改进的 OAuth2.0 协议及其安全性分析①

陈 伟, 杨伊彤, 牛乐园

(中南民族大学 计算机科学学院, 武汉 430074)

摘 要: 随着OAuth2.0协议的广泛应用, 其安全性受到了人们的重点关注. 为了增强OAuth2.0协议的安全性, 本文首先引入数字签名技术, 提出一个改进的 OAuth2.0 协议. 它支持授权服务器对资源拥有者和客户端的身份认证. 并且在计算模型下基于 Blanchet 演算, 应用一致性对授权服务器认证资源拥有者和客户端进行建模, 最后使用自动化工具 CryptoVerif 分析和证明了其认证性.

关键词: 认证性; 计算模型; 自动化验证; 安全协议

Improved OAuth2.0 Protocol and Analysis of its Security

CHEN Wei, YANG Yi-Tong, NIU Le-Yuan

(School of Computer, South-Center University for Nationalities, Wuhan 430074, China)

Abstract: With the wide applicaitons of OAuth2.0 protocol, people have payed a special attention to its security. In order to enhance its security, in this study the digitial signature firstly is introduced, then an improved OAuth2.0 protocol is proposed which has the authentication from authorization server to client and authorization server to resource owner. At the same time based on the Blanchet calculus in computaional model, the correspondence is applied to model the authentication from authorization server to client and authorization server to resource owner, and finally the authentication is proved by CryptoVerif.

Key words: authentication; computational model; automatic verification; security protocol

OAuth(开放授权)协议是一个开放标准, 它是OpenID的一个补充, 允许用户不提供用户名和密码让第三方应用访问自己在某一网站上的资源(如照片、社交好友信息等). OAuth2.0[1]是“用户验证和授权”标准的最新版本.

国内的百度开放平台、腾讯开放平台等开放平台都支持 OAuth 2.0 协议. 国外的 Face book 的 Graph API、Google 的 Google API 也都支持 OAuth2.0. 由于 OAuth2.0 协议的普遍应用, 其安全性受到了人们的重点关注. Chari 等[2]使用通用可复合安全框架形式化 OAuth2.0 协议, 指出最理想的对第三方授权的方法是基于口令的认证方案. Corella-Lewison[3]也对 OAuth2.0 的抗拒绝服务攻击进行了分析. 时子庆等[4]分析了 OAuth2.0 的刷新令牌, 给出 OAuth2.0 服务器端的设计方案. Xu 等[5]对 OAuth2.0 的安全属性进行了分析, 提出在没有安全机制(如 SSL 协议)支持的情况下, OAuth2.0 协议不提供认证性.

目前, 对于安全协议[6]的形式化分析存在两种方法[7]. 一种是符号分析方法[8], 该方法将协议中使用的密码系统看做是完美的无漏洞的黑盒, 并在此基础上对安全协议进行分析和证明[9]. 另一种是计算方法, 该方法假设协议使用的密码系统是不完美的. 相对于符号方法, 计算方法更接近于实际应用的情况, 但难于实现自动化验证. 2006 年, Blanchet 提出了一种基于进程概率演算[10]的计算模型并开发了自动化工具 CryptoVerif[11]. CryptoVerif 是第一个基于计算方法的自动化验证工具.

故本文首先引入数字签名技术, 提出一个改进的 OAuth2.0 协议. 它支持授权服务器对资源拥有者和客户端的身份认证. 并且在计算模型下基于 Blanchet 演

① 基金项目:国家民族事务委员会自然科学基金(12ZN008)

收稿时间:2013-08-08;收到修改稿时间:2013-10-17

算, 应用 correspondence(一致性)对授权服务器认证资源拥有者和客户端进行建模, 最后使用自动化工具 CryptoVerif 分析和证明了其认证性.

1　改进的OAuth2.0协议

改进的 OAuth2.0 定义了四个角色: 资源拥有者、资源服务器、客户端及授权服务器. 资源拥有者是一个能够授权访问受保护资源的实体. 当资源拥有者是一个人时, 称其为终端用户. 资源服务器管理着受保护的资源, 它能够接收和响应使用访问令牌的资源请求. 客户端一般是一个应用程序, 代表资源拥有者及其授权来请求受保护的资源. 授权服务器具有认证资源拥有者和发布访问令牌的功能. 当其成功认证资源拥有者并获得资源拥有者的授权后, 将发布访问令牌给客户端.

改进的 OAuth2.0 协议如图 1 所示. 首先, 终端用户在授权服务器上进行注册, 在浏览器上填写自己将在授权服务器上注册的参数 username 和 password 并用私钥 sku 签名后提交, 完成注册过程. 客户端也需在授权服务器上进行注册, 客户端向授权服务器提交客户端类型(client Type)、重定向 URI(redirect_uri)等必须的信息来完成注册. 授权服务器接收到客户端的注册请求后, 生成客户端标识符 client_id 和客户端密钥 client_secret 并发送给客户端. 这两个数据是相互绑定的, 在之后的请求中用来证明客户端的身份.

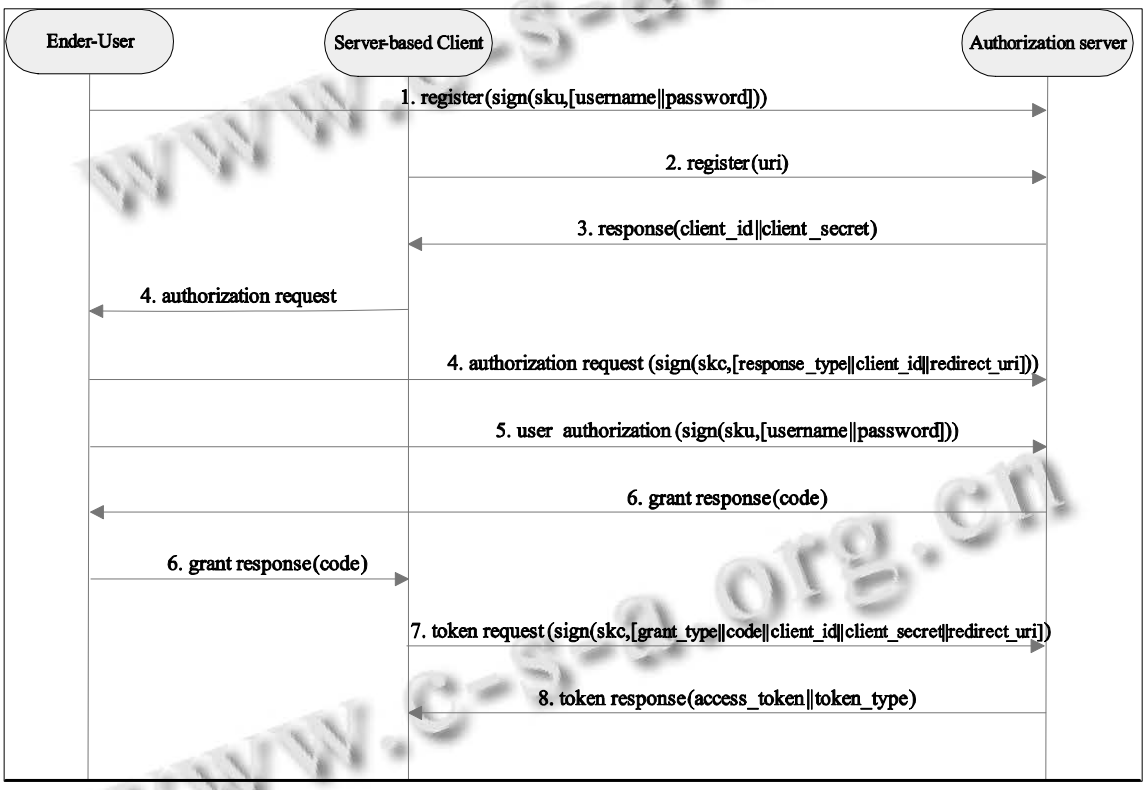

图 1　改进的 OAuth2.0 协议

改进的 OAuth2.0 协议由以下几个步骤组成.

1.1 客户端授权许可请求

当客户端需要获得终端用户(即资源拥有者)在资源服务器上的资源时, 首先生成参数 response_type, 同时加上 redirect_uri 及注册时从授权服务器获得的 client_id. 客户端将这些参数签名后添加到终端用户授权 endpoint URI 的查询参数部分, 并通过 HTTP 的重定向机制将终端用户的浏览器重定向到终端用户授权 endpoint.

授权服务器接收到客户端的请求后, 确认该请求是否具备必须的参数以及这些参数是否有效. 若有效, 则授权服务器验证资源拥有者(通过资源拥有者的 username 和 password 参数)并获取用户的授权.

1.2 授权服务器响应授权许可请求

授权服务器在响应授权许可请求前, 需先获得资源拥有者的授权. 授权服务器发送授权验证消息要求

验证资源拥有者的身份，在收到资源拥有者的 username 和 password 参数后验证是否正确，然后询问资源拥有者是否同意授权. 资源拥有者同意授权后，授权服务器生成参数 code，并将其与客户端授权许可请求时的 client_id 和 redirect_uri 参数绑定. 然后将其添加到客户端重定向 URI 的询问部分，发送给客户端，并将终端用户的浏览器重定向到客户端 redirect_uri 的 URI 值. 响应授权许可请求步骤完成.

1.3 客户端请求访问令牌

客户端获得授权码后，还必须通过授权码在授权服务器上获得访问令牌才能访问资源服务器上的资源. 客户端生成 grant type 参数(其值为“authorization code”)，并与 code、client_id、client_secret、redirect_uri 等参数一起签名并添加到授权服务器的令牌 endpoint 的询问部分，发送给授权服务器. 其中 code 的值必须为响应授权许可请求中接收到的 code 值，client_id 和 redirect_uri 的值必须与授权许可请求时的值相同，client_secret 的值必须为与 client_id 绑定由授权服务器发布的值.

1.4 授权服务器响应令牌请求

授权服务器接收到客户端的令牌请求消息后，需做以下验证才能响应令牌请求:

① 验证客户端的身份，授权服务器检查签名是否正确，检查 client_id、client_secret 参数的值是否正确并存在绑定关系.

② 验证 code，授权服务器检查 grant type 的值是否为“authorization code”，然后检查 code 的值是否正确，最后还需检查 code 与 client_id 是否存在绑定关系.

③ 验证 redirect_uri，授权服务器需验证 redirect_uri 是否注册，并且检测 redirect_uri 的值是否与客户端授权许可请求时的 redirect_uri 的值相同.

以上验证均通过，则授权服务器响应客户端的访问令牌请求. 授权服务器通过发布访问令牌来响应该请求，授权服务器生成 access token、token type 等参数发送给客户端. access token 即访问令牌，客户端可以使用它与资源服务器交互来获取资源拥有者授权的资源. Token type 则是对访问令牌 access token 的描述，若客户端未能理解 token type 的内容则无法获得所需的资源.

客户端获得访问令牌后，就能使用它来取得资源服务器上已授权的资源. 客户端与资源服务器交互获得授权资源的方式不属于 OAuth2.0 协议的范围，由资源服务器和客户端协商完成. 至此整个协议过程完成.

2 应用Blanchet演算对改进的OAuth2.0形式化建模

2.1 Blanchet 演算和 CryptoVerif 工具

Blanchet 演算是一种概率多项式演算，用来建模安全协议. 其中，消息表示成字串，密码原语表示为对字串的函数操作. Blanchet 演算包含项和进程，其语法规则如下.

在 Blanchet 演算中，c 为通道名称的集合，消息通过输入和输出通道传递到网络上，从而能被攻击者获取. 项 $x[M_1,...,M_m]$ 表示变量访问，$M_1,...,M_m$ 为变量数组 x 的索引. 项 $f[M_1,...,M_m]$ 返回函数 f 作用于 $M_1,...,M_m$ 的结果.

进程分为输入进程 Q 和输出进程 P. 输入进程 Q 在一个通道上接收消息. 一般包括空进程、并行进程 $Q|Q'$、进程复制 $!^{i\le N}Q$、接收消息 $c[M_1,...,M_l](x_1[i]:T_1,...,x_k[i]:T_k);P$. 不做任何事情；$Q|Q'$ 表示进程 Q 和 Q' 并行执行；$!^{i\le N}Q$ 复制 N 份并行的 Q 进程；$c[M_1,...,M_l](x_1[i]:T_1,...,x_k[i]:T_k);P$ 表示通道 $c[M_1,...,M_l]$ 接收到消息 $x_1[i],...,x_k[i]$，$T_1,...,T_k$ 为消息参数的类型，然后执行进程 P. 输出进程 P 包括 $\overline{c[M_1,...,M_l]}(x_1[i],...,x_k[i]);Q$、$new\ x[i_1,...,i_m]:T;P$、$let\ x[i_1,...,i_m]:T=M\ in\ P$、$if\ defined(M_1,...,M_l)\wedge M\ then\ P\ else\ P'$ 以及 $find\left(\oplus_{j=1}^{m}u_{j1},\cdots,u_{jm_j}[i]\le n_{jm_j}\ suchthat\ defined\left(M_{j1},\cdots,M_{jl_j}\right)\wedge M_j\ then\ P\right)else\ P'_j$. $\overline{c[M_1,...,M_l]}(x_1[i],...,x_k[i]);Q$ 在通道 $c[M_1,...,M_l]$ 上输出消息 $x_1[i],...,x_k[i]$ 后执行 Q；$new\ x[i_1,...,i_m]:T;P$ 随机产生一个 T 类型的数，存储为 $x[i_1,...,i_m]$ 后执行 P；$let\ x[i_1,...,i_m]:T=M\ in\ P$ 将项 M 的字串的值存储为 T 类型的变量 $x[i_1,...,i_m]$ 后执行 P；$if\ defined(M_1,...,M_l)\wedge M\ then\ P\ else\ P'$ 判断项 $M_1,...,M_l$ 是否定义且 M 是否成立，判断为真则执行 P，否则执行 P'；$find\left(\oplus_{j=1}^{m}u_{j1},\cdots,u_{jm_j}[i]\le n_{jm_j}\ suchthat\ defined\left(M_{j1},\cdots,M_{jl_j}\right)\wedge M_j\ then\ P\right)else\ P'_j$ 是访问数组，是否存在索引 $u_{j1},...,u_{jm_j}[i]\le n_{jm_j}$ 使得 $M_{j1},...,M_{jl_j}$ 已定义和 M_j 成立，若存在则执行 P_j，否则执行 P'.

CryptoVerif 是一款基于计算模型的自动化分析安全协议的工具，能够直接证明安全协议是否具有定义的安全属性. 其中攻击者被定义为多项式时间图灵机，

密码原语则为对消息字串处理的函数. 分析安全协议的安全属性大致分为三个步骤: 首先建模参与协议的各方和攻击者, 用 Blanchet 演算描述各参与进程. 其次建模安全目标, 即定义要证明的安全属性. 最后证明安全目标, CryptoVerif 使用转换规则, 对建立的模型不断规约, 若能在概率多项式时间内证明安全目标, 则表明协议具有该安全属性.

2.2 定义的事件及证明目标

图 2 给出了证明目标, *event Authorization*(*three*, *y*) ==> *User*(*three*, *x*) 表明在事件 *event Authorization*(*three*, *y*) 发生时, 事件 *User*(*three*, *x*) 一定已发生, 用来验证授权服务器认证终端用户. 同样 *event Authorizationa*(*y*) ==> *Client*(*x*) 用来验证授权服务器认证客户端.

```
event User(host,seed).
event Client(seed).
event Authorization(host,username).
event Authorizationa(code).
query x:seed, y:username;
event Authorization(three,y) ==> User(three,x).
query x:seed, y:code;
event Authorizationa(y) ==> Client(x).
```

图 2　定义的事件及证明目标

2.3 初始化进程

初始进程如图 3 所示, *start*() 表示协议启动, *Authorization_server* 进程表示协议中的授权服务器实体, *client* 进程表示客户端实体, *web_server* 进程表示客户端的 *web* 服务器, *user* 进程则表示终端用户. 由于现实中授权服务器与客户端和终端用户往往是一对多的关系, !*N*1 *client* 表示多个 *client* 进程并行运行, 模拟现实环境.

```
process
    in(start,());
    new ru:keyseed;
    let sku:skey = skgen(ru) in
    let pku:pkey = pkgen(ru) in
    new ra:keyseed;
    let ska:skey = skgen(ra) in
    let pka:pkey = pkgen(ra) in
    new rc:keyseed;
    let skc:skey = skgen(rc) in
    let pkc:pkey = pkgen(rc) in
    c̄ < pku, pka, pkc >;
    ((!N Authorization_server)
    |(!N1 client)|(!N3 user)|(!N2 web_server))
```

图 3　初始化进程

2.4 使用的密码原语及通道定义

图 4 给出了所使用的密码原语及定义的通道. 选用 Cryptoverif 中预定义的一个概率公钥签名方案 UF_CMA_signature, 它具有抵抗选择消息攻击的属性. *channel* 定义用于消息传递的通道.

```
proba Psign.
proba Psigncoll.
expand UF_CMA_signature(keyseed, pkey, skey, message,
signature, seed, skgen, pkgen, sign, check, Psign, Psigncoll).
channel start,c,c1,c2,c3,c4,c5,c6,c7,c8,c9,
c10,c11,c12,c13,c14,c15,c16,c17,c18,c19,20,c21,c22.
```

图 4　密码原语与通道定义

2.5 终端用户进程

终端用户进程如图 5 所示, *user* 通过语句 *new name*:*username* 生成自己的用户名 *name* 及 *new pass*:*password* 生成口令 *pass*. 通过 *c*1 通道将 *name* 和 *pass* 签名后传递给授权服务器. *c*2(*string_form_author*:*cleartext*) 表示 *user* 进程通过 *c*2 通道接收到来自 *Authorization_server* 的消息 *string_form_author*. 若该消息为询问消息 *Authentication_message*, 则同意授权, *event User*(*r*4) 发生. *user* 进程通过 *c*3 通道将已注册的 *name* 和 *pass* 连接并签名后发送给授权服务器来验证身份, 进行授权后进程结束.

```
let user =
c(); new name:username;
new pass:password; new r1:seed;
let messageone:message = concatA(name, pass) in
c̄1 < one, messageone, sign(messageone, sku, r1) >;
c2(string_form_author:cleartext);
if string_form_author =
Authentication_message then
new r2:seed; event User(r2);
c̄3⟨three, concatA(name, pass),
   sign(concatA(name, pass), sku, r2)⟩.
```

图 5　终端用户进程

2.6 客户端进程

客户端进程如图 6 所示, *client* 进程首先 *new uri*:*redirect_uri* 生成重定向 URI 参数 *uri* 并由通道 *c*4 输出到授权服务器注册. 注册通过后在 *c*5 通道接收到 *client_id* 参数和 *client_secret* 参数. *client* 需要请求终端用户授权时, 将 *code_response*、*clientid*_

和 *uri* 连接并用私钥签名后传递给授权服务器. 获得授权后, 授权服务器响应授权许可请求将授权码发布给 *web* 服务器, 客户端在 *c*7 通道接收到 *web* 服务器转发的 *code_fromweb* . 然后将收到的授权码与 *clientid_*、*client*sec*ret_*、*uri*、*code_grant* 连接并私钥签名后通过 *c*8 通道发送给授权服务器, 请求获取访问令牌, 同时 *event Client*(*r*4) 发生. 授权服务器验证客户端的身份并验证客户端发送的参数的正确性, 全部验证通过后授权服务器发布访问令牌. *client* 通过 *c*9 接收到授权服务器发布的访问令牌 *accesstoken* 和令牌类型 *tokentype_a* . 客户端就能够使用访问令牌获取资源服务器上的受保护资源.

```
let client =
c0();   new uri : redirect_uri;
c4 < uri >; c5(= A,id_secret : message);
let concatB(clientid_ : client_id,
client secret_ : client_secret) = id_secret in
let messagetwo : cleartext =
concatE(code_response,clientid_,uri) in
new r3 : seed;
c6 < two,messagetwo,sign(messagetwo,skc,r3) >;
c7(code_fromweb : code);
let messagefive : message = concatF(code_grant,
code_fromweb,clientid_,client secret_,uri) in
new r4 : seed;   event Client(r4);
c8 < five,messagefive,sign(messagefive,skc,r4) >;
c9(accesstoken : access_token,tokentype_a : token_type).
```

图 6 客户端进程

2.7 Web 服务器进程

web_server 进程起转发消息的作用. 将 *client* 的授权许可请求转发给授权服务器, 授权服务器响应授权许可请求后, 将接收到的授权码转发给 *client* 进程.

```
let web_server =
  c10(=two,messagetwo_c : message,
      signmtwo_c : signature);
  c11 < web,messagetwo_c,signmtwo_c >;
  c12(= web,code_from_a : code);
  c13 < code_from_a >.
```

图 7 Web 服务器进程

2.8 授权服务器进程

授权服务器首先接收客户端的注册, 由 *c*14 接收客户端的注册信息 *urireg* , 之后生成客户端标识符 *clientid* 和客户端密钥 *client*sec*ret* . 同时绑定这两个参数并存储为 *id*sec*retsuccess* , 绑定 *clientid* 与 *urireg* . 在 *c*15 通道传输 *clientid* 和 *client*sec*ret* 给注册的 *client* . 授权服务器同样接受资源拥有者的注册, 在 *c*16 通道接收到资源拥有者的注册信息. 检验签名后得到注册用户名 *namereg* 、注册口令 *passwordreg* . *client* 要获得资源拥有者的授权, 需向授权服务器发起授权许可请求, 由授权服务器作为媒介来获得授权. 授权服务器在 *c*17 通道接收到授权许可请求消息 *twomessage* 及签名 *twosign* , 用公钥 *pkc* 验证签名后获得参数 *responsetypeone* 、*clientidone* 及 *urione* . 授权服务器先检测 *responsetypeone* 的值是否是常量 *coderesponse* , 验证通过则在 *c*18 通道向资源拥有者发送授权消息 *Authertication_message* , 要求 *user* 进行授权. *user* 在授权之前需验证身份, 授权服务器在 *c*19 通道接收到 *user* 的验证消息 *threemessage* 和签名 *threesign* . 验证签名后获得用户名 *nameauthor* 和口令 *passwordauthor* . *find* 语句验证用户名和口令是否正确, 验证通过则允许授权. 之后事件 *event Authorization*(*nameauthor*) 发生, 表明授权服务器验证了终端用户的身份. 然后连接 *clientidone* 和 *urione* 为 *idurione* , 通过 *find* 语句检测 *idurione* 是否注册. 检测通过后生成一个授权码 *authorcode* , 由 *c*20 通道传递给请求客户端, 并将其与 *clientidone* 绑定为 *idcodeone* . 授权服务器在 *c*21 通道接收到消息 *fivemessage* 和签名 *fivesign* , 验证签名通过后得到参数: 授权许可类型 *granttypetwo* 、授权码 *codeforauthortwo* 、客户端标识符 *clientidtwo* 、客户端密钥 *client*sec*rettwo* 以及重定向 URI *uritwo* . 授权服务器检测 *granttypetwo* 的值是否为常量 *code_grant* , *find m <= N suchthat defind*(*authorcode*[*m*]) && (*authorcode*[*m*] = *codeforauthortwo*) *then* 语句检测授权码是否正确 *find q <= N suchthat defind*(*id*sec*retsuccess*[*q*]) && (*id*sec*retsuccess*[*q*] = *concatB*(*clienttwo*,*client*sec*rettwo*)) *then*' 验证 *clientidtwo* 和 *client*sec*rettwo* 是否正确与绑定, *find r <= N suchthat defind*(*idurione*[*r*]) && (*idurione*[*r*] = *concatC*(*clientidtwo*,*uritwo*)) *then* 验证 *clientidtwo* 与 *uritwo* 是否与获取授权码时的客户端标识符和重定向 URI 相同. *find s <= N suchthat defind*(*idcodeone*[*s*]) && (*idcodeone*[*s*] = *concatD*(*clientidtwo*,*codeforauthortwo*)) *then* 验证授权码 *codeforauthortwo* 与 *clientidtwo* 的绑定关系. 全部验证均通过则授权服务器生成访问令牌 *token* 和令牌说明 *tokentype* , 在 *c*22 通道将其发送给 *client* . 客户

端获得 *token* 与 *tokentype* 后，与资源服务器进行交互，取得受保护的资源.

```
let Authorization_server =
c14(urireg : redirect_uri); new clientid : client_id;
new client sec ret : client_sec ret;
let id sec retsuccess : message = concatB(clientid, client sec ret) in
let idurisuccess : message = concatC(clientid, urireg) in
c15 < A, id sec retsuccess >;
c16(= one, onemessage : message, onesign : signature);
if check(onemessage, pku, onesign) then
let concatA(namereg : username,
passwordreg : password) = onemessage in
c <>; c17(= web, twomessage : message, twosign : signature);
if check(twomessage, pkc, twosign) then
let concatE(responsetypeone : response_type, clientidone
: client_id, urione : redirect_uri) = twomessage in
if responsetypeone = code_response then
c18 < Authertication_message >;
c19(= three, threemessage : message, threesign : signature);
if check(threemessage, pku, threesign) then
let concatA(nameauthor : username,
passwordauthor : password) = threemessage in
find k <= N suchthat defined (onemessage[k])
&& (onemessage[k] = threemessage) then
find p <= N suchthat defined (namereg[p])
&& (namereg[p] = name_author) then
event Authorization(three, nameauthor);
let id_uri_one : message = concatC(clientidone, urione) in
find l <= N suchthat defind (idurisuccess[l])
&& (idurisuccess[l] = idurione) then
new authorcode : code;
let idcodeone : message = concatD(clientidone, authorcode) in
c20 < four, authorcode >;
c21(= five, fivemessage : message, fivesign : signature);
if check(fivemessage, pkc, fivesign) then
let concatF(granttypetwo : grant_type, codeforauthortwo
: code, clientidtwo : client_id, client sec rettwo : client_sec ret,
uritwo : redirect_uri) = fivemessage in
if granttypetwo = code_grant then
find m <= N suchthat defind(authorcode[m])
&& (authorcode[m] = codeforauthortwo) then
find q <= N suchthat defind(id sec retsuccess[q]) &&
(id sec retsuccess[q] = concatB(clienttwo, client sec rettwo)) then
find r <= N suchthat defind(idurione[r]) &&
(idurione[r] = concatC(clientidtwo, uritwo)) then
find s <= N suchthat defind(idcodeone[s]) &&
(idcodeone[s] = concatD(clientidtwo, codeforauthortwo)) then
new token : access_token; new tokentype : token_type;
c22 < token, tokentype > .
```

图 8　授权服务器进程

3　验证结果

使用自动化证明工具 Cryptoverif 进行分析，经过多次化简处理和观察等价，在 22 个 Game 序列转换后得到证明结果，如图 9 所示. 结果表明，在改进的 OAuth2.0 协议中，数字签名机制能够实现授权服务器对终端用户和授权服务器对客户端的身份认证.

图 9　证明结果

4　结论

为了增强 OAuth2.0 安全性，本文对 OAuth2.0 协议进行了改进，引入数字签名机制，实现授权服务器对终端用户和授权服务器对客户端的身份认证. 并在计算模型下基于 Blanchet 演算，对改进的 OAuth2.0 协议进行形式化，应用一致性对授权服务器认证资源拥有者和授权服务器认证客户端进行建模. 然后，把用 Blanchet 演算建模的 OAuth2.0 协议转换为自动化工具 CryptoVerif 的输入，最后使用 CryptoVerif 分析和证明了其认证性. 以后的工作计划给出改进的 OAuth2.0 协议的 Java 实现，并且对其安全性进行分析和验证.

参考文献

1 Hardt D. The OAuth 2.0 Authorization Framework. IETF RFC 6749. http://tools.ietf.org/html/rfc6749.

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

3 Corella F, Karen P. Lewison. security analysis of double

(下转第 39 页)

参考文献

1 Datcu M, Seidel K, Image information mining: exploration of image content in large archives. Aerospace Conference Proc. 2000, 3. 253–264.

2 Dai Q, Liu JB, Liu SB, Ma CH. The research on intelligent content-based remote sensing image retrieval with multi-features. The 17th IASTED IASTED International Conference on Communication, Internet, and Information Technology. CIIT 2012, Baltimore, MD, United states. May. 2012. 14–16.

3 戴芹,刘建波,刘士彬.综合多特征遥感图像智能检索方法的概念设计.地球信息科学学报,2011,3:16.

4 曾月,范玉顺.工作流管理系统 Web 客户端的设计与实现.计算机工程与应用,2002,38(2):130–133.

5 戴浩.JBPM 工作流管理系统在 OA 中的应用.电脑学习,2011,2:130–1315.

6 赵耀,袁梅宇,夏文财.基于 Flex 的 jBPM Web 流程设计器的研究与设计.贵州大学学报(自然科学版),2013,29(2):107–110.

7 曾炜,阎保平.工作流模型研究综述.计算机应用研究,2005,22(5):11–13.

8 van der Aalst WMP. The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers, 1998, 8(01): 21–66.

9 Medina-Mora R, Winograd T, Flores R, et al. The action workflow approach to workflow management technology. Proc. of the 1992 ACM conference on Computer Supported Cooperative Work. ACM. 1992. 281–288.

10 JBPM team. JBPM User Guide. http://docs.jboss. org/jbpm/v5.4/userguide/ 2013.3.

11 傅明,张玮.基于 J2EE 开源工作流引擎 JBPM 的设计实现.计算技术与自动化,2008,27(4):111–114.

12 谈丽.基于 JBPM+FLEX 的 Ring Cloud2.0 系统的研究与实现[学位论文].北京:北京交通大学,2010.6

13 王宇明,庄继晖.JBPM-一个开源的 J2EE 工作流管理系统.微处理机,2006,10(5):113–115.

14 顾文轩,王琼,徐汀荣.基于 JBPM 的工作流管理系统的研究与设计.计算机应用与软件,2009,26(5):104–106.

15 赵瑞东,陆晶,时燕.工作流与工作流管理技术综述.科技信息,2007,8:105–107.

16 van der Aalst W, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. IEEE Trans. on Knowledge and Data Engineering, 2004, 16(9): 1128– 1142.

17 Wohed P, Russell N, Ter Hofstede AHM, et al. Patterns-based evaluation of open source BPM systems: The cases of JBPM, OpenWFE, and Enhydra Shark. Information and Software Technology, 2009, 51(8): 1187–1216.

18 Recker JC, Indulska M, Rosemann M, et al. How good is BPMN really? Insights from theory and practice. 2006.

19 Sindre G. An analytical evaluation of BPMN using a semiotic quality framework. Advanced topics in database research, 2006, 5: 94.

20 李玺,胡志刚,胡周君等.基于截止时间满意度的网格工作流调度算法.计算机研究与发展,2011,48(5):877–884.

21 胡春华,吴敏,刘国平等.一种基于业务生成图的 Web 服务工作流构造方法.软件学报,2007,18(8):1870–1882.

(上接第 30 页)

redirection protocols. 2011. http://pomcor.com/techreports/DoubleRedirection.pdf.

4 时子庆,刘金兰,谭晓华.基于 OAuth2.0 的认证授权技术.计算机系统应用,2012,21(3):260–264.

5 Xu XD, Niu LY, Meng B. Automatic verification of security properties of OAuth2.0 protocol with cryptoverif in computational model. Information Technology Journal, 2013, (12): 2273–2285.

6 薛锐,雷新锋.安全协议:信息安全保障的灵魂-安全协议分析研究现状与发展趋势.中国科学院院刊,2011,26(3):287–296.

7 邵飞.基于概率进程演算的安全协议自动化分析技术研究[学位论文].武汉:中南民族大学,2011.

8 Dolev D, Yao A. On the security of public key protocols. IEEE Trans. on Information Theory, 1983, 29(2): 198–208.

9 朱玉娜.密码协议符号分析方法的计算可靠性研究[学位论文].郑州:解放军信息工程大学,2008.

10 郑清雄.基于 Spi 演算的安全协议验证.计算机应用与软件,2011,28(3):262–264,292.

11 Blanchet B. A computationally sound mechanized prover for security protocols. IEEE Symposium on Security and Privacy. 2006. 140–154.