УДК 004.056.5

Гончар С.А.[1], аспірант

Використання “bitcoin” - схеми для безпарольної авторизації на сервері.

[1] Київський національний університет імені Тараса Шевченка,03680, м. Київ, пр-т Глушкова 4д, e-mail: sg.gonchar@gmail.com

S. A. Gonchar [1], postgraduate

Using "bitcoin" - scheme for server authorization without password.

[1] Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova av., 4d, e-mail: sg.gonchar@gmail.com

У даній статті представлено новий підхід для безпарольного доступу на сервер, використовуючи “bitcoin” - схему. Проаналізовано переваги та недоліки запропонованого підходу в порівнянні з іншими підходами.

Ключові слова: криптографія, bitcoin, клієнт-сервер.

This paper presents new approach for client-server systems authentication without password using "bitcoin"- scheme. Presents list of major contemporary approaches used for client-server systems authentication, given their description and some principles of using them in practice. Shown their advantages and disadvantages in specific systems. Described basic rules of the proposed approach, based on using "bitcoin" - scheme for server authorization without password, and differences in comparison with the classic "bitcoin"- scheme. Shown main schemes of using this approach when we create transactions for both classical (in time when client transfer money) and modified (in time when client authentication on the server) approaches. Presented modified scheme for constructing blocks and transactions in current chain in accordance with the client-server structure. Analyzed advantages and disadvantages of the proposed approach. Held comparison with other approaches, which are widely used in modern client-server systems to authenticate clients in some resource.

Key Words: cryptography, bitcoin, client-server.

Статтю представив чл.-кор. НАНУ, д.ф.-м.н., проф. Анісімов А.В.

Вступ

У даний час клієнт-серверна архітектура набуває все більш широкого розповсюдження, оскільки в більшості випадків користувачі намагаються отримати певну інформацію централізовано, використовуючи сервер, на якому вона зберігається.

Більшість таких ресурсів є захищеними, оскільки велика кількість інформації на них є конфіденційною і вона повинна оборонятись від зловмисників. В основному доступ до цієї інформації на серверах відбувається за допомогою авторизації користувачів.

Зараз на практиці використовують безліч підходів, які позволяють користувачам проходити авторизацію на сервері і отримувати доступ до певної інформації у відповідності до свого аккаунта. Самим простим засобом авторизації на даний момент є використання логіна та пароля, створених в процесі реєстрації на певному ресурсі. Саме логін та пароль позволяє ідентифікувати користувача і надати йому доступ до інформації.

Окрім логіна та пароля можуть використовуватись також інші підходи, які передбачають використання ресурсів комп’ютера користувача та його унікальні адреси в мережі.

Усі ці підходи є ефективними, але не дуже захищеними. Кожен із цих параметрів авторизації може бути підібраним зловмисником і він може получити доступ до конфіденційної інформації. Саме тому проблема авторизації на клієнт-серверних архітектурах є актуальною і потребує нових підходів, які дадуть змогу максимально знизити імовірність підбору параметрів авторизації, тим самим захистивши цю інформацію від зловмисників.

Існуючі підходи

У сучасних клієнт-серверних архітектурах найбільш поширеними є наступні засоби авторизації на серверах [1] :

1. *Проста авторизація* – можлива тільки після реєстрації на ресурсі, а для здійснення самої авторизації потрібна пара логін-пароль, які надаються користувачеві після реєстрації. Перевагою даного підходу є його простота, а от недоліком є те, що для різних ресурсів можуть бути різні облікові дані, які потрібно запам’ятовувати, але не завжди це вдається.
2. *Авторизація по OpenID* – метод

авторизації, для використання якого потрібна реєстрація у так званого «провайдера ідентифікації» і «залежна сторона» - кінцевий ресурс, який намагається ідентифікувати користувача. Особливістю цього підходу є те, що реєстрація на самому ресурсі не є обов'язковою, а провайдер ідентифікації може бути для різних ресурсів. Перевагою в підході є використання однієї пари логін-пароль для багатьох ресурсів, а недоліком – цей підхід ще мало розповсюджений і дуже мало ресурсів його використовують.

3. *Епит-авторизація* – прив'язка «аккаунта» до мобільного телефону, номеру. При використанні даного підходу на телефон відправляється смс-повідомлення з унікальним кодом, який потім використовується для авторизації на ресурсі. Перевагою даного підходу є те, що дана авторизація в порівнянні з іншими є найбільш захищеною, оскільки номери телефонів є унікальними і отримати код зможе лише користувач, якому належить цей номер, а недоліком в даному підході є те, що він мало розповсюджений. Також він може обмежувати доступ, якщо телефона немає поряд, або ж телефон може бути украденим, в такому випадку доступ до ресурсу може получити зловмисник.

4. *Авторизація з використанням унікальних параметрів комп'ютера* [2] – даний підхід дає можливість авторизувати користувача, використовуючи його IP-адрес, MAC-адрес, Windows Login і т.п., або їх комбінації. Використовуючи даний підхід, користувач має змогу получити доступ до ресурсу, не використовуючи пари логін-пароль або іншої подібної авторизації. Перевагою даного підходу є те, що він позволяє користувачу не приймати безпосередню участь в процесі авторизації, оскільки використовуючи дані комп'ютера вона пройде автоматично, а недоліком в даному підході є те, що кожна зміна комп'ютера, або інших ресурсів, призведе до неможливості авторизуватись на ресурсі.

Усі описані вище засоби авторизації, які зараз широко використовуються, мають ті чи інші проблеми. Для кінцевого користувача не буде дуже ефективним використання пари логін-пароль кожен раз, коли він захоче відвідати певний ресурс. Прив'язка до телефону також заставлятиме користувача виконувати певні дії, щоб авторизуватись. І не варто забувати про те, що на практиці кожен із цих засобів авторизації може бути взломаним підбором зловмисником.

Основні принципи роботи "bitcoin"-схеми.

Розглянемо основні принципи роботи "bitcoin"-схеми.

Bitcoin – це система електронних грошей, яка працює за принципом pear-to-pear [3]. У bitcoin немає центру управління, система працює автономно і підпорядковується відомим алгоритмам, а інформація про транзакції зберігається у всіх учасників системи.

В bitcoin вся інформація про транзакції зберігається в ланцюгу блоків [4]. Блоки передаються в форматі JSON. Кожен блок містить заголовки і список транзакцій. Заголовок складається з декількох властивостей, серед яких є також хеш попереднього блоку. Таким чином, весь ланцюг блоків зберігає всі транзакції за весь час роботи bitcoin.

В останніх версіях системи весь ланцюг блоків зберігається повністю кожним клієнтом, що робить цю систему децентралізованою.

На малюнку 1 показано принцип роботи "bitcoin"-схеми в момент створення нової транзакції.

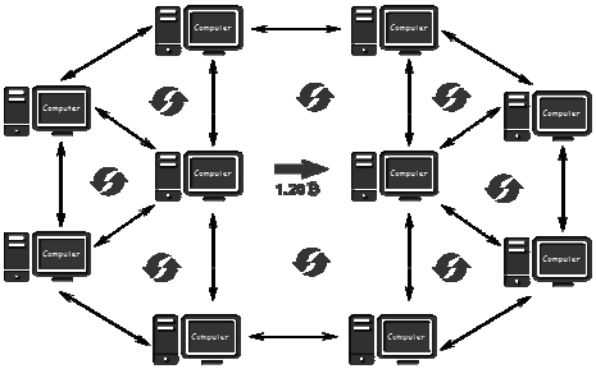

Рис. 1 Принцип роботи "bitcoin"-схеми в момент транзакції.

Учасники діляться на 2 групи: на тих, хто працює над новим блоком і тих які не працюють. Один із клієнтів створює нову транзакцію і виконує її розсилку іншим клієнтам, які займаються генерацією блоку. Вони добавляють цю транзакцію до свого блоку і продовжують генерацію. Коли хтось згенерує такий блок, він запечатується і розсилається по мережі. Дальше клієнти провіряють блок і дані всередині нього на валідність. Якщо ніяких проблем немає, то транзакції вважають виконаними і цей блок зберігається у кожного з клієнтів.

Використання "bitcoin"-схеми для безпарольної авторизації на сервері.

Пропонується використовувати "bitcoin"-схему для безпарольного доступу до певного ресурсу (серверу). Оскільки дана схема працює лише з клієнтами, а ми хочемо використати її для клієнт-серверної системи, модифікуємо її відповідним чином. В даному випадку в якості

певних клієнтів виступатимуть сервера. Розглянемо, як зміниться сама схема при авторизації клієнта на сервері :

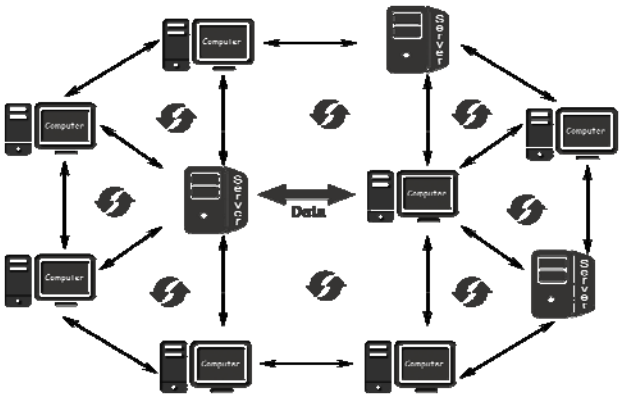

Рис. 2 Принцип роботи “bitcoin”-схеми в момент авторизації клієнта на сервері.

Як видно на малюнку 2, в даній модифікованій схемі у нас в транзакції буде передаватись певна інформація, розшифрувати яку зможе лише клієнт або сервер.

Як відомо, “bitcoin”-схеми дають змогу коректно ідентифікувати вірні транзакції, основуючись на певному ланцюгу зв’язків. Це відбувається, коли іде обмін даними між певними учасниками і створюється транзакція. Інші учасники, маючи коректну інформацію про учасників які створили транзакцію, перевіряють їх валідність в системі, і якщо валідність підтверджується, транзакція підтверджується. Даний ланцюг зв’язків можна використати саме для безпарольної авторизації клієнтів на серверах, які знаходяться в цьому ланцюгу. Сервер, авторизуючи клієнта, перевірить, чи дійсно клієнт має з ним зв'язок, використавши інших учасників у цьому ланцюжку, у яких буде достовірна інформація про те, чи дійсно цей клієнт має зв'язок з цим сервером. Якщо валідність учасника буде підтверджена, йому буде наданий доступ до ресурсу.

Розглянемо з яких елементів буде складатись сам блок більш детально. Усі наступні викладення базуються на стандартних параметрах “bitcoin”-схеми з модифікацією певних параметрів у відповідності до клієнт-серверного підходу.

- *hash* – SHA-256 хеш заголовка блоку. Такий хеш являється досить випадковим, а тому час його обрахування є визначеним.

- *ver* – версія схеми блоку.

- *prev_block* – хеш попереднього блоку в ланцюзі. Саме завдяки цій властивості ланцюг неможливо підробити, замінивши в ній один із блоків, оскільки хеш блоку завжди залежить від хеша попереднього блоку в ланцюзі. Якщо змінити один із блоків, прийдеться наново створювати всі наступні.

- *mrkl_root* – список хешів транзакцій. Хеш блоку повинен обов’язково залежати від транзакцій, щоб їх неможливо було підробити. Але обраховувати його напряму буде дуже довго, якщо кількість транзакцій велика. Тому спочатку хешуються самі транзакції, а потім їх хеші використовуються для обрахування хешу всього блоку.

- *time* – відображає час створення блоку.

- *bits* – одна із самих важливих властивостей. Являється скороченою формою цільового значення хеша. Блок вважається згенерований, коли його хеш менший цього цільового значення.

- *nonce* – число, яке, починаючи з нуля, інкрементується після кожної ітерації обрахування хеша. Так і проходить перебір, поки хеш не буде меншим цільового значення.

- *n_tx* – кількість транзакцій в списку.

- *size* – розмір блоку в байтах.

Розглянемо більш детально з чого складаються самі транзакції. Транзакції містяться в блоках у вигляді списку. Вони, так само як і блоки, вистроюються в ланцюги. Кожна транзакція повинна указувати, звідки вона братиме певні дані. Для задання адресата використовуватиметься його публічний ключ. Щоб адресат міг використовувати отримані дані, він повинен створити нову транзакцію, яка буде брати ці дані з попередньої і перенаправляти їх на інший адрес. Для того щоб перевіряти що користувач використовує для авторизації саме свої особисті дані, а не чужі, він повинен залишити в своїй транзакції свій цифровий підпис. Тоді в будь який момент часу можна впевнитись, що всі транзакції являються валідними. На практиці це може бути реалізовано за допомогою наступних властивостей:

- *hash* – хеш всієї транзакції. Транзакції хешуються 2 рази. Перший раз під час обрахування хеша транзакції. Другий раз – під час обрахування хеша блоку. Окрім того, кожний блок посилається на хеш попереднього блоку, а кожна транзакція – на хеш попередньої транзакції. Тому якщо поміняти транзакцію і якимсь чином її хеш не поламається, то зламаються всі інші хеші і змінений ланцюжок блоків буде відхилений іншими користувачами.

- *ver* – версія схеми транзакції.

- *vin_sz* – кількість попередніх транзакцій, із яких дані переводяться на нові адреси.

- *vout_sz* – кількість адрес, на які

відправляються дані.

- *size* – розмір транзакції в байтах.

- *in* – містить список входів транзакції. В якості входів використовуються попередні транзакції (*prev_out*).

У кожного виходу в свою чергу є наступні властивості:

- *hash* – хеш попередньої транзакції.

- *n* – так як у транзакції може бути декілька виходів, потрібно указувати з якого саме беруться дані. Для цього випадку якраз і існує ця властивість. В ньому міститься порядковий номер виходу попередньої транзакції, починаючи з 0.

- *scriptSig* – у цій властивості відправник повинен доказати, що він переводить дійсно свої дані, а не чужі. Для цього він указує публічний ключ отримувача попередньої транзакції, тобто свій ключ, так як він повинен бути отримувачем. Окрім того від добавляє ECDSA підпис цієї ж транзакції, яка створена його приватним ключем. Це доказує, що він розпоряджається своїми даними, а не чужими.

- *data* – містить дані, які будуть передані на новий адрес.

- *scriptPubKey* – ця властивість, разом із *scriptSig* складають ключ, за допомогою якого іде перевірка на валідність. Сценарій перевіряє транзакцію на валідність. Використання подібного сценарію дає великі можливості для опису умов отримання даних адресатом. А тому в якості верифікації можуть служити різні паролі або перевірки, які будуть доступні лише певним користувачам.

Переваги та недоліки даного підходу

Основною перевагою запропонованого підходу є те, що він дасть нам змогу авторизуватись на певних ресурсах не використовуючи облікових даних. Усе що буде потрібно для авторизації, це буде лише список усіх блоків і транзакцій, за допомогою яких працює “bitcoin”-схема. У порівнянні з існуючими підходами, даний підхід дасть змогу мінімізувати можливість взлому ресурсів противником, використовуючи облікові дані клієнта, оскільки в “bitcoin”-схемі авторизація буде проходити лише в тому випадку, коли інші користувачі будуть підтверджувати валідність.

Основним недоліком даного підходу є те, що нам прийдеться зберігати весь час ланцюг блоків на комп’ютерах або інших ресурсах, звідки буде вестись комунікація з сервером. Також можуть бути невеликі затримки при перевірці валідності іншими учасниками, але дану проблему можна вирішити, зменшивши довжину хеша блоку, оскільки в клієнт-серверній структурі немає смислу використовувати великі хеші через те що тут не відбуватимуться маніпуляції з грошима, а буде відбуватись лише обмін інформацією.

Висновки

Даний підхід дасть змогу розширити “bitcoin”-схему на клієнт-серверні структури, при цьому сервер не потребуватиме у клієнта обов’язкової авторизації і сама авторизація в цьому випадку проходитиме автоматично. Це дасть змогу значно розширити клієнт-серверний обмін даними. При цьому весь цей обмін інформації буде захищеним і його неможливо буде підробити, оскільки робота в мережі bitcoin основана на принципі, результату роботи, якого тяжко добитись, але легко перевірити.

Список використаних джерел

1. *Ломалкин О.* Авторизация в веб: какой она может быть? [Електронний ресурс] / 2008 – Режим доступу до ресурсу : http://habrahabr.ru/post/28443
2. Типы авторизации пользователей [Електронний ресурс] / 2007 – Режим доступу до ресурсу : https://usergate.ru/support/axel/authorization_types.html
3. Bitcoin [Електронний ресурс] / 2010 – Режим доступу до ресурсу : http://bitcoinme.ru
4. *Сигов А.* Bitcoin. Как это работает [Електронний ресурс] / 2011 – Режим доступу до ресурсу : http://habrahabr.ru/post/114642

References

1. LOMALKIN, O., (2008) *Avtorizatsiya v web: kakoy ona mozhet byit?* [Online] Avaliable from: http://habrahabr.ru/post/28443 [Accessed: 20th April 2014].
2. Usergate. (2007) *Tipy avtorizatsii polzovateley* [Online] Avaliable from: https://usergate.ru/support/axel/authorization_types.html [Accessed: 20th April 2014].
3. Bitcoinme. (2010) *Bitcoin* [Online] Avaliable from http://bitcoinme.ru [Accessed: 20th April 2014].
4. SIGOV, A., (2011) *Bitcoin. Kak eto rabotaet* [Online] Avaliable from http://habrahabr.ru/post/114642 [Accessed: 20th April 2014].

Надійшла до редколегії 22.04.14