SAKUA2 – Shared Access Key User Authentication 2

anoop.kumar.narayanan@gmail.com

Version 1.1.2

Public key encryption replacement algorithm

Preface

The idea behind SAKUA2 encrypted connection is to fully thwart Man in the middle attack and provide a substitute for Public key encryption algorithm. The key idea behind this is to use shared information that both the service provider and client knows to setup and create a shared key and then use that to login to the service provider.

Domino password in this context is a term used to identify the technology of using pin, a random crypto string and pattern to generate a dynamic and strong password, which could also serve as a shared key for an encrypted connection. The user pin (known only to the client and the service provider) is used along with the random crypto string to generate a new random crypto string. This is accomplished not by using mathematical functions but instead by making use of shuffling, by swapping and rotating the string with the help of random numbers. The resultant string is pretty strong, contains all the characters that were there in the initial string that was received but cannot be recreated by a third party. The pattern is then used as indices to select specific characters from the random crypto secure string. This is the base of SAKUA2 encrypted connections.

SAKUA2 : Shared Access-Key User Authentication 2

Registration of Information

- Registration to a website is done via a Normal HTTPS session.
- DominoPassword Pattern and Pin.
- Some shared information that the site may request like some social account names, Secondary phone, Secondary email, fax, social accounts, DOB, some interesting websites names or emails that you would like to have or some crazy name that you think your friends should have, gps info, location information, account id, account branch location, shared cryptic string/phrase …etc

Authentication and Authorization

1. User visits website let's say a banking website.
2. Website requests login info
    1. User enters: loginname
3. Website based on the login information presents shared information that is known by both the service provider and the End-User
    1. Example of shared information that a website could request (atleast 4), this could be randomized and doesn't necessarily have to be the same, and could request different characters from the listed strings.
        1. DOB XX/uu/eeYY
        2. Twitter: xxxxpppp
        3. Secondary Phone: zzzvvZZiii
        4. Random (saved by you): nnnnnnnfffffmmmmmm
        5. Transaction id: On date 30/12/2024@18:24hoursUTC : IIIIooooooooo
        6. Secondary Email: tttxxxxyyyyxx@yyyyyyy.com …etc
4. Website then asks the user to enter some of the masked character values.
    1. Please enter: XXxxYYvvZZffff (This would generate the seed pin that is used to generate the seed string (Check the library code.
5. Website prompts: (This information are the ones you setup on the website at the time of registration or during update)
    1. Enter The Pin:
    2. Enter the domino pattern:
        1. Three 6 by 6 matrix, to represent around 108 printable characters of ascii.
    3. Check DominoPasswordGen2.html , in this webapp the seed pin isn't used and and the seed string is fixed.
6. The website then sends the seed pin, pin and pattern along with username, random server string and nonce to the underlying layer or external component on the client machine.
7. The component then calculates the shared key and creates an encrypted link between the End-Users browser and website's server component.
    1. Seed pin + Pre-seed string (known to both server and client) = Seed String
    2. Seed String + Pin = Secure crypto string
    3. Secure crypto string + pattern = 18+ Character long shared key
    4. The shared key itself can be used to authenticate and validate the user.

5. The layer then sends “SAKUA2-Ver1-Header: loginname, server-random:120384981237098173091723847861378479831798413874137984139, timestamp + timezone” in encrypted data to the subdomain server on a specific port.

8. Server on receiving the username and based on the shared information questionnaire calculates the seed pin.
    1. The Server component can now calculate the shared key based on this information.
        1. Seed pin + Pre-seed string (known to both server and client) = Seed String
        2. Seed String + Pin = Secure crypto string
        3. Secure crypto string + pattern = 18+ Character long shared key
    2. Based on the Url the server is expecting a connection to with the shared key on a specific port on a specific subdomain url.
    3. The server component receives the encryption keys for various users, and trys to decrypt the message based on the plaintext info (this is also present part of the encrypted data) “**SAKUA2-Ver1-Header: loginname, server-random:120384981237098173091723847861378479831798413874137984139, timestamp + timezone,** ” associated with the encrypted key and uniquely identifies a user to his or her account.
9. The user is then authenticated and validated.
    1. **FLAW – Unfortunately, using an alternate pin and pattern would result in a same encryption key. But its not easy to calculate, see the FLAW section of this document.**
        1. **To thwart the authorization issue, the anagram of the user password generated using a similar shuffling algorithm and user pin is sent across as well.**
        2. **SAKUA2-Ver1-Header-Password: loginname, anagrampasswordbasedonuserpin,**
10. All HTTP/SMTP/Other protocol data is then sent and received by both client and server components implementing shared key-based TCP/UDP encrypted connection.

The SAKUA2 is not a protocol, all information is displayed and entered is through a web browser and the encrypted connection is created using a private key algorithm.
The SAKUA2 client and server component uses the encrypted connection and HTTP header to validate and authenticate the end user.

The HTTP Header should contain:

SAKUA2-Ver1-Header : loginname, server-random:120384981237098173091723847861378479831798413874137984139, timestamp + timezone,
SAKUA2-Ver1-Header-Password: loginname, anagrampasswordbasedonuserpin,

These two headers are sent across in the HTTP headers or SMTP headers, if any other binary or text-based protocol is used, then these headers would either be in the first line or the last line of the data or header sections respectively. Headers must end with **“, “.**

NOTE: THERE IS NO RESTRICTION ON PORT NUMBER, use above 1024. Websites can use different server ports for each user-session to avoid flooding.

FLAW

Using an alternate pin and alternate pattern (a different sequence with a same number of it) would yield the same shared key.

However, there are 3 variables here you need to calculate,
X – Alternate pin
Y – Alternate pattern, (new sequence, same number pattern input), two components are present here, the sequence (Y1) and number (Y2).
Z – Validating the encrypted data using the same shared key with the alternative pin and pattern.

To predict the shared key, assuming you create the crypto string using any pin, which is 108 characters long,

Combination would be n!/((n-r)!*r!), where n is 108 and r is the shared key length (which should also be predicted). N is always 108, and r could be from 18+, if its 18, 108!/(90! * 18!).

To ensure that even if these 3 (X, Y1 and Y2, attackers don't know X, Y1 and Y2) successfully faked, and since the user pin and user pattern isn't sent across, the website receives an anagrammed version of password based on the user pin for AA purpose. This should sufficiently thwart MITM attacks and Spoofing attacks (with alt pin and alt pattern).

SAKUA2-Ver1-Header : loginname, server-random:12038498123709817309172384786137847983179841387413798413 9, timestamp + timezone,
SAKUA2-Ver1-Header-Password: loginname, anagrampasswordbasedonuserpin,

These two headers are sent across in the HTTP headers or SMTP headers, if any other binary or text-based protocol is used, then these headers would either be in the first line or the last line of the data or header sections respectively. Headers must end with **“, “.**

NOTE: THERE IS NO RESTRICTION ON PORT NUMBER, use above 1024. Websites can use different server ports for each user-session to avoid flooding.

NOTE

- The pattern length has to 18+ characters long.
- Optionally, a server random could be mixed in with the shared key to create a larger shared key.

Links

(Old Website, Removed, As the name is similar to an IBM technology name.)
https://github.com/anpnrynn/DominoPassword/

Old Website (containing old code for reference only)
https://github.com/anpnrynn/SAKUA2/

Official Github site, the latest code, test apps, android apks, tools and webapp are all present on:
https://github.com/anpnrynn/SAKUA2Random/