GENERATORS OF ONE-TIME TWO-FACTOR AUTHENTICATION PASSWORDS

Olga Ussatova¹, Saule Nyssanbayeva²
¹Al-Farabi Kazakh National University, Almaty, Kazakhstan; ²Institute of Information and Computational Technologies, Almaty, Kazakhstan

Abstract. The paper presents algorithms for generating a one-time two-factor authentication passwords where application of trigonometric functions have been considered. To protect the opening of a one-time password, a secret string is read that consists of a sequence of randomly generated characters. The second factor is due to the fact that the code has a certain validity period. The presented password generators allow the formation of secret words and trigonometric functions that the proposed two-factor authentication method consists of. The algorithm presented was implemented in Java Script. The algorithm includes blocks for checking randomly generated words and functions.

Keywords: password generator, two-factor authentication, data protection

1. Material and methods

The proposed system of information protection based on two-factor authentication using a combination of two factors: permanent and one-time passwords [7]. The user chooses a permanent password (the first factor) himself and uses it when registering an account. Before authorization must be registered in the application using a smartphone.

For the analysis of the generator used the method of complete enumeration [1, 2]. According to this method, the length of the string is taken into account (the length of the string is 5 characters in the appendix) and, for example, the search speed of 100,000 words per second is used. The number of options is calculated by the formula:

\[ S = A^n \]  

where A is the number of characters and n the length of the string.
An example of the analysis of the generator is presented in table 1.

Table 1. Analysis of the generator

| Number of characters | Number of options | Persistence | Search Time |
|----------------------|-------------------|-------------|-------------|
| 1                    | 52                | 5 byte      | Less than a second |
| 5                    | 380204032         | 26 byte     | 63 minutes  |

$S = 52^5 \times 380204032$

Due to the fact that, according to the developed two-factor authentication algorithm, the generation of a one-time password occurs every 20 seconds, the probability of hacking the generated secret word is almost impossible. This confirms the efficiency of the proposed generator.

3. Trigonometric Function Generator

As stated above, the generation of a one-time password is based on the result of the selected trigonometric function, which has a number of variable parameters. The choice is made in accordance with the result of the obtained hash function of the SHA256 standards, where the first characters are used, which will be indices in a table of 256x256 dimension. By this index, the function will be selected and its parameters will be determined. According to the results of the calculation, digits after the comma are taken as a one-time temporary password, starting from the 5th position and 6 digits long.

The resulting number will be a temporary password that must be entered into the application. To implement this method, a generator of trigonometric functions has been developed, the use of which will greatly facilitate the formation of these functions. The algorithm of the generator of the trigonometric function is shown in Figure 1–3.

To generate a trigonometric function, the number of variables is taken as the basis. There are 7 of them in this generator: a, b, c, x, y, p1, p2. Initially, a list of variables is formed, resulting in a random number of variables Count from 1 to the number of variables minus 1. Then, the array is searched through the array with certain variables N times based on a random number from 0 to the length of the array minus 1.

Read variable from the array, which is added in the new array and removed from the old.

After the cycle is completed, a list of variables for the function is formed.

Based on this list, the constituent parts (Math.sin(a), 1 / Math.tan(p2)) of the format – "[[Math.sin()", "Math.cos()", "Math.tan()", "(1 / Math.tan())"]". The ComponentsCount function (the number of elements minus 1) starts the loop through the array with the generated variables. In the loop at each step, a random number componentIndex from 0 to componentsCount is formed. The element with the corresponding value of the componentIndex index is converted by replacing the symbol "with a variable from the list and added to the new array.

As a result, a list of component parts with variables is formed. Next, rows are formed based on a random number from 1 to 3. In the cycle, the components, separated by signs of mathematical expressions, merge randomly.

After receiving the strings, they are joined separated by mathematical expressions.
Software implementation of the generator has the following form:

```javascript
module.exports.getFunc = function() {
  // function generator
  funcVariablesList = []
  let variablesList = ['a', 'b', 'c', 'x', 'y', 'p1', 'p2']
  let variablesCount = getRandomInt(1, variablesList.length - 1)
  for (let i = 0; i <= variablesCount; i++) {
    let variableIndex = getRandomInt(0, variablesList.length - 1)
    funcVariablesList.push(variablesList[variableIndex])
    variablesList.splice(variableIndex, 1)
  }
  return getComponents(funcVariablesList)
}

function getComponents(variablesList) {
  let funcVariablesList = []
  let components = ['Math.sin(*)', 'Math.cos(*)', 'Math.tan(*)',
    '(1/Math.tan(*))', '(*)']
  let componentCount = components.length - 1
  variablesList.forEach(item => {
    let componentIndex = getRandomInt(0, componentCount)
    funcVariablesList.push(components[componentIndex].replace('*', item))
  })
  return getFunction()
}

function getFunction() {
  // forming a block of variables
  let variablesBlock = `(\$\{funcVariablesList.join\(\;\)\}) =>` + `\$\{funcVariablesList[i]}\)`
  // forming a body function
  let bodyFunction = `\$\{variablesBlock\}`
  let bodyFunction = `
  for (let seriesIndex = 0; seriesIndex <= seriesCount; seriesIndex++) { // there will be 2 rows
    let seriesBlock = "
    for (let i = 0; i <= seriesList.length - 1; i++) {
      let expressionIndex = getRandomInt(0, expressions.length - 1)
      if (i != seriesList.length - 1)
        seriesBlock += `\$\{funcComponents[componentIndex]}\` + expressions[expressionIndex]
      else {
        seriesBlock += `\$\{funcComponents[componentIndex]}\`
      }
    }
    seriesList.push(`\$\{seriesBlock\}`)
  }
  return getBlock(\$\{funcVariablesList[i]}\) + `\$\{funcVariablesList[i]}\`'
}

let funcBlock = `\$\{funcVariablesList[i]}\` + `\$\{funcVariablesList[i]}\`'
let func = variablesBlock + funcBlock
let randomMin = getRandomInt(min, max) + Math.floor(Math.random() * (max - min + 1) + min)
```

Fig. 3. Function `getFunction`
Function getRandomInt(min, max) { // function to get a random number for a given range
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

As a result, we obtain a generated string function, which we use to calculate a one-time two-factor authentication password.

4. Conclusion

The use of generators to work in the formation of a one-time password, allows you to enhance the level of protection of the described system. Entropy is traditionally a measure of the strength of passwords - a measure of uncertainty, usually measured in bits. One bit entropy corresponds to the uncertainty of the choice of two passwords, two bits of 4 passwords, etc. The strength of a password should be considered only in the context of a specific password authentication system. This is due to the fact that different systems in varying degrees implement (or do not implement at all) the mechanisms for counteracting attacks aimed at breaking passwords, and also because some systems contain errors or use unreliable algorithms.

References

[1] Alata E., Nicomette V., Kaaniche M., Dacier M., Herrb M.: Lessons learned from the deployment of a high-interaction honeypot. Proc. Dependable Computing Conference (EDCC06), Coimbra, Portugal, October 18-20, 2006, 39-46.

[2] Ayankoya F., Ohwo B.: Brute-Force Attack Prevention in Cloud Computing Using One-Time Password and Cryptographic Hash Function. International Journal of Computer Science and Information Security (IJCIS) 17(3)/2019, 7-19.

[3] Bahaa Q.M.: Preventing brute force attack through the analyzing log. Iraqi Journal of Science 55(3)/2013, 663-667.

[4] https://www.nist.gov (available 02.09.2018).

[5] https://www.seagate.com/files/www-content/solutions-content/security-and encryption/id/docs/fips-sed-ir-mb-605-2-1302-ru.pdf (available 12.10.2018).

[6] Lamar A.: Types of threats to database security (http://www.brighthub.com/ computing/itsecurity/articles/61402.aspx), 2012, (available 18.03.2019).

[7] Nyssanbayeva S., Ussatova O.: Two-factor authentication in the automated control system. International scientific conference Information Science and Applied Mathematics. Almaty, 2018, Vol. II, 239-242.

M.Sc. Olga Ussatova

e-mail: uoa_olga@mail.ru

Al-Farabi Kazakh National University, Institute of Information and Computational Technologies
Ph.D. student in the specialty: «Information Security Systems». In 2003 graduated of the KazNTU – specialty 3704 – “Software computer technology and automated systems”. In 2014 graduated of the Turan University – master’s degree of “Computing system and Software”. Research interests research theme - database protection using two-factor authentication.

ORCID ID: 0000-0002-5276-6118

Prof. Saule Nyssanbayeva

e-mail: sultasha1@mail.ru

Institute of Information and Computational Technologies, Information Security Laboratory. A specialist in the field of information security. The goals and objectives of the research are development and analysis of methods, algorithms and means of cryptographic protection of information based on modular arithmetic during its transmission and storage in info communication systems and networks.

ORCID ID: 0000-0002-5835-4958

otrzymano/received: 15.05.2019 przyjęto do druku/accepted: 15.06.2019