基于 TOTP 的 Web 改进认证①

赵建勋

(西安文理学院 数学与计算机工程学院, 西安 710065)

摘 要: 为了提高 TOTP 协议应用在 Web 认证中的安全性, 依照 HOTP 认证三原则改进了基于 TOTP 的认证设计. 改进后的认证系统在一个时间窗口内增加了一个认证次数阈值和时间戳用来更好的抗蛮力攻击和重放攻击, 增加随机数和 MD5 哈希算法轻量化地抵抗中间人攻击. 最后用 PHP 语言设计了一个安全、实用的 Web 认证系统.

关键词: HOTP; TOTP; 一次性口令; Web 认证; 攻击

Improved Web Authentication Based on TOTP

ZHAO Jian-Xun

(School of Mathematics and Computer Engineering, Xi'an University, Xi'an 710065, China)

Abstract: The paper makes an improved authentication method order by Three-Protocol of HOTP authentication method based on TOTP. The authentication method uses an authentication number threshold and a timestamp to resist brute force attacks and replay attacks, uses a random number and the MD5 encryption resist Man-in-the-Middle attack. Finally, a safe and useful Web authentication protocol is designed by PHP .

Key words: HOTP; TOTP; One-time-password; Web authentication; attacks

传统的 Web 认证下用户的用户名和口令是确定不变的, 这种方式最大的缺陷就是一旦用户名和口令的信息被非法窃取, 就可能导致重放攻击. 一次性口令(One-Time Password)系统下每次登录根据不确定因素产生一个新的口令从而避免了这种重放攻击, 增强了认证的安全性. HOTP[1](HMAC-Based One-Time Password)和 TOTP[2](Time-Based One-Time Password)是 OATH 组织成员分别基于事件和时间因子的两种一次性口令算法标准的认证算法, 其中 TOTP 可以看作是 HOTP 的一种扩展. 基于时间因子的 TOTP 提供了一个具有短暂时效性的一次性口令, 这是现代安全认证协议所需要的, 但是仍然还有许多安全隐患. 本文对 TOTP 认证协议进行了分析和改进, 并在 Web 上进行了设计, 实现在用户界面的形式和使用习惯方面保持了原有风格的同时, 又加固了认证的安全性.

1 TOTP算法简述

HOTP 算法是基于一个加法计数器 C 和一个静态的对称密钥 K, 该密钥 K 仅有客户端令牌和认证服务器知道. 使用 HMAC-SHA-1[3]算法得到 HOTP 值, 由于 HMAC-SHA-1 计算输出为 160 比特, 使用截短函数 Truncate 以便于用户输入, 总的函数可以表述为 HOTP(K,C)[1] = Truncate(HMAC-SHA-1(K,C), N), 其中 K 是共享的密钥, C 是加法计数器, N 是截短为十进制的位数.

TOTP 是基于时间的算法, 其中一个代表时间基准和时间步长关系的一个整数 T, 取代 HOTP 加法计数器 C. 即 TOTP(K,T)=Truncate(HMAC-SHA-1(K,T), N).

1.1 TOTP 值的生成

由 HOTP 值的生成过程得出 TOTP 值的生成大体上分为以下三步.

第一步: 确立时间整数 T = (Current Unix time - T0) / X, 其中 Current Unix time 就是当前 Unix 时间, T0 代表 Unix 的时钟开始计算的时间, 默认值是 0, X 代表一个时间的步长, 默认值为 30 秒. 例如, 如果当前 Unix 时间是 29 秒, T0= 0 和时间步长 X = 30, 则 T = 0; 如

① 收稿时间:2015-01-15;收到修改稿时间:2015-03-04

果当前的 Unix 时间是 30 秒则 T =1.

第二步: 生成一个 HMAC-SHA-1 值, 令 HS = HMAC-SHA-1(K,T),其中 HS 是一个 160 位的二进制数.

第三步: 对 HMAC-SHA-1 截短. 首先动态截短生成一个 31 位的二进制数,即 Sbits = DT(HS); 其次将 Sbits 转换为 10 进制数, 即 Snum = StToNum(Sbits); 最后返回将 Snum 转为所要的 N 位十进制数字, 即 TOTP = Snum mod 10^N.

1.2 传统的 TOTP 认证

一个安全的 OTP 系统不仅在于 OTP 值的生成, 而且在认证协议合理设计方面也是非常重要的[4]. 传统 TOTP 的认证过程如图 1 所示, 步骤如下:

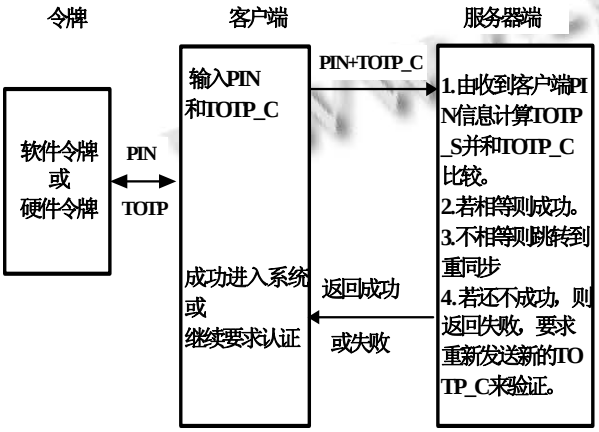

图 1 传统 TOTP 认证协议交互过程

①当客户端要在认证自己的身份时, 会把自己的个人识别码(PIN)和令牌上的 TOTP_C 值(双因子)发给服务端,服务器端根据 PIN 找到用户共享的 K 计算出 TOTP_S 值, 然后 TOTP_C 和 TOTP_S 相互比较.

②若 TOTP_C 和 TOTP_S 相等, 则验证成功.

③若 TOTP_C 和 TOTP_S 不相等, 跳转至重同步机制继续将 TOTP_C 和 TOTP_S 比较. 在重同步机制里, TOTP_C 分别与 TOTP_S 前 W 个和后 W 个值进行比较, 若相等则验证成功, 此时服务器端记录下当前针对用户的验证时间的飘移值 D, 以后此用户再认证的时候, 则新的 TOTP_S=TOTP(K,T+D)达到与令牌同步认证的目的.

④若在重同步机制下还没有验证成功, 则验证失败, 要求重新发一个用户的令牌值 TOTP_C 来认证.

对 TOTP 认证协议来说, 理论上只要客户机和服务器端时间相同, 则 TOTP_C 和 TOTP_S 相等. 但是在实际过程中由于网络的延迟, TOTP_C 和 TOTP_S 生成时刻不同, 若两者恰好在一个时间步长窗口之内, 则 TOTP_C 和 TOTP_S 相等; 若落在不同的时间步长窗口, 则可能造成验证失败.

2 漏洞分析及改进

2.1 与 HOTP 认证要求相比较

TOTP 是 HOTP 的一个扩展, 可以根据 HOTP 的三个安全认证要求[1]来分析 TOTP 并改进. 首先, TOTP 既有“所知”: PIN, 又有“所得”: 令牌, 达到了双因子认证[5]的要求; 其次, TOTP 并没有对客户端验证失败后重新发送令牌值的次数有一个限制, 使得缺少一个锁定机制来抵抗蛮力攻击; 最后, 无法避免重放攻击, 若在用户验证成功的时候被窃取到 PIN 和 TOTP_C 值, 在其时间步长范围内就可能被非法认证.

2.2 中间人攻击

由于此认证协议是单向的, 即只能服务器验证客户端, 而不能客户端验证服务器端, 这样使得攻击者作为中间人可以冒充服务器端来达到会话劫持, 进而对客户信息窃取、篡改等, 对服务器身份进行认证是非常必要的. 而此服务端接受认证是被动的, 客户一旦发现服务端非法就停止连接, 所以设计认证过程可以比较轻量化[6].

2.3 改进

由以上分析可得出三种最主要的漏洞, 本文进行了相应的改进.

2.3.1 蛮力攻击

设计一个阈值 T 来限制失败后的重复认证次数. 对每个用户的数据库表里加上一个验证失败次数值 ErrCount 的字段, 验证失败后 ErrCount 自加 1, 若值大于等于 T 的时候锁定该用户, 不允许再次登录直到服务端手动解除. 而当验证成功后 ErrCount 重置为 0.

2.3.2 重放攻击

设计一个对每一个成功验证用户的 TOTP 值都确立的时间戳 LastIn, 在同一个时间步长范围内若有重放攻击, 即被窃取 TOTP 值会重新发送到服务端再生成一个时间戳, 这时候服务端会比较这两个时间戳, 一旦发现两者相同, 则被认定是重放攻击, 则返回失败.

2.3.3 中间人攻击

对双因子的“所知”由 Uid 和 PIN 两个组成, 作为公钥被服务器和客户端共享. 首先客户端发送哈希函数值 MD5(Uid)和一个随机数 Rnum 给服务端; 其次服务器验证 MD5(Uid), 并计算 MD5(PIN+Rnum)发给客户端(+表示串联); 客户端也计算 MD5(PIN+Rnum), 若相等则验证了服务器, 则再继续进行客户端的认证. 这样在信道上传输的都是哈希函数 MD5, 即使被中间人获取到, 其也无法直接破解到真值. 另外, 在一个在安全信道上(例如 SSL/TLS, IPSec 链路)进行认证能更好的抵抗中间人攻击, 本文暂不研究[7].

3 改进的TOTP认证设计

相对于传统的 C/S 架构, OTP 系统认证往往需要一个客户端程序来专门运算加密、解密等, 而现在 Web 应用大多数都是 B/S 架构, 用户只需要一个浏览器就可以代替. 本文对客户端的 TOTP 值的本地计算可以用客户端脚本语言 JavaScript 实现, 这个程序是嵌入在 PHP 页面中的. 服务器端的 TOTP 值可以用服务器脚本语言 PHP 实现[8].

3.1 相关符号和数据

3.1.1 服务端用户的信息数据表

一个用户信息表在服务端初始化可如表 1.

表 1　一个 Uid 和 PIN 分别是“job”和“123”的用户数据表初始化值

Uid	PIN	LastIn	DriftTime	Dtiming	ErrCount	LockFlag
job	123	0	0	-W	0	0

Uid: 用户名, 客户端和服务端的共享密钥; PIN: 个人认证码, 客户端和服务端的共享密钥; DriftTime: 上一次认证成功的时间漂移值; Dtiming:是测试的漂移值; LastIn:最后成功验证的时间函数值; ErrCount: 失败验证的次数; LockFlag: 用户锁定标志位.

3.1.2 TOTP 值

TOTP_C: 客户端 TOTP 值, TOTP_C = TOTP(PIN,Tc), 对于 Tc= d.getTime()/X/1000, 其中 d 是 JavaScript 语言里时间 Date 对象的一个实例, getTime()返回的是当前从 Unix 纪元到当前时间的毫秒数, X 为默认的 30 秒钟的时间步长.

TOTP_S: 服务器端 TOTP 值, TOTP_S = TOTP(PIN,Ts). 其中 Ts= time ()/X+ DriftTime , 其中 time()是 PHP 语言里返回当前从 Unix 纪元到当前时间的秒数; DriftTime 是一个时间步长范围(取值为[-W,W])的漂移值, 用于服务端和客户端的 TOTP 值重同步.

3.2 TOTP 认证过程设计

①用户访问服务端的认证页面, 输入自己的计算好的 MD5(Uid)和一个随机数 Rnum, 传输给服务端.

②服务端对表里的 Uid 进行计算并查询找到其对应的 PIN 值, 并计算 MD5(Uid+Rnum)发送给客户端.

③客户端计算并验证 MD5(Uid+Rnum), 若验证不成功则停止对此服务端的认证, 并返回错误代码; 若成功则客户端发送自己的 TOTP_C 值给服务端.

④服务端首先验证此 PIN 的用户的 LockFlag 位是否被锁定, 若锁定则返回认证失败类型; 若没有锁定则将 TOTP_C 与 TOTP_S 进行比较.

⑤若 TOTP_C 与 TOTP_S 相等, 则比较 Ts 与 LastIn, 若相等则判定为重放攻击, 直接返回认证失败; 否则更新 LastIn, 重置错误计数器 ErrCount, 记录下时间漂移值 DriftTime

⑥若 TOTP_C 与 TOTP_S 不相等, 则对 Ts 从前 W 个到后 W 个值修正比较 TOTP_C, 若有相等的则⑤; 若最终还是不等则 ErrCount 自加 1 并返回认证失败.

⑦该用户下一次认证的到来, 若 ErrCount 大于阈值 T, 则锁定该用户不得登录.

一个用户的整个过程如图 2 所示.

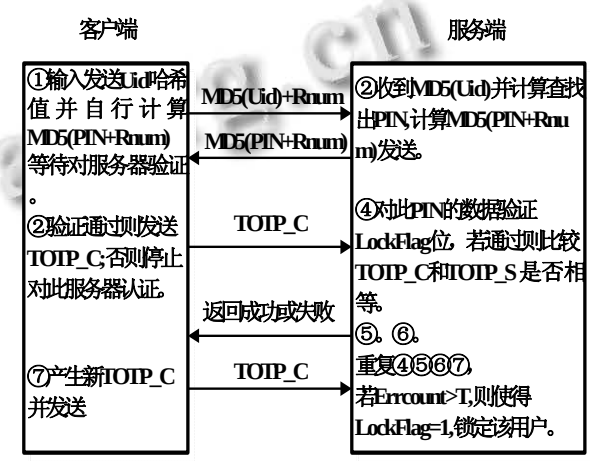

图 2　改进的基于 TOTP 的 Web 认证过程

3.3 核心部分的实现测试

3.3.1 TOTP_S 和 TOTP_C 的 Web 生成

本文基于 Web 的 TOTP 值成功用 PHP 和 JavaScript 两种脚本在服务端和客户端生成实现, 如图 3 所示, 在共享密钥为“NIHAOMA”, 时间戳相差在一个时间

窗口内(30 秒), TOTP_S 和 TOTP_C 相等.

图 3 分别用 PHP 和 JavaScript 生成的 TOTP 值

3.3.2 Web 认证过程的实现

本部分由于篇幅所限, 用类似 C 语言的伪代码表示并加上注释, 以下是认证核心算法部分, 即从图 3 的第④步的开始实现.

```
session_start();
//页面开始时, 对当前要认证的用户 session 启动
------ //对验证 MD5(PIN+Rnum)过程略去
if (LockFlag==0){    //若用户没有被锁定
for(int j= DriftTime;j<=2W+DriftTime;j++){
//在漂移值基础上 2W 大小的窗口内重同步
if (TOTP_C==TOTP_S){
//比较客户和服务端 TOTP 值
if ( Ts == LastIn){
//新认证时间和最后一次成功的认证时间若相同
LockFlag=1;          //确定为重放攻击, 锁定用户
return 0;}           //立即返回失败处理
      else{
//确保是新 TOTP_C 值的因子 Ts, 排除重放攻击
LastIn= Ts;//   更新最后验证成功的时间因子值
ErrCount=0;   //错误次数值置 0
DriftTime=Dtiming+W;  //记录漂移值
return 1;  }}//在 2W 的窗口内重同步后成功验证
else  {Dtiming =j-W+1;
//测试在[-W,W]内从左边开始的漂移值
ErrCount++; } }          //在 2W 窗口内继续比较
if (LastIn==0)  {ErrCount++;
//在 2W 窗口内重同步后失败,验证失败值加 1
if (ErrCount>T)
//重同步失败次数超过阈值 T, 确定是蛮力攻击
LockFlag=1;   //锁定用户
return 0; }}      //直接返回失败处理
```

4 总结

TOTP 生成算法是 HOTP 的一个扩展, 其用时间的递增的随机性代替了计数器的规律的自加行为, 对入侵者来说增强了不可预测性. TOTP 的认证相比 HOTP 并没有增加安全性, 改进后 TOTP 认证算法使得认证过程不仅考虑了事件因素, 也增加了时间的因素, 能够抗重放攻击, 中间人攻击和蛮力攻击. 笔者应用改进的 TOTP 认证算法在 PHP 中实现, 证明了能较好的应用在 Web 认证中, 但是对于用户注册阶段的安全认证, 还有待于进一步研究.

参考文献

1 M'Raihi D, Bellare M, et al. HOTP: An HMAC-Based One-Time Password Algorithm. http://www.apps.ietf.org/rfc/rfc4226.html. 2005-11.

2 M'Raihi D, Machani S, Pei M, et al. TOTP: Time-based one-time password algorithm. http://tools. ietf.org/html/ rfc6238. 2011-8.

3 Krawczyk H, Bellare M, Canetti R. HMAC: Keyed-Hashing for Message Authentication. http://www.ietf. org/rfc/rfc 2104. txt.

4 刘天法,冯启源,杨桂勇.基于一次性口令的认证技术的研究.计算机应用与软件,2007,(8):177–179,206.

5 Birch DGW. Digital identity management. London: Gower Publishing, Ltd. 2007–5: 113–118.

6 杨明星,彭新光.一种轻量级一次性口令认证方案.小型微型计算机系统,2014,(8):1808–1811.

7 孙子谦,王雅琴.基于时间一次性口令的 Linux 登录认证的研究与实现.计算机应用,2014,(S1):55–56,69.

8 方俊.一种 B/S 模式一次性口令系统.计算机系统应用.2012,21(9):124–127,78.