DOI: 10.58254/viti.3.2023.03.21

УДК 004.9

канд. техн. наук Іляш Ю. Ю. ORCID: 0009-0006-5786-7205
(Прикарпатський національний університет ім. Василя Стефаника)

ІМПЛЕМЕНТАЦІЯ СИСТЕМИ ДІАГНОСТИКИ СКЛАДНОСТІ ПАРОЛЯ

Системи діагностики паролів стають все більш актуальними в сучасному світі в зв'язку з ростом кількості інтернет-сервісів та потреби у захисті особистих даних. Діагностика складності пароля дозволяє оцінити стійкість пароля до різних видів атак, що включають брутфорс, словникові атаки та інші методи.

У роботі розроблено систему перевірки складності пароля, яка дозволяє врахувати нові підходи до формування унікальності вибраної послідовності символів на противагу стандартним методам, які використовуються в найпоширеніших системах генерування паролів, таких як довжина, наявність спецсимволів і цифр, використання великих і малих літер. У роботі пропонується здійснювати перевірку на наявність повторень та збігів з найтиповішими паролями, для чого до розробленої системи підключено базу даних на 10 000 паролів.

Для пришвидшення перевірки на збіги використано алгоритм відстані Левенштейна. Алгоритм Левенштейна працює за допомогою динамічного програмування. Він розглядає два рядки, які мають бути порівняні, і створює матрицю, яка показує мінімальну кількість редагувальних операцій, необхідних для перетворення одного рядка в інший. Дана реалізація дозволила отримати більшу швидкодію генерування порівняно з існуючими системами. У роботі детально прописані алгоритми роботи всіх основних етапів перевірки пароля на складність та запропонована кодова реалізація мовою Javascript. Алгоритм Левенштейна може бути корисним для визначення подібності текстів або для пошуку можливих правильних варіантів неправильно написаних слів. Також, його можна використовувати для реалізації автозаповнення в програмах, які використовують введення користувача.

Реалізована система може бути інтегрована в багато різних програм та вебсервісів для автоматичного визначення складності пароля, що дозволяє використовувати її в різних областях, таких як банківська сфера, медичні сервіси та інші. В результаті роботи над проектом було підтверджено, що використання системи діагностики складності пароля може значно зменшити ризик несанкціонованого доступу до систем, що містять важливу конфіденційну інформацію.

Ключові слова: *алгоритм Левенштейна, генерування пароля, складність пароля, брутфорс.*

Y. Iliash Implementation of the password complexity diagnostic system.

Password diagnostic systems are becoming increasingly relevant in the modern world due to the growth of internet services and the need to protect personal data. Evaluating password complexity allows for assessing the strength of passwords against different types of attacks, including brute force, dictionary attacks, and other methods.

In this work, a system for checking password complexity was developed, which takes into account new approaches to forming the uniqueness of the selected sequence of characters. As opposed to standard methods used in the most common password generating systems, such as length, the presence of special characters and numbers, and the use of upper and lower case letters, this work proposes to check for repetitions and matches with the most common passwords. For this purpose, a database of 10,000 passwords was connected to the developed system.

To speed up the checking for matches, the Levenshtein distance algorithm was used. The Levenshtein algorithm works using dynamic programming. It considers two strings that need to be compared and creates a matrix that shows the minimum number of editing operations required to transform one string into the other. This implementation allowed for faster password generation compared to existing systems. The work details the algorithms for all the main stages of password complexity checking and proposes code implementation in Javascript.

The Levenshtein algorithm can be useful for determining text similarity or searching for possible correct variants of misspelled words. It can also be used for implementing autocomplete in programs that use user input.

The implemented system can be integrated into many different programs and web services for automatic password complexity determination, allowing for its use in various areas such as banking, medical services, and others. As a result of work on this project, it has been confirmed that using a password complexity diagnostic system can significantly reduce the risk of unauthorized access to systems containing important confidential information.

Keywords: *levenstein algorithm, password generation, password complexity, brute force.*

Постановка проблеми. Створення пароля високої складності – це досить складна задача. Через високу обчислювальну потужність сучасних комп'ютерів підбір паролів стає значно простішим. З метою безпеки краще використовувати паролі, які будуть складнішими для підбору системами брутфорсу.

У сучасному медійному та мобільному світі розробка нових алгоритмів передбачає візуалізацію досліджень, створення інтерфейсу користувача. Тому не менш важливим

завданням є створення вебсайту, на якому можна буде перевірити свій пароль на складність та отримати поради щодо його покращення.

Завдяки розробленим алгоритмам можна отримати дані про складність паролів та збільшити рівень безпеки користування вебсервісами. Створений алгоритм може порівнювати введені паролі на схожість із паролями, які внесені в його базу паролів, стандартні системи не роблять такої валідації.

Реалізована система допомагатиме створювати складніші паролі, що знизить ризик злому акаунтів та втрати даних.

Аналіз публікацій за темою дослідження.

Велику увагу безпеці даних приділяють в медичній сфері, забезпечуючи віддалений доступ і перевірку стану здоров'я пацієнтів у будь-який час і в будь-якому місці. У таких випадках важливі системи генерування надійного пароля, щоб дані не були викрадені зловмисниками [1; 2]. На сучасному етапі розвитку систем захисту та безпеки значну увагу приділяють так званим ботнетам, які стали основною загрозою для інфраструктури на основі Інтернету речей, націлених на такі вразливості програмного забезпечення, як слабкі паролі або паролі за замовчуванням, щоб зібрати армію скомпрометованих пристроїв, які можуть служити смертоносною кіберзброєю проти цільових систем, мереж і служб. У статтях описують шляхи та зусилля щодо пом'якшення цієї проблеми шляхом розробки системи виявлення вторгнень, яка знаходиться всередині пристроїв Інтернету речей, щоб забезпечити покращену видимість, таким чином досягаючи посилення безпеки таких пристроїв [3; 4].

Проте описані дослідження стосуються вже внутрішніх аспектів систем зберігання даних, неналежна увага приділяється до первинного етапу реєстрації користувачів, та вибір способів захисту даних, зокрема рівень складності пароля до кабінету, акаунту, емейла. Високий рівень складності пароля, вже може на первинному етапі відсікти левову частку зловмисників, або, принаймні, суттєво збільшити час на підбір ключів.

Метою статті є розробка шляхів імплементації системи діагностики складності пароля для зменшення ризику несанкціонованого доступу до систем, що містять важливу конфіденційну інформацію.

Виклад основного матеріалу.

Аналіз задачі та реалізація алгоритмів. Після аналізу систем перевірки паролів на різних вебсайтах було встановлено, що приблизно 95 % сайтів використовують прості правила для перевірки складності пароля: якщо є символи від [a-z], [A-Z], [0-9] та [@-$], то пароль вважається складним. Якщо, наприклад, ввести пароль "Password1@", то він без проблем пройде перевірку сайтом, хоча сучасними комп'ютерами такі паролі зламуються дуже легко, але пароль "whait-ready-black-adddsdf" не пройде і вважатиметься слабким, хоча його підібрати набагато складніше. Тому було вирішено створити набір правил та реалізацію вебсайту, що дозволить робити перевірки надійніше і точніше.

Основними критеріями складності пароля визначено:

– довжину. Якщо додати 2 символи до пароля, складність підбору збільшиться в 500 разів (якщо використовуються символи ASCII). Мінімально рекомендована довжина – 8 символів, оптимальна – більше 11;

– наявність спецсимволів і цифр. Не всі системи підбору паролів використовують спецсимволи під час процесу підбору;

– використання як малих, так і великих букв. Це збільшує діапазон символів і на процес підбору потрібно більше часу;

– відсутність повторень символів на кшталт "ааа" та послідовностей цифр за зростанням чи спаданням на кшталт "1234" та "9876";

– відсутність поширених комбінацій на клавіатурі «qwerty» та «asdfgh»;

– відсутність повторів та стандартних слів "Pass","Password" та інших. Повтори полегшують процес підбору, а стандартні слова внесені в бази паролів систем для брутфорсу.

Використовуючи дані критерії, було створено алгоритм перевірки складності пароля. Його створення було розділено на декілька етапів. Алгоритм було реалізовано методами мови програмування JavaScript. Першим етапом було створення перевірки на довжину.

```
<script type="text/javascript" >
  var strength;
  var pass;
  addEventListener('keydown', function checkpass(){
     strength = 100;
     pass = document.getElementById('password').value;
     if (pass.length < 8) {
        strength -= 90;
     }
     else
     if (pass.length >= 8 && pass.length < 12)
        strength += pass.length;
     else
        strength += pass.length * 5;
  });
</script>
```

У цьому скрипті створено змінну strength, яка буде змінювати своє значення залежно від довжини пароля. Якщо пароль менше 8 символів, її значення буде дорівнювати 10. Якщо від 8 до 11, значення буде дорівнювати 100 + довжина пароля, а якщо більше 12, її значення буде дорівнювати 100 + довжина пароля, помножена на 2.

Ці значення були обрані через те, що паролі до 8 символів підбираються методом брутфорсу до 3 днів, а з кожним наступним символом складність підбору зростає в 256 разів, якщо використовується кодування символів ASCII. При довжині пароля більше 11 символів комп'ютерам стає складніше підбирати паролі, тому що кількість можливих комбінацій зростає у 256 разів з кожним додатковим значенням і обчислювальна потужність сучасних комп'ютерів може бути недостатньою. Наприклад, сильний пароль на 14 символів звичайний комп'ютер може підбирати декілька трильйонів років. Функція checkpass() викликається при натисканні користувачем будь-якої клавіші. Вибір саме такого способу її виклику зумовлений тим, що набагато зручніше просто вводити пароль в текстове поле вебсторінки і він буде перевірятися в реальному часі, ніж вводити пароль на натискати кнопку "перевірити".

Наступним кроком була реалізована перевірка на наявність великих і малих букв, цифр і спецсимволів у паролі.

```
<script type="text/javascript" >
   var strength;
   var pass;
   var s_letters = "qwertyuiopasdfghjklzxcvbnm";
   var c_letters = "QWERTYUIOPLKJHGFDSAZXCVBNM";
   var numbers = "0123456789";
   var special = "!@#$%^&*()_-+=\|/.,:;[]{}";
   var check_s = false;
   var check_c = false;
   var check_n = false;
   var check_sp = false;
   addEventListener('keydown', function checkpass(){
      strength = 100;
      pass = document.getElementById('password').value;
      if (pass.length < 8) {
         strength -= 90;
```

```
        }
        else
        if (pass.length >= 8 && pass.length < 12)
          strength += pass.length;
        else
          strength += pass.length * 5;
        pass = pass.split('');
        for (let i = 0; i < pass.length; i++) {
          if (!check_s && s_letters.indexOf(pass[i]) !== -1) {
            check_s = true;
            strength += 50;
          }
          else if (!check_c && c_letters.indexOf(pass[i]) !== -1){
            check_c = true;
            strength += 50;
          }
          else if (!check_n && numbers.indexOf(pass[i]) !== -1){
            check_n = true;
            strength += 50;
          }
          else if (!check_sp && special.indexOf(pass[i]) !== -1){
            check_sp = true;
            strength += 50;
          }
        }
      });
    </script>
```

Після доповнення даний скрипт отримав змінні s_letters, c_letters, numbers, special, в яких зберігаються малі і великі букви, числа і спецсимволи, які можна використовувати в паролях на будь-яких вебресурсах, та змінні check_s, check_c, check_n та check_sp, які зберігають значення результатів перевірки на наявність малих і великих літер, чисел і спецсимволів. Перевірка відбувається за допомогою циклу. Перед циклом змінна pass, що містить пароль, розбивається на окремі символи, після чого в тілі циклу кожен символ порівнюється із набором символів зі змінних, які були ініціалізовані раніше. Після перевірки значення змінних check_s, check_c, check_n та check_sp встановлюється true, якщо перевірку було пройдено, або false, якщо перевірку не було пройдено. Також за кожну пройдену перевірку до змінної strength додається 50.

Наступним кроком було реалізовано перевірку на повтори трьох і більше символів.

```
<script type="text/javascript" >
    var strength;
    var pass;
    var s_letters = "qwertyuiopasdfghjklzxcvbnm";
    var c_letters = "QWERTYUIOPLKJHGFDSAZXCVBNM";
    var numbers = "0123456789";
    var special = "!@#$%^&*()_-+=\|/.,:;[]{}";
    var check_s = false;
    var check_c = false;
    var check_n = false;
    var check_sp = false;
    var repeated = 0;
    addEventListener('keydown', function checkpass(){
      strength = 100;
```

```
        pass = document.getElementById('password').value;
        if (pass.length < 8) {
          strength -= 90;
        }
        else
        if (pass.length >= 8 && pass.length < 12)
          strength += pass.length;
        else
          strength += pass.length * 5;
        pass = pass.split('');
        for (let i = 0; i < pass.length; i++) {
          if (!check_s && s_letters.indexOf(pass[i]) !== -1) {
            check_s = true;
            strength += 50;
          }
          else if (!check_c && c_letters.indexOf(pass[i]) !== -1){
            check_c = true;
            strength += 50;
          }
          else if (!check_n && numbers.indexOf(pass[i]) !== -1){
            check_n = true;
            strength += 50;
          }
          else if (!check_sp && special.indexOf(pass[i]) !== -1){
            check_sp = true;
            strength += 50;
          }
          if (pass[i]===pass[i-1]){
            repeated++;
          }
          else
            repeated=0;
          if (repeated >= 3){
            strength-=50;
          }
        }
      });
    </script>
```

Для реалізації цієї перевірки було створено змінну repeated. Перевірка відбувається в циклі. Якщо даний символ рівний попередньому, то значення змінної росте. Якщо значення змінної repeated більше або дорівнює 3, то від значення змінної strength віднімається 30.

Наступним кроком була реалізована перевірка на послідовності по зростанню та спаданню, стандартні слова та комбінації на клавіатурі. Для того щоб спростити цей процес, було вирішено створити базу паролів, в якій збережено стандартні послідовності та слова. Вона буде зберігатися в файлі формату .json та імопортуватися в головний скрипт. Для спрощення створення даної бази було використано дані Марка Бернета зі статті "10,000 Top Passwords" [3].

```
<script type ="text/Javascript>
import {passes} from "./passes.json"
passes = passes.split('\n');

function levenshtein(s1, s2, costs) {
```

```
    var i, j, l1, l2, flip, ch, chl, ii, ii2, cost, cutHalf;
    l1 = s1.length;
    l2 = s2.length;
    costs = costs || {};
    var cr = costs.replace || 1;
    var cri = costs.replaceCase || costs.replace || 1;
    var ci = costs.insert || 1;
    var cd = costs.remove || 1;
    cutHalf = flip = Math.max(l1, l2);
    var minCost = Math.min(cd, ci, cr);
    var minD = Math.max(minCost, (l1 - l2) * cd);
    var minI = Math.max(minCost, (l2 - l1) * ci);
    var buf = new Array((cutHalf * 2) - 1);
    for (i = 0; i <= l2; ++i) {
      buf[i] = i * minD;
    }
    for (i = 0; i < l1; ++i, flip = cutHalf - flip) {
      ch = s1[i];
      chl = ch.toLowerCase();
      buf[flip] = (i + 1) * minI;
      ii = flip;
      ii2 = cutHalf - flip;
      for (j = 0; j < l2; ++j, ++ii, ++ii2) {
        cost = (ch === s2[j] ? 0 : (chl === s2[j].toLowerCase()) ? cri : cr);
        buf[ii + 1] = Math.min(buf[ii2 + 1] + cd, buf[ii] + ci, buf[ii2] + cost);
      }
    }
    return buf[l2 + cutHalf - flip];
}
var strength, length_of, pass, s_letters = "qwertyuiopasdfghjklzxcvbnm",
    c_letters = "QWERTYUIOPLKJHGFDSAZXCVBNM", numbers = "0123456789",
    special = "!@#$%^&*()_-+=\|/.,:;[]{}", check_s, check_c,
    check_n, check_sp, st_use, pass_ch, repeated = 0;
addEventListener('keydown', function checkpass(){
    pass = document.getElementById('password').value;
    strength = 100;
    length_of = pass.length;
    check_s = false; check_c = false;
    check_n = false; check_sp = false; st_use = false;
    if (pass.length < 8) {
      strength -= 90;
    }
    else
    if  (pass.length >= 8 && pass.length < 12)
      strength += pass.length;
    else
      strength += pass.length * 5;
    pass = pass.split('');
    for (let i = 0; i < pass.length; i++) {
      if (!check_s && s_letters.indexOf(pass[i]) !== -1) {
        check_s = true;
        strength += 50;
```

```
        }
        else if (!check_c && c_letters.indexOf(pass[i]) !== -1){
            check_c = true;
            strength += 50;
        }
        else if (!check_n && numbers.indexOf(pass[i]) !== -1){
            check_n = true;
            strength += 50;
        }
        else if (!check_sp && special.indexOf(pass[i]) !== -1){
            check_sp = true;
            strength += 50;
        }
        if (pass[i]===pass[i-1]){
            repeated++;
        }
        else
            repeated=0;
        if (repeated >= 3){
            strength-=30;
        }
    }
        for(let j = 0; j < passes.length; j++)
        if (levenshtein(pass,passes[j].split('')) < pass.length - 3) {
            st_use = true;
            strength = -60;
            break;
    }
    console.log(st_use);
});
</script>
```

Для перевірки схожості пароля з паролями з бази було реалізовано функцію алгоритму відстані Левенштейна [1; 2]. За допомогою цього алгоритму пароль порівнюється із паролями з бази.

Після рефакторингу коду було отримано скрипт, який повністю підходить для перевірки паролів на складність. Переваги даного скрипту над скриптами, які використовуються для тестування паролів на інших вебплатформах:

точність;

можливість тонкої настройки;

ефективність роботи;

швидкодія (порівняно з методом Монте-Карло).

Розробка інтерфейсу

Після завершення роботи над алгоритмом було створено простий односторінковий сайт для демонстрації його роботи. Першим етапом було створення HTML-документа та його наповнення.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PassCheck</title>
    <link rel="stylesheet" href="css/index.css">
```

```
    <link rel="script" href="js/checker.js">
    <link rel="preconnect" href="https://fonts.gstatic.com">
    <link href="https://fonts.googleapis.com/css2?family=Mukta&display=swap" rel="stylesheet">
</head>
<body>
    <h1 class="header">Перевірте Ваш пароль</h1>
    <label for="password"></label><input id="password" type="password"
placeholder="Введіть пароль" value=""><br>

    <script type="text/javascript" src="js/checker.js"></script><br>
    <div id="pi" class="progressbar" hidden>
      <div id="bard" class="bar">
        <br>
      </div>
    </div>
    <p id ="Funny"></p>

</body>
</html>
```

В результаті було отримано ось таку сторінку (рис. 1):

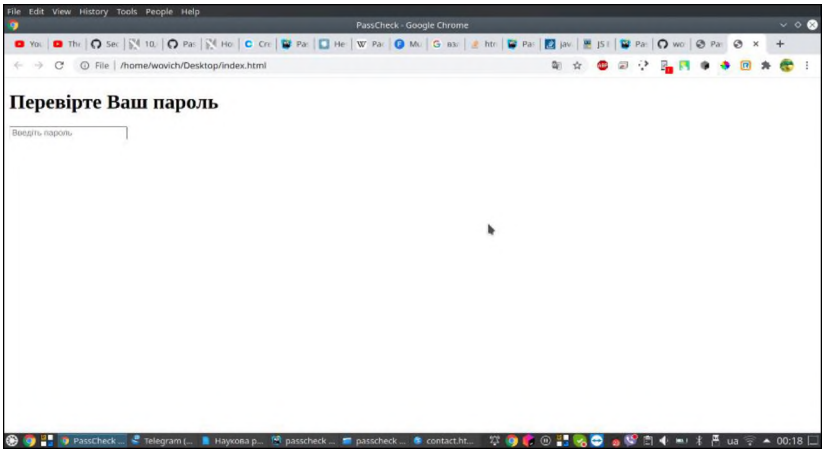

Рис. 1. Загальний вигляд сторінки

Після створення HTML-документа та написання коду сторінки було створено CSS-документ.

```
body{
    background-color: #093824;
    text-align: center;
    margin: 0;
    font-family: Mukta,Serif;
}

.header {
    color: #78FECF;
    margin: 35px 0 45px 0;
    padding: 30px 0px 10px 0px;
    font-size: 55px;
}
#Funny{
```

```
    color: #78FECF;
    font-size: 16px;
}

#password{
    background-color: #78FECF;
    text-align: center;
    font-size: 45px;
    width: 100%;
    padding: 50px 0px 60px 0px;
    outline: none;
    border: none;
}

#password::placeholder{
    color: #093824;
    caret-color : #78FECF;
}
#password:active, #password :hover,#password :focus {
    outline: none;
    border: none;
    caret-color : #093824;
    color: #093824;

}
.progressbar{
    background-color: #C6CCB2;
    padding: 5px 10px;
    width:480px;
    margin: auto;
    border-radius: 15px;
}

#bard{
    background-color: #BF4E30;
    max-width: 480px;
    border-radius: 15px;
}
```

В результаті сайт набув презентабельного вигляду (рис. 2).

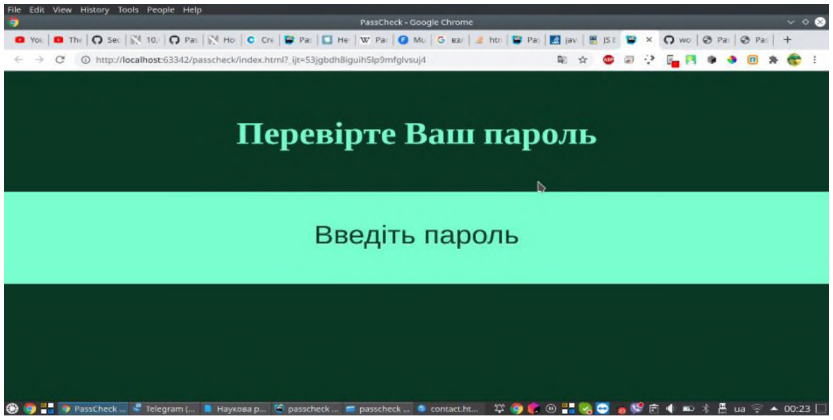

Рис. 2. Вигляд сторінки тестування

Наприкінці роботи до вебсайту було підключено алгоритм перевірки паролів на складність. Після завершення роботи над вебсайтом було виконано повний цикл тестування (рис. 3, 4).

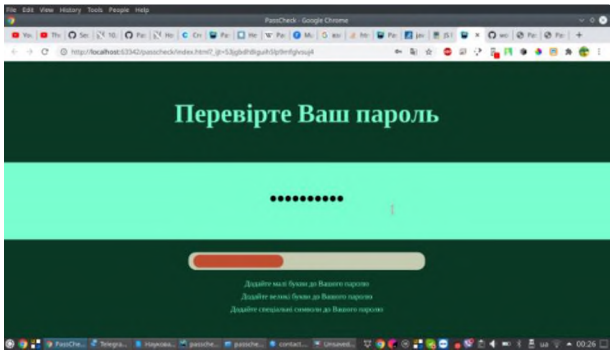

Рис. 3. Перевірка пароля

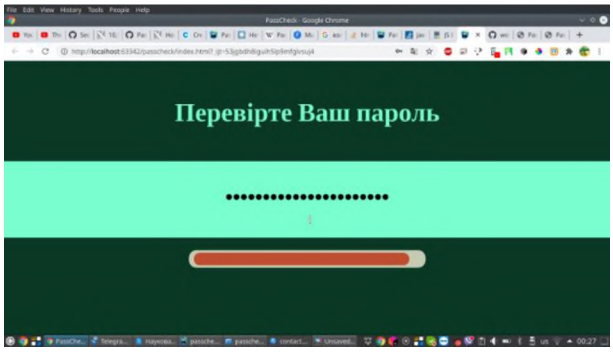

Рис. 4. Результат перевірки пароля

Висновки. Після аналізу проблем, пошуку та опрацювання інформації було отримано надійний алгоритм перевірки паролів на надійність та створено вебсайт, на якому можна отримати інформацію про надійність свого пароля та поради щодо його покращення. Алгоритм, що є результатом даного дослідження, має такі переваги над іншими:

вища точність;

можливість тонкої настройки за допомогою зміни коефіцієнтів;

швидкість роботи (порівняно з методом Монте-Карло) та легкість підключення до існуючого сайту;

перевірка пароля на схожість із паролями з підключеної бази.

Завдяки цьому можливо збільшити рівень безпеки користуванням мережею Інтернет та знизити ризик втрати даних чи крадіжки особистої інформації. Також через можливість доповнення чи заміни бази паролів та корегування коефіцієнтів даний алгоритм більш зручний і ефективний, ніж алгоритми інших сайтів чи вебплатформ.

Подальші напрямки досліджень даної тематики слід сконцентрувати на розробках інноваційних методів створення паролів, зокрема вивченні нових підходів до створення паролів, які були б інтуїтивно зрозумілі, легкі для запам'ятовування та міцні з точки зору криптографічних вимог. Важливим аспектом в будь-яких сучасних дослідженнях є використання машинного навчання та штучного інтелекту, зокрема використання алгоритмів машинного навчання для аналізу та класифікації паролів з метою виявлення слабких та вразливих варіантів.

Не менш важливим напрямком є дослідження поведінкової аналітики користувачів, зокрема аналіз звичок користувачів та їхніх підходів до створення паролів для виявлення шаблонів та слабких місць, які можуть бути використані зловмисниками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wagner R. A., Fischer M. J. The string-to-string correction problem // Journal of the ACM. Volume 21. Issue 1. Pp. 168–173. January 01, 1974. URL: https://doi.org/10.1145/321796.321811.

2. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. Т. 163. № 4. С. 845–848.

3. Bruce Schneier. Snakeoil: Warning Sign #5: Ridiculous key lengths» // Schneier on Security. February 15, 1999. URL: https://www.schneier.com/crypto-gram/archives/1999/0215.html.

4. Burnett Mark. 10,000 Top Passwords // Medium. URL: https://xato.net/10-000-top-passwords-6d6380716fe0.

5. Справочник по HTML // HTMLBOOK: веб-сайт. URL: http://htmlbook.ru/html.