时间:2014-03-03;收到修改稿时间:2014-03-24

人工进行分析, 有使用 Cryptoverif[6], 以及使用 Alloy[7] 对 OAuth2.0 进行形式化验证的工作. 不过他们的验证过程仍旧有很大的复杂性; 并且有些流程并没有考虑入内, 例如使用刷新令牌获取新的访问令牌和刷新令牌, 而实际的系统对刷新令牌的使用非常广泛.

AVISPA[8]已经针对很多协议进行了验证[9], 是一个成熟的协议验证工具. AVISPA 官方团队现在已经从 33 个工业级安全协议中发现了 215 个漏洞, 并且根据分析结果发现了很多先前未知的攻击方法. 这些安全协议的验证过程对我们对 OAuth2.0 的验证过程有很大的参考性. 这些结果可以从 AVISPA 的官方网站上找到.

这个工具的底层验证方法基于 SAT(the Boolean Satisfiable) solver[10]对状态空间进行遍历分析, 但是因为对所有的状态空间进行访问需要大量的时间和空间, 在有限的时间和空间下可能不能访问到所有的状态. 虽然现在针对 SAT solver 的计算过程已经有很多改进方法, 但是随着状态空间爆炸, 仍旧可能无法在有限的时间空间对所有状态进行全遍历, 因此我们对协议进行分析的时候, 可能会有些潜在的问题无法找出, 这是目前的一个技术限制[11,12]. 不过对于简化的协议, 状态空间有限, 仍旧可以找出大部分潜在的问题.

AVISAP 主要针对密码协议进行验证[13], 因此建模过程一般与流程一致, 但是并没有人使用 AVISPA 对 OAuth2.0 授权过程进行验证. 因为授权过程中服务器需要对某些消息进行解码并且根据解码结果判断回复消息, 针对这一种流程在其对其他的协议的验证过程中未有成熟建模形式.

3 背景知识

这一章中我们介绍一下相关的背景知识, 主要包括对 OAuth2.0 协议[14]的描述, 对工具 AVISPA 的描述, 对使用的建模语言 HLPSL(High-level protocol specification language: 高层协议形式语言)[15]的描述, 以及对已知的 OAuth2.0 协议漏洞的描述.

3.1 OAuth2.0 协议介绍

OAuth2.0 与 OAuth1.0 版本存在比较大的区别, 这里我们主要介绍 2.0 版本, 其中存在四个角色:

① 资源拥有者(resource owner): 是指受限资源的拥有者, 受限资源存放在资源服务器, 第三方需要资源拥有者授权才能够访问这些受限资源.

② 资源服务器(resource server): 用于存放资源, 这些资源大部分需要对应的资源拥有者授权后才能够被第三方访问.

③ 客户端(client): 第三方应用, 可能需要访问资源服务器上某些用户的受限资源, 此时需要向授权服务器申请授权.

④ 授权服务器(authorization server): 实现授权逻辑, 经过了授权服务器的授权, 客户端才能访问资服务器上的受限资源.

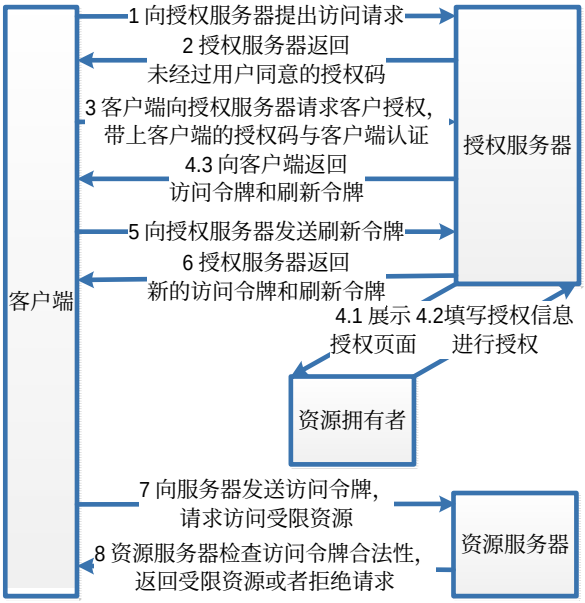

图 1 OAuth 协议流程图

最常见的 OAuth 流程如图 1 所示, 下面解释一下一些其中出现的重要术语:

① 授权许可(authentication grant): 第三方应用向授权服务器请求的授权许可, 标示这项授权服务初始的各方信息, ,其中的信息可以被修改, 可以被扩充.

② 客户端凭证(client credential): 代表了客户端相关的信息, 这些信息主要包括在授权服务成功认证之后的返回地址等等.

③ 令牌(token): 在授权认证成功之后授权服务器给客户端的用于访问资源的识别码. 包括访问令牌(Access token)和刷新令牌(refresh token): 访问令牌可以用于在资源服务器访问相应资源. 有一定的时效; 刷新令牌时效比访问令牌长, 当访问令牌失效后可以用这个令牌向服务器申请更新目前的访问令牌和刷新令牌.

其中 5,6 是可选的过程, 因为刷新验证码的使用很频繁, 我们为了同时验证这个过程, 在正常流程中插入了 5,6 两个步骤.

3.2 AVISPA 简介

AVISPA 是一个针对网络安全协议的自动化验证工具, 可以用于测试. 使用 HLPSL 对协议进行描述, AVISPA 可以自动对这个协议进行翻译, 成为一种中间格式(IF: intermediate format), 之后在后端可以调用四种不同的分析技术对协议进行分析. AVISPA 架构如图 2 所示:

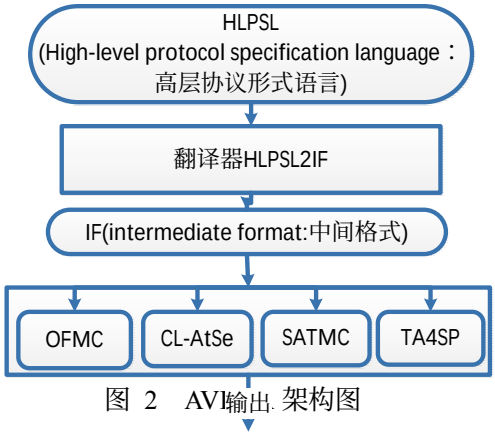

图 2　AVISPA 架构图

AVISPA 的四种后端分析技术如下所示:

① OFMC(on-the-fly model-checker)通过探测系统的变迁, OFMC 能够完成协议的错误查找和有限会话的验证. 它支持密码操作的代数性质的规范, 以支持各种协议模型.

②CL-AtSe(Constriant-Logic-based Atack Searcher)通过强大的简化探测法和冗余排除技术来执行协议, 完成有限会话的错误查找和验证. 它建立在模型化的方式上, 并且是对密码操作的代数性质的延伸.

③ SATMC(SAT-based Model-checker)针对有类型的协议模型, 是建立在通过 IF 语言描述的, 有限域上变迁关系的编码的公式, 初始状态和状态集合的说明代表了整个协议的安全特性, 此公式将反馈给 SAT 状态推导机, 并且建立的任何一个模型都将转化为一个攻击事件.

④ TA4SP(Tree Automata based on Automatic Approximation for the Analysis of Security Protocols)通过树形语言和重写机制估计入侵者的知识, 根据不同的保密特性, TA4SP 能够判断一个协议是否有缺陷, 或者是几个回合的会话是否安全, 可以完成无限协议的验证.

从上面的描述中可以看到, AVISPA 后端验证涉及的技术比较复杂, 但最核心的都是将正常模式和攻击模式都转化为布尔表达式组, 然后利用 SAT solver 对这些布尔表达式进行求解. 如果针对某种攻击模式转换的布尔表达式组有解, 则此攻击有效, 这个协议存在不安全隐患. 但是在求解的过程中可能涉及到很多不同的优化方式, 因此存在多种不同的后台技术. 我们可以选择多种方式对模型进行验证, 也可以单独选择其中一种.

AVISPA 的后端分析技术中的入侵者模型全部基于 Dolev-Yao 攻击者模型. Dolev-Yao 攻击者模型中, 攻击者具有如下能力: ①可以窃听所有经过网络的消息; ②可以组织和截获所有经过网络的消息; ③可以存储所获得的或者自身创造的消息; ④可以根据存储的消息伪造消息, 并且发送该消息; ⑤可以作为合法的主体参与协议的运行. AVISPA 会自动为最终模型添加攻击者模式并且验证相关攻击是否可能发生.

3.3 HLPSL 简介

利用 AVISPA 进行建模验证, 使用的是 HLPSL 语言. HLPSL 是基于角色的一种语言, 我们需要根据系统行为定义不同角色, 然后针对每个角色阐明他们的特定数据以及行为, 角色之间通过消息的传递来进行交互.

模型由很多角色(Role)以及目标(Goal)组成, 每个角色有输入参数, 局部变量以及行为. 完整的模型除了参与协议的不同角色意外, 还含有一些特殊的角色: ①会话(session), 指代构成会话流的组合角色, 这些会话可以调用不同的掐角色, 构建完整的会话流; ②环境(environment)定义了模型的主体, 包括这个环境中的基本变量, 角色, 攻击者知识, 正常会话以及攻击会话.

模型中我们可以定义一些系统安全目标: 在流程中同时使用 witness 和 request 用于检查某些信息的可信度(可信是指要求某些消息必须从某些主体发出而不能是其他主体), 使用 secret 表示在规定的几个角色中保持某些信息的私密性.

系统也需要进行定义入侵者, 包括入侵者所拥有的知识和部分正常行为: 拥有的知识定义在 intruder_knowledge 里面, 可以是环境中的其他主体, 也可以是密钥或者消息; 行为定义在入侵者 session 里

面, 一般是正常的入侵者会话过程. 系统会自动根据 Dolev-Yao 模型添加进入侵者的各种入侵行为, 并且进行分析.

最后系统还要定义一些验证目标(goal), 主要目标是认证性和私密性, 分别用 authentication_on 和 secret 表示, 分别表示某些消息一定是从某主体传递到另一主体, 以及某些消息在某些固定主体之间的私密性.

3.4 **OAuth2.0** 协议漏洞

已经有人针对 OAuth2.0 协议进行非形式化的安全研究[16], 目前已知的 OAuth2.0 协议的漏洞存在两种形式:

(1) 根据 Ryan Paul 在[17]中所述, 当桌面应用使用 OAuth2.0 的时候可能存在漏洞. 当客户端是一个网络应用的时候, 客户端凭证放在安全服务器里面, 攻击者很难获取; 但是如果客户端是一个桌面程序, 客户端的客户端凭证存放在本地客户端里面, 攻击者可以通过逆向工程找到这个客户端凭证然后伪装这个客户端, 欺骗资源拥有者登录从而得到资源拥有者的受限资源. 现在的推特客户端就存在这样的问题.

(2) 中国的知道创宇安全研究团队证明中国大部分网站存在 OAuth 漏洞[18], 这个漏洞是攻击者通过截取含有正常客户端发送的授权码和客户端凭证组合成的回调地址, 并且诱骗登录的用户发送 HTTP 请求到这个地址, 此时由于用户与服务器的 session 还存在, 这个时候就会自动进行认证, 从而骗取到用户的授权, 窃取用户的受限资源.

总结一下之上的两个漏洞, 可以知道最关键的点是在其他人可以窃取客户端凭证之后伪装原应用诱骗用户登录, 从而得到资源拥有者的信任, 从而得到受限资源.

4 建模

为了使用 AVISPA 进行验证 OAuth, 我们使用一种建模技术来模拟信息的验证. 在模型中我们将两方之间的需要验证的私密信息表示为两者的对称密钥, 例如 A 向 B 发去访问令牌 AT 以请求对受限资源进行访问, B 要验证这个访问令牌并且根据验证决定向 A 发回受限资源还是拒绝其请求, 我们就将 A 向 B 发送一个用 AT 加密的消息, B 用解密的方法对 AT 进行验证.

我们先用一种形式化的方法描述整个流程, 其中 A->B 代表主体 A 向主体 B 发送消息, {X}Kab 代表使用密钥 Kab 对消息 X 加密. 我们使用 C 表示客户端, O 表示资源拥有者, A 表示授权服务器, S 表示资源服务器, 使用 Koa 表示资源拥有者和授权服务器之间的, 使用 Kca 表示客户端和授权服务器之间的通信密钥, 使用 Ka 表示授权许可, 使用 Kc 表示客户端凭证, 使用 Ko 表示资源拥有者授权信息(密码等内容组成的用户凭证), 使用 Kcs1 表示访问令牌, 使用 Kcs2 表示刷新令牌, 用 T 表示最后的内容最后整个流程表示如下:

1. C->A:{Kapp}Kca
2. A->C:{Ka}Kca
3. C->A:{Ka,Kc}Kca
4. A->O:{Ka,Kc}Koa
5. O->A:{Ko}Koa
6. A->C:{Kcs1,Kcs2}Kc
7. C->A:{Kcs2}Kcs2
8. A->C:{Kcs1,Kcs2}Kc
9. C->S:{Kc}Kcs1
10. S->C:{T} Kc

我们在 HLPSL 模型中使用六个 role 来表示系统中涉及的四个角色

(1) 一个 client 用于表示客户端, 主要的迁移如下所示:

① State = 0 /\RCV_CA(start)=|>

State':=2 /\ Napp':=new() /\SND_CA({Napp'}_Kca)

表示客户端在接收到系统开始信号(start)之后, 向授权服务器发送授权请求(Napp).

② State = 2 /\ RCV_CA({Na'}_Kca) =|>

State':= 4

/\Kc':=new()/\ SND_CA({Na'}_Kca,Kc')

表示在接收到授权服务器发送过来的未经用户授权的授权码(Na)之后, 向服务器发送更改后的授权码(Na)并且将客户端凭证(Kc)发送给授权服务器.

③ State = 4

/\ RCV_CA({Ncs1'}_Kc,{Ncs2'}_Kc)

=|>State':=6/\SND_CA({Ncs2'}_Ncs2')

表示在接收到授权服务器的访问令牌(Ncs1)和刷新令牌(Ncs2)之后向授权服务器发送刷新令牌(Ncs2), 以便获取新的访问令牌和刷新令牌.

④ State= 6/\ RCV_CA({Ncs1'}_Kc,{Ncs2'}_Kc)

=|>State':=8/\SND_CS({Kc}_Ncs1')

表示在接收到授权服务器新的访问令牌(Ncs1)和刷新令牌(Ncs2)之后向资源服务器发送访问令牌(Ncs1), 以便获取受限资源.

(2) 一个 server 用于表示资源服务器, 主要的迁移如下所示:

① State = 7/\ RCV_SA({Ncs1'}_Ksa)

=|> State':=8

表示接收到授权服务器发送过来的相关客户端的访问令牌(Ncs1).

② State = 8/\ RCV({Kc'}_Ncs1')=|> State':=9/\

T':=new()/\SND({T'}_Kc')/\secret(T',t,{C,S})

/\witness(S,C,client_server_kc,Kc')

表示在接收到客户端发来的访问令牌(Ncs1)以及客户端凭证(Kc)时候向相关客户段发送受限资源(T).

(3) 一个 owner 用于表示资源拥有者, 主要的迁移如下所示

State = 5/\ RCV({Na'}_Kao,{Kc'}_Kao)

=|> State':=7/\No':=new()/\SND({No'}_Kao)

表示在授权服务器展示了相关授权申请(Na)和申请授权的客户端相关信息(Kc)之后, 资源拥有者向授权服务器发送授权信息(No).

(4) 一个 authorization 用于表示授权服务器

① State=1/\ RCV_CA ({Napp'}_Kca)

=|> State':=3 /\ Na':=new()/\SND_CA({Na'}_Kca)

表示在接收到客户端发来的授权申请(Napp)后向客户端发送未经授权的授权码(Na).

② State=3/\ RCV_CA ({Na'}_Kca,Kc')

=|> State':=5 /\SND_AO({Na'}_Kao,{Kc'}_Kao)

表示在接收到客户端发来的授权码(Na)以及客户端凭证(Kc)时, 向资源拥有者展示相关信息(Na,Kc), 引导用户进行授权.

③ State=5/\ RCV_AO ({No'}_Kao)

=|> State':=7 /\Ncs1':=new()/\Ncs2':=new()/\

SND_CA({Ncs1'}_Kc,{Ncs2'}_Kc)

表示在接收到资源拥有者的授权信息(No)之后, 将访问令牌(Ncs1)和刷新令牌(Ncs2)发回提出请求的客户端.

④ State=7/\ RCV_CA

({Ncs2'}_Ncs2')=|> State':=9 /\Ncs1':=new()

/\SND_CA({Ncs1'}_Kc,{Ncs2'}_Kc)

/\SND_AS({Ncs1'}_Ksa)

表示在接收到刷新验证码(Ncs2)并且解析成功之后向相关客户端发送回新的访问令牌(Ncs1)和刷新令牌(Ncs2).

(5) 一个 session 用来定义之前四个 role 同时运行:

client (C,A,O,S, Kcs, Kca ,

SND_CA, RCV_CA, SND_CS, RCV_CS)

/\ owner(C,A,O,S, Kao, SND_OA, RCV_OA)

/\ server(C,A,O,S, Kcs ,Ksa ,

SND_SC, RCV_SC,RCV_SA)

/\ authorization(C,A,O,S,Kao,Kca,Ksa,

SND_AC, RCV_AC,SND_AO,RCV_AO,SND_AS)

(6) 一个 environment 用来定义整个系统, 其中定义了一个正常 session 和一个攻击者 session:

session(c,a,o,s,kcs, kca, kao,ksa)/\

session(i,a,o,s, kis,kia,kao,ksa)

其中 i 为攻击者, 这里的攻击者不知道正常客户端与授权服务器和资源服务器的通信信道密码, 因此无法破译这两个通信信道的消息.

从之上的建模中可以看出, 当资源服务器需要对客户端发来的访问令牌(Ncs1)进行验证的时候, 我们将访问令牌作为客户端发来的消息的私钥进行加密({Kc}_Ncs1), 之后资源服务器需要对这个消息解密, 如果能正确解密则证明 Ncs1 是正确的. 这个模拟过程实现了资源服务器验证访问令牌(Ncs1)的流程.

5 验证与结果

我在 AVISPA 中使用 ofmc 模式对该模型进行了验证, 最后运行的结果如图 3 所示, 确实存在相应的攻击模式, 如果客户端凭证存储不安全的话, 最后可以利用客户端凭证骗取资源拥有者的信任并且成功获取到资源拥有者的受限资源.

攻击者在这个过程中伪装有效客户端, 最后的攻击者模式串解释如下:

```
root@ubuntu:/opt/avispa-1.1# bash avispa /mec
fmc
% OFMC
% Version of 2006/02/13
SUMMARY
  UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  /opt/avispa-1.1/testsuite/results/final.if
GOAL
  secrecy_of_t
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.15s
  visitedNodes: 148 nodes
  depth: 5 plies
ATTACK TRACE
i -> (a,9): {x236}_kia
(a,9) -> i: {Na(1)}_kia
i -> (a,9): {Na(1)}_kia
(a,9) -> i: {Na(1)}_kao
i -> (a,9): {Na(1)}_kao
(a,9) -> i: {Ncs1(3)}_dummy_sk
i -> (a,9): {x281}_x281
(a,9) -> i: {Ncs1(4)}_dummy_sk,{Ncs1(4)}_ksa
i -> (s,3): {Ncs1(4)}_ksa
i -> (s,3): {x308}_x307
(s,3) -> i: {T(6)}_x308
i -> (i,17): T(6)
i -> (i,17): T(6)
```

图 3 使用 AVISPA 对 OAuth2.0 验证结果

(1) i -> (a,9): {x236}_kia: 攻击者首先向授权服务器提出请求

(2) (a,9) -> i: {Na(1)}_kia: 授权服务器向攻击者返回未经用户同意的授权码

(3) i -> (a,9): {Na(1)}_kia: 攻击者向授权服务器发送之前得到的授权码和通过不正当手段取得的客户端凭证

(4) (a,9) -> i: {Na(1)}_kao: 授权服务器向用户请求对得到的授权码进行授权

(5) i -> (a,9): {Na(1)}_kao: 用户对得到的授权码进行授权, 由于看到的是客户端凭证是正常客户端, 所以用户同意授权.

(6) (a,9) -> i: {Ncs1(3)}_dummy_skk: 授权服务器将正常的授权发送给持有客户端凭证的攻击者.

(7) i -> (a,9): {x281}_x281: 攻击者将上面得到的刷新令牌刚发给授权服务器.

(8) (a,9)

-> i: {Ncs1(4)}_dummy_sk,{Ncs1(4)}_ksa: 授权服务器将新的访问令牌和刷新令牌发回攻击者

(9) i -> (s,3): {Ncs1(4)}_ksa: 授权服务器告知资源服务器相关的访问令牌

(10) i -> (s,3): {x308}_x307: 攻击者向资源服务器发送访问令牌

(11) (s,3) -> i: {T(6)}_x308: 服务器验证该访问令牌后向提出请求的攻击者发送受限资源

(12) i -> (i,17): T(6): 攻击者得到受限资源

6 总结与展望

通过使用 AVISPA 对 OAuth2.0 协议的建模和验证, 我们找到了一种导致用户私密信息泄露的攻击模式, 这结果与披露的攻击行为一致.

在建模的过程中, 我们将需要服务器进行验证的消息作为对称密钥来进行建模, 这种建模方式为我们验证其他需要解析消息并且根据解析结果执行相关逻辑的协议提供了一种建模思路.

这个工作对我们使用形式化的工具验证其他协议有一定的借鉴意义, 例如支付协议, 这些协议都对安全性要求比较高, 需要通过形式化的验证确保其安全性.

我们的后续工作主要包括以下三个方面:

① 我们对整个协议进行了抽象, 因此模型中隐藏了太多参数细节, 但是可能某些参数也会引起整个协议的问题, 之后我们要要添加详细的参数丰富模型的细节, 并且进行验证, 希望能够找出更多协议的漏洞.

② 使用同样的思路验证其他对可靠性要求比较高的协议, 例如支付协议.

③ 继续深入研究其他协议验证工具, 查看其他工具在验证该协议上的效果, 以便指导其他协议的验证.

参考文献

1 Miller SP, et al. Kerberos authentication and authorization system. Project Athena Technical Plan. 1987.

2 Hardt D. The OAuth 2.0 authorization framework. 2012.

3 Clarke EM, Wing JM. Formal methods: State of the art and future directions. ACM Computing Surveys (CSUR),1996, 28(4): 626–643.

4 卿斯汉.安全协议.北京:清华大学出版社,2005.

5 Chari S, Jutla CS, Roy A. Universally Composable Security Analysis of OAuth v2. 0. IACR Cryptology ePrint Archive, 2011: 526.

6 Xu X, Niu L, Meng B. Automatic verification of security properties of OAuth 2.0 protocol with cryptoverif in computational model. Information Technology Journal, 2013, 12(12).

7 Pai, Suhas, et al. Formal verification of OAuth 2.0 using Alloy framework. 2011 International Conference on Communication Systems and Network Technologies (CSNT). IEEE, 2011.

8 AVISPA webiste. http://www.avispa-project.org/

9 Armando A, Basin D, Boichut Y, et al. The AVISPA tool for the automated validation of internet security protocols and applications. Computer Aided Verification. Springer Berlin Heidelberg, 2005: 281–285.

10 Moskewicz MW, Madigan CF, Zhao Y, et al. Chaff: Engineering an efficient SAT solver. Proc. of the 38th Annual Design Automation Conference. ACM. 2001. 530–535.

11 Holzmann GJ. Algorithms for automated protocol verification. AT&T Technical Journal, 1990, 69(1): 32–44.

12 Meadows CA. Formal verification of cryptographic protocols: A survey. Springer Berlin Heidelberg, 1995.

13 Viganò L. Automated security protocol analysis with the AVISPA tool. Electronic Notes in Theoretical Computer Science, 2006, 155: 61–86.

14 Hammer-Lahav DE, Hardt D. The OAuth2. 0 Authorization Protocol. IETF Internet Draft. 2011.

15 AVISPA Team HLPSL Tutorial. European Community under the Information Society Technologies Program. http://www.AVISPA-project.org

16 Lodderstedt T, McGloin M, Hunt P. OAuth 2.0 threat model and security considerations. http://tools.ietf.org/html/rfc6819. 2013.

17 Paul R. Compromising Twitter's OAuth security system. http://arstechnica.com/security/2010/09/twitter-a-case-study-on-how-to-do-oauth-wrong/.

18 Ping0s. OAuth 认证机制中普遍的安全问题, http://www.freebuf.com/articles/web/5997.html.