Case

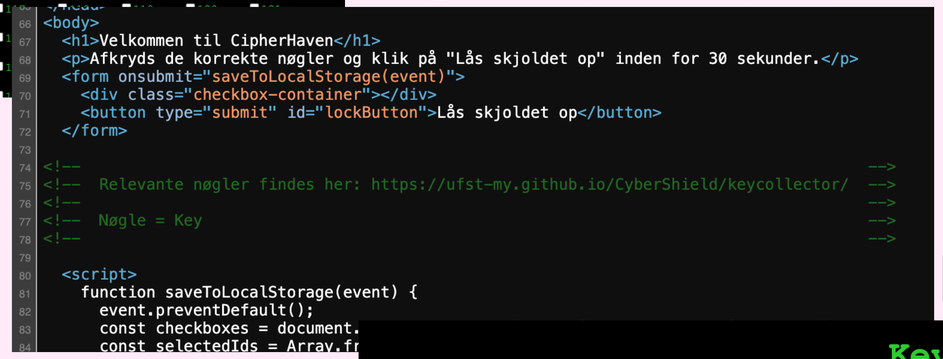
Operation CyberShield: Helten fra det mørke web

Simon Echers

Udforskning

Åbning af siden

Inspektion af kilde

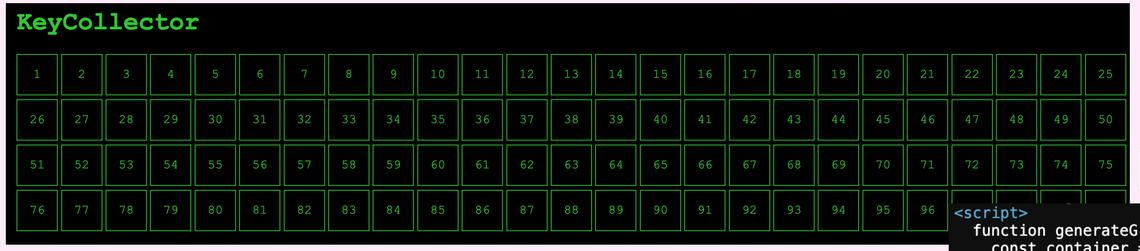


```
<body>
  <h1>Velkommen til CipherHaven</h1>
  <p>Afkryds de korrekte nøgler og klik på "Lås skjoldet op" inden for 30 sekunder.</p>
  <form onsubmit="saveToLocalStorage(event)">
    <div class="checkbox-container"></div>
    <button type="submit" id="lockButton">Lås skjoldet op</button>
  </form>

<!--                                                    -->
<!--  Relevante nøgler findes her: https://ufst-my.github.io/CyberShield/keycollector/  -->
<!--                                                    -->
<!--  Nøgle = Key                                       -->
<!--                                                    -->

  <script>
    function saveToLocalStorage(event) {
      event.preventDefault();
      const checkboxes = document.
      const selectedIds = Array.fr
```

KeyCollector fundet

Udforskning

Side 1 af KeyCollector



Inspektion af kilde

```
<script>
  function generateGrid() {
    const container = document.getElementById("grid");
    for (let i = 1; i <= 100; i++) {
      const cell = document.createElement("div");
      cell.className = "grid-item";
      cell.textContent = i;
      if (i % 10 === 1) {
        cell.setAttribute("data-id", "Key");
      } else {
        const variants = ["Ley", "Kea", "Kay", "Kee", "Koy"];
        const randomText = variants[Math.floor(Math.random() * variants.length)];
        cell.setAttribute("data-id", randomText);
      }
      container.appendChild(cell);
    }
  }
  window.onload = generateGrid;
/script>
```

Side 4 af KeyCollector

Inspektion af kilde

```
<script>
  const container = document.getElementById("container");
  for (let i = 301; i <= 400; i += 10) {
    const group = document.createElement("div");
    group.className = "group";
    const header = document.createElement("div");
    header.className = "group-header";
    header.textContent = `Gruppe ${i}-${i + 9}`;
    const content = document.createElement("div");
    content.className = "group-content";
    header.onclick = function () {
      if (!content.innerHTML) {
        for (let j = i; j < i + 10; j++) {
          const item = document.createElement("div");
          item.className = "grid-item";
          item.textContent = j;
          if (j % 10 === 1) {
            item.setAttribute("data-id", "Key");
          }
          content.appendChild(item);
        }
      }
      content.style.display = content.style.display === "block" ? "none" : "block";
    };
    group.appendChild(header);
    group.appendChild(content);
    container.appendChild(group);
  }
</script>
```

Observationer

1. Siderne har ikke samme udseende
   - Kan dog identificeres på …getElementById(x). x = “container”, når vi har en side som side 4.
2. Vi kan aflæse koderne næsten direkte
3. Objekterne er genereret ved hjælp af JavaScript(tror jeg)…
4. Der er 1000 kasser på hovedsiden -> kan ikke tastes ind manuelt på under 30 sekunder

På nuværende tidspunkt har man nok til at generere koderne uden særlig meget programmering. Koderne er netop tallene 1, 11, .., 101, 111, .., 981, 991. 100 styks.

Kode - struktur

KeyCollector

Identificerer og scraper gyldige nøgler

KeyInserter

Markerer nøgler og bekræfter dem

DEN ENDELIGE STRUKTUR AF DE INDIVIDUELLE KLASSER

KeyCollector - get_script_code

```
def get_script_code(self,page_nos):
    """
    page_nos er en liste af værdier af siderne man ønsker.

    Opdaterer self.htmls med HTML-koden i <scripts> på sider med koderne
    man ønsker.
    """
    self.htmls = defaultdict(list) #For at være sikker på at vi ikke overskriver den flere gange.
    make_url = lambda page_no: self.base_url + f"page{page_no}.html" #Lille funktion til at lave URL
    page_nos = [x for x in page_nos if 1 <= x and x <= 10] #Filtrere værdier udenfor rækkevidde.
    for page_no in page_nos:
        url = make_url(page_no)
        html = BeautifulSoup(req.get(url).content, 'html.parser')
        as_str = str(html.find_all("script").pop()) #Vi har kun brug for <script> delen, nok til at identificere.
        identifier = re.findall(r"document.getElementById\(\"container\"\)",as_str) #For at se om de er grupperede.
        key = 'groups' if identifier else 'cells'
        self.htmls[key].append(as_str)
```

Denne del af koden er til forberedelse inden selve behandlingsarbejdet går i gang

Jeg instantierer vores dictionary igen, for ikke at skrive det samme stykke HTML ind flere gange.
Så opretter jeg en lille hjælpefunktion til dynamisk at sætte det rigtige sidetal ind i linket.
Filtrerer værdier som ikke er i intervallet for siderne fra.

KeyCollector - get_script_code

```python
def get_script_code(self,page_nos):
    """
    page_nos er en liste af værdier af siderne man ønsker.

    Opdaterer self.htmls med HTML-koden i <scripts> på sider med koderne
    man ønsker.
    """
    self.htmls = defaultdict(list) #For at være sikker på at vi ikke overskriver den flere gange.
    make_url = lambda page_no: self.base_url + f"page{page_no}.html" #Lille funktion til at lave URL
    page_nos = [x for x in page_nos if 1 <= x and x <= 10] #Filtrere værdier udenfor rækkevidde.
    for page_no in page_nos:
        url = make_url(page_no)
        html = BeautifulSoup(req.get(url).content, 'html.parser')
        as_str = str(html.find_all("script").pop()) #Vi har kun brug for <script> delen, nok til at identificere.
        identifier = re.findall(r"document.getElementById\(\"container\"\)",as_str) #For at se om de er grupperede.
        key = 'groups' if identifier else 'cells'
        self.htmls[key].append(as_str)
```

Denne del af koden er hvor vi indsamler det relevante HTML-kode

For hvert sidetal i listen laver skaber vi det respektive link.
Laver en forespørgsel på sidens indhold.
Omdanner indholdet til HTML-kode og finder den relevante script-del og dernæst identificerer om det er en side med grupper.
Til sidst tilføjes det til den dictionary vi oprettede alt efter hvad karakter HTML koden havde

KeyCollector - treat_groups

```python
def treat_groups(self):
    """
    Vi skal finde dem for både groups og cells, så jeg skriver en metode for hver.

    Givet en groups script html skal den finde koderne og returnere dem.
    """
    html_script_code = self.htmls['groups']
    RegEx_loops = r"for \(.*{"
    RegEx_from_to = r". \+?<?=? \d*"
    RegEx_If = r". % \d* === \d*"
    for script_code in html_script_code:
        loops = re.findall(RegEx_loops, script_code)
        outer_loop = loops[0] #inner_loops = loops[1], hvis der skal mere fleksibilitet. Men alle værdier tjekkes i d
        outer_from, outer_to, step = [int(element.split()[-1]) for element in re.findall(RegEx_from_to, outer_loop)]
        If = re.findall(RegEx_If,script_code).pop() #Man kan blive ved med at gøre det mere fleksibelt, men nu laver
        modulus = lambda x: x % int(If.split(" ")[2]) == int(If.split(" ")[-1])
        for value in range(outer_from, outer_to+1): #Her har vi naturligvis antaget at vi tjekker alle værdierne imel
            if modulus(value):
                self.keys.add(value)
```

Forberedelse til at udvinde nøgler fra HTML-koden

Vi indhenter først det HTML-kode hvor koderne var grupperet. Efterfølgende definerer vi vores regular expressions mønstre til at udvinde nøglerne fra HTML-koden(det er sådan jeg valgte at gøre det)

KeyCollector - treat_groups

```python
def treat_groups(self):
    """
    Vi skal finde dem for både groups og cells, så jeg skriver en metode for hver.

    Givet en groups script html skal den finde koderne og returnere dem.
    """
    html_script_code = self.htmls['groups']
    RegEx_loops = r"for \(.*{"
    RegEx_from_to = r". \+?<?=? \d*"
    RegEx_If = r". % \d* === \d*"
    for script_code in html_script_code:
        loops = re.findall(RegEx_loops, script_code)
        outer_loop = loops[0] #inner_loops = loops[1], hvis der skal mere fleksibilitet. Men alle værdier tjekkes i
        outer_from, outer_to, step = [int(element.split()[-1]) for element in re.findall(RegEx_from_to, outer_loop)]
        If = re.findall(RegEx_If,script_code).pop() #Man kan blive ved med at gøre det mere fleksibelt, men nu laver
        modulus = lambda x: x % int(If.split(" ")[2]) == int(If.split(" ")[-1])
        for value in range(outer_from, outer_to+1): #Her har vi naturligvis antaget at vi tjekker alle værdierne ime
            if modulus(value):
                self.keys.add(value)
```

Har begynder indhentningen af nøgler

Først finder vi ALLE loops(vi har to for grupperne).
Vi gemmer resultatet fra det yderste loops endepunkter(samt skridtet, men det bliver ikke anvendt til noget).
Derefter finder vi den del af koden der bestemmer betingelsen for at være en nøgle og danner en funktion ud fra det.
Til sidst tester vi ALLE(diskuteret) værdier mellem de yderste loops endepunkter og tilføjer dem til mængden af nøgle hvis de opfylder betingelsen.

KeyInserter - fill_boxes

```
def fill_boxes(self,keys):
    """
    Det her har voldet flest problemer. Men det lykkedes at finde
    v.h.a. deres værdi og gudskelov er der intet der overlapper.

    Finder derefter knappen hvilket er nemt fordi den har et tydel
    """
    elements = [self.browser.find_by_value(str(i)) for i in keys]
    for element in elements:
        element.click()
    button = self.browser.find_by_id('lockButton').pop()
    button.click()
```

Vi starter med at finde de sideelementer som er en kasse man kan vinge af - disse er identificeret ud ved den værdi de hver især har.

Derefter klikker vi på hver af dem og til sidst finder vi knappen for at bekræfte koderne v.h.a. dens tag.

main - hovedlogikken

```python
from KeyFetcher import KeyFetcher
from BrowserInteraction import KeyInserter

page = "https://ufst-my.github.io/CyberShield/" #U
browser_name = "chrome" #Vigtigt at man har browse
page_nos = [i+1 for i in range(10)]

keyfetcher = KeyFetcher()
keyfetcher.get_keys(page_nos)
keys = keyfetcher.keys
keyinserter = KeyInserter(browser_name,page,keys)
```