

KIERUNEK: CYBERBEZPIECZEŃSTWO

# Bezpieczeństwo aplikacji webowych Projekt

## Raport 1: Zadanie rozgrzewkowe

Autor:  
Aleksandra Angelika Zwolińska

PROWADZĄCY PRACĘ:  
Mgr inż. Przemysław Świercz

# Spis treści

Wstęp.....	3
1. Cel ćwiczenia .....	3
2. Pierwsza gra programistyczna.....	3
2.1. Rozwiązanie pierwszego poziomu .....	4
2.2. Rozwiązanie drugiego poziomu .....	5
2.3. Rozwiązanie trzeciego poziomu .....	6
2.4. Rozwiązanie czwartego poziomu.....	8
2.5. Rozwiązanie piątego poziomu .....	10
2.6. Rozwiązanie szóstego poziomu .....	12
2.7. Rozwiązanie siódmego poziomu .....	14
2.8. Rozwiązanie ósmego poziomu .....	17
3. Druga gra programistyczna .....	20
3.1. Rozwiązanie pierwszego poziomu .....	20
3.2. Rozwiązanie drugiego poziomu .....	21
3.3. Rozwiązanie trzeciego poziomu .....	22
3.4. Rozwiązanie czwartego poziomu.....	22
3.5. Rozwiązanie piątego poziomu .....	24
3.6. Rozwiązanie szóstego poziomu .....	26
3.7. Rozwiązanie siódmego poziomu .....	27
3.8. Rozwiązanie ósmego poziomu .....	28
3.9. Rozwiązanie dziewiątego poziomu.....	31
Spis ilustracji .....	33

# Wstęp

Gry programistyczne uwzględnione w tym raporcie zostały stworzone przez UW-TEAM i można je znaleźć na ich oficjalnej stronie <https://uw-team.org/>. Należy zaznaczyć, że opisane zostały tylko dwie pierwsze gry z serii HACKME.

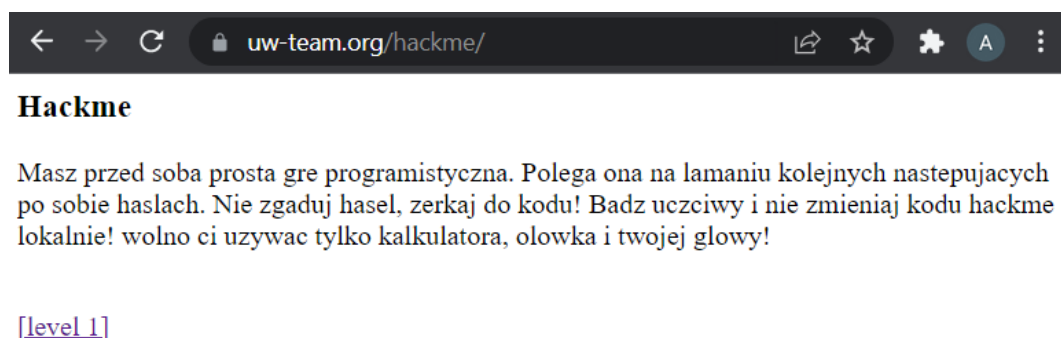
W przygotowanym dokumencie znajdują się rozwiązania poziomów wraz z przedstawieniem toku rozumowania, który doprowadził do znalezienia poprawnego hasła.

## 1. Cel ćwiczenia

Głównym celem ćwiczenia było przejście wszystkich poziomów w dwóch grach programistycznych oraz sporządzenia raportu z przejścia obu gier. Do rozwiązania każdego z poziomów należy podać odpowiednie hasło. Dozwolone było korzystanie z kodu źródłowego, kalkulatora, kartki oraz ołówka.

## 2. Pierwsza gra programistyczna

Pierwsza gra programistyczna znajdowała się pod adresem: <https://uw-team.org/hackme/>



**Rysunek 2.1. Strona główna pierwszej gry HACKME**

Aby rozwiązać każdy z poziomów, po przejściu na konkretny poziom był sprawdzany kod źródłowy strony. Żeby go wyświetlić należało skorzystać ze skrótu klawiszowego **CTRL + U** lub kliknąć na stronę **prawym przyciskiem myszy**, a następnie z listy rozwijanej wybrać **"Wyświetl źródło strony"**. Inną opcją było wybranie opcji **„zbadaj”** po kliknięciu **prawym przyciskiem myszy** na stronę lub przy pomocy skrótu klawiszowego **CTRL + SHIFT + I**. Używając tej opcji najbardziej będą nas interesować zakładki **Elements** oraz **Sources**.

## 2.1. Rozwiązanie pierwszego poziomu

Kod pierwszego poziomu zawierał gotowe rozwiązanie, które wystarczyło wpisać jako hasło i przejść do kolejnego poziomu. Najpierw jednak warto omówić pokrótce kod, który dostajemy.

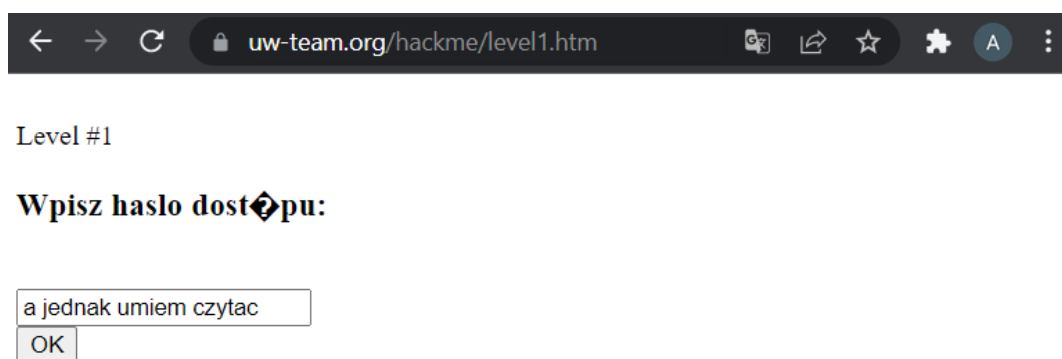
W samym skrypcie możemy znaleźć trzy sekcje (bloki kodu). Pierwszą z nich jest funkcja *sprawdz?()* na samej górze, która nigdy się nie wykona ponieważ nie jest ona nigdzie wywołana w kodzie.

Drugą częścią jest kod odpowiedzialny za wyświetlenie prostego interfejsu strony. Mamy wyświetloną informację odnośnie numeru poziomu na którym się znajdujemy oraz prośbę o wpisanie hasła. Kolejnym elementem jest pole tekstowe do wpisania hasła, którego id to właśnie hasło. Ostatnim elementem jest przycisk „OK”, który po wciśnięciu wykonuje funkcję *sprawdz()*.

Trzecią i ostatnią sekcją jest funkcja bliźniaczo podobna do tej pierwszej. Funkcja ta jest wywoływana po kliknięciu przycisku „OK” na stronie. Jej zadaniem jest porównanie wartości pola tekstowego, gdzie podajemy hasło i sprawdzenie czy podany przez nas string brzmi *„a jednak umiem czytać”*. Jeśli podany warunek jest spełniony zostajemy przekierowani na stronę o adresie [https://uw-team.org/hackme/ok\\_next.htm](https://uw-team.org/hackme/ok_next.htm), natomiast jeśli podana wartość w polu o id hasło jest niepoprawna, użytkownik dostaje komunikat „Złe hasło”.

```
Zawijaj tekst ☐
1 <HTML>
2 <script>
3 function sprawdz?(){
4   if (document.getElementById('haslo').value=='i am too lame') {self.location.href="zaq.htm";} else {alert('Zle haselko :')}}
5 }
6 </script>
7 <br>Level #1
8 <h3>Wpisz haslo dostepu:</h3>
9 <br><input type="text" name="haslo" id="haslo">
10 <br><input type="button" value="OK" onClick="sprawdz()">
11 </HTML>
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55 <script>
56 function sprawdz(){
57   if (document.getElementById('haslo').value=='a jednak umiem czytac') {self.location.href='ok_next.htm';} else {alert('Zle haselko :')}}
58 }
59 </script>
60
```

Rysunek 2.1.1. Kod źródłowy pierwszego poziomu gry HACKME 1.0



*Rysunek 2.1.2. Rozwiązanie pierwszego poziomu gry HACKME 1.0*

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: *a jednak umiem czytac*. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: [https://uw-team.org/hackme/ok\\_next.htm](https://uw-team.org/hackme/ok_next.htm), gdzie możemy rozwiązać kolejny poziom gry.

## 2.2. Rozwiązanie drugiego poziomu

Większość kodu na poziomie drugim jest podobna do poprzedniego, dlatego omówienie tych części zostało pominięte. Nowym elementem kodu jest dodanie dodatkowego pliku do skryptu (*haselko.js*). Zawartość tego pliku jest pokazana na [rysunku 2.2.2](#). Mamy tam zawarte dwie zmienne które są kluczowe do przejścia na kolejny poziom gry.

Funkcja *sprawdz()* jest skonstruowana w dokładnie taki sam sposób jak poprzednio, z trzema wyjątkami:

- Pierwszą z nich jest parametr do którego przyrównywana jest wartość pola tekstowego o id hasło. W tym przypadku jest to zmienna *has*, której wartość odpowiada temu co należy wpisać aby przejść na kolejny poziom.
- Drugą różnicą jest inaczej podany adres strony do jakiej zostaniemy przekierowani po podaniu prawidłowego hasła. W tym miejscu mamy podaną zmienną *adresik*, która została zawarta właśnie w tym dodatkowym pliku. Zatem po wprowadzeniu poprawnego hasła i naciśnięciu przycisku OK zostaniemy przekierowani na stronę o adresie: <https://uw-team.org/hackme/formaster.htm>
- Trzecia różnica jest zmianą kosmetyczną, ponieważ zmienił się komunikat wyświetlany gdy użytkownik poda złe hasło. W tym przypadku alert brzmi: „Nie... to nie to hasło... „.

```

Zawijaj tekst
1 <HTML>
2 <script src="haselko.js"></script>
3 <script>
4 function sprawdz(){
5   if (document.getElementById('haslo').value==has) {self.location.href=adresik;} else {alert('Nie... to nie to haslo...');}
6 }
7 </script>
8 <br>Level #2
9 <h3>Wpisz hasło dostępu:</h3>
10 <br><input type="text" name="haslo" id="haslo">
11 <br><input type="button" value="OK" onClick="sprawdz()">
12 </HTML>
13

```

*Rysunek 2.2.1. Kod źródłowy drugiego poziomu gry HACKME 1.0*

```

← → ↻ uw-team.org/hackme/haselko.js
OFERTA – Projekto... ALEKSANDRA ZWO... Kenwood KVL8400... Budżet mieszkania...

var has='to bylo za proste';
var adresik='formaster.htm';

```

*Rysunek 2.2.2. Dodatkowa zawartość dodana do drugiego poziomu gry HACKME 1.0*

```

← → ↻ uw-team.org/hackme/ok_next.htm
OFERTA – Projekto... ALEKSANDRA ZWO... Kenwood KVL8400... Budżet mieszkania...

Level #2

Wpisz hasło dostępu:

to bylo za proste
OK

```

*Rysunek 2.2.3. Rozwiązanie drugiego poziomu gry HACKME 1.0*

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: **to było za proste**. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <https://uw-team.org/hackme/formaster.htm>, gdzie możemy rozwiązać kolejny poziom gry.

## 2.3. Rozwiązanie trzeciego poziomu

Analizę kodu zaczniemy od mniej istotnych części kodu jak np. funkcja `right(e)`, która jest zdefiniowana, ale nie jest nigdzie w kodzie wywołana. Kolejnym takim kawałkiem kodu jest kolejna funkcja `prawy(tx)`, która również nigdy nie jest wywoływana. Oba te kawałki kodu mogłyby zostać usunięte ze skryptu i nie wpłynęłoby to w żaden sposób na działanie kodu.

**Funkcja `sprawdz()`** jest skonstruowana w dokładni taki sam sposób jak poprzednio. Wartością która jest porównywana jest zmienna `ost`. Jest ona również wartością wykorzystywaną w adresie strony dla kolejnego poziomu gry.

```

Zawijaj tekst
1 <HTML>
2 <script>
3 function right(e) {
4   if (navigator.appName == 'Netscape' &&
5       (e.which == 3 || e.which == 2))
6     return false;
7   else if (navigator.appName == 'Microsoft Internet Explorer' &&
8       (event.button == 2 || event.button == 3)) {
9     alert('Prawy nie działa...');
10    return false;
11  }
12  return true;
13 }
14 var dod='unknow';
15 function prawy(txx){
16   var txt=txx;
17   document.onmousedown=right;
18   if (document.layers) window.captureEvents(Event.MOUSEDOWN);
19   window.onmousedown=right;
20 }
21 prawy();
22 var literki='abcdefgh';
23 var ost='';
24 function losuj(){
25   ost=literki.substring(2,4)+'que'+dod.substring(3,6);
26 }
27
28 function sprawdz(){
29   losuj();
30   if (document.getElementById('haslo').value==ost) {self.location.href=ost+'.htm';} else {alert('Nie... to nie to haselko...');}
31 }
32 </script>
33 <br>Level #3
34 <h3>Wpisz hasło dostępu:</h3>
35 <br><input type="text" name="haslo" id="haslo">
36 <br><input type="button" value="OK" onClick="sprawdz()">
37 </HTML>
38

```

Rysunek 2.3.1. Kod źródłowy trzeciego poziomu gry HACKME 1.0

Najważniejszą częścią tego poziomu jest rozszyfrowanie jaka wartość kryje się pod zmienną *ost*. Zaczniemy od tego, że początkowo zmienna ta jest pusta. Jej wartość jest zmieniona przez *funkcję losuj*, która jest wywołana na samym początku *funkcji sprawdz()*.

```

function losuj() {
  ost=literki.substring(2,4)+'que'+dod.substring(3,6); }

```

Aby odszyfrować wartość zmiennej *ost*, najpierw należy zrozumieć czym jest *substring*. Jest to funkcja wywoływana dla konkretnej zmiennej, która jest typu string (ciąg znaków). Zgodnie z wzorem przedstawionym poniżej, odnosi się ona do ciągu znaków mieszczącego się w podanym zakresie. Przy czym należy pamiętać, że miejsca są indeksowane od zera, a podane przez nas miejsce końcowe nie będzie już brane pod uwagę.

*[nazwa\_zmiennej].substring(miejsce\_startowe,miejsce\_końcowe)*

Odwołując się już bezpośrednio do naszego kodu, łatwo zauważyć że zmienna *ost* to ciąg znaków wyciętych ze zmiennej *literki* i zmiennej *dod* oraz dodaniem między nimi ciągu „que”.

Zmienna dod					
0	1	2	3	4	5
u	n	k	n	o	w

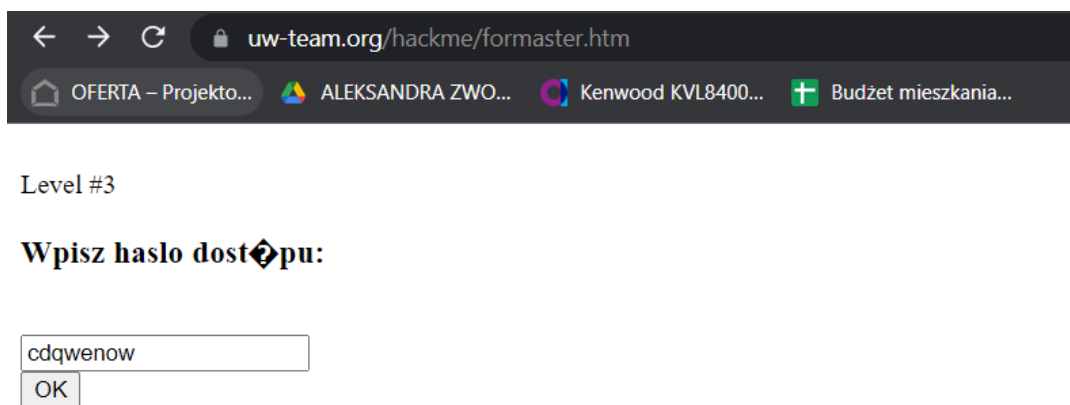
Zmienna literki							
0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

Funkcja `substring` dla zmiennej `literki`, gdzie miejscem startowym jest znak o indeksie 2, a miejscem końcowym znak o indeksie 4. Zgodnie z tym co zostało wcześniej przedstawione odnośnie samego działania funkcji `substring` i patrząc na wyżej zamieszczoną tabelkę widzimy, że wartość jaka zostanie zwrócona przez tą funkcję jest ciąg znaków „`cd`”.

```
ost = 'cd' + 'que' + dod.substring(3,6);
```

Ten sam schemat należy przeprowadzić ze zmienną `dod`. W tym przypadku pozycja startowa ustawiona jest na znak o indeksie 3, a znakiem końcowym jest znak o indeksie 6. Zatem wartość jaka zostanie zwrócona przez tą funkcję jest ciąg znaków „`now`”.

```
ost = 'cd' + 'que' + 'now';
```



Rysunek 2.3.2. Rozwiązanie trzeciego poziomu gry HACKME 1.0

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: `cdqwenow`. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <https://uw-team.org/hackme/cdqwenow.htm>, gdzie możemy rozwiązać kolejny poziom gry.

## 2.4. Rozwiązanie czwartego poziomu

Na tym poziomie gry cała zabawa rozgrywa się w funkcji `sprawdz()`, która do tej pory wyglądała zawsze prawie tak samo. Pierwszą rzeczą jaka zostaje wykonana jest przypisanie wartości wpisanej do pola tekstowego o nazwie `haslo` do zmiennej `zaq`.

Następnie sprawdzany jest pierwszy warunek, gdzie zwracana jest wartość `true` lub `false` w wyniku użycia funkcji `isNaN`, gdzie jako argument podajemy wcześniej zdefiniowaną zmienną `zaq`. Funkcja `isNaN` sprawdza, czy przekazany jej argument nie ma wartości `NaN` („nie-liczba”).



Zatem jeśli użytkownik poda wartość liczbową w polu hasła, funkcja zwróci wartość *false* i zostaną wykonane instrukcje z bloku poleceń podanych po słowie *else*. Jeśli jednak użytkownik poda hasło nie będące liczbą, funkcja zwróci wartość *true*, a sam użytkownik dostanie komunikat o tym, że hasło jest niepoprawne.

W kolejnym bloku kodu mamy zdefiniowanie zmiennej *wynik* i przypisanie do niej konkretnej wartości liczbowej, która jest hasłem dla tego poziomu. Wynika to z kolejnej instrukcji warunkowej, która mówi nam: jeśli wartość pola tekstowego gdzie użytkownik podaje hasło jest równe wartości przypisanej do zmiennej *wynik* to możemy przejść do kolejnego poziomu gry. Oczywiście jeśli wynik nie będzie się zgadzał użytkownik dostanie powiadomienie o tym, że podał złe hasło.

```
Zawijaj tekst ☐
1 <HTML>
2 <script>
3 function sprawdz(){
4   zaq=document.getElementById('haslo').value;
5   if (isNaN(zaq)) {alert('Zle haslo!')} else {
6     wynik=(Math.round(6%2)*(258456/2))+(300/4)*2/3+121;
7     if (zaq==wynik) {self.location.href='go'+wynik+'.htm';} else {alert('Zle haslo!')}
8   }
9 }
10 </script>
11 <br>Level #4
12 <h3>I co by tu teraz zrobic?</h3>
13 <br><input type="text" name="haslo" id="haslo" size=20>&nbsp;<input type="button" value="?" onClick="sprawdz()">
14 </HTML>
```

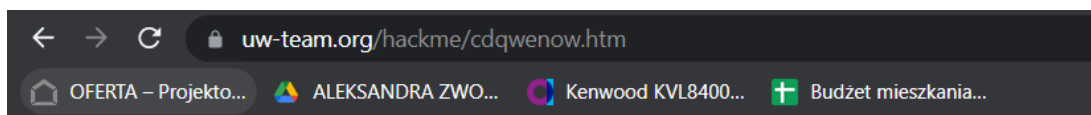
Rysunek 2.4.1 Kod źródłowy czwartego poziomu gry HACKME 1.0

Skupmy się teraz jednak na rozwiązaniu jaka liczba kryje się pod zmienną *wynik*. Idąc od lewej strony natykamy się na *Math.round()*. Jest to wywołanie metody *round* dla obiektu *Math*, której parametrem zawsze jest liczba. Metoda ta zwraca daną liczbę w zaokrągleniu do najbliższej liczby całkowitej.

W tym przypadku przekazany parametr jest liczbą całkowitą, ponieważ mamy działanie *6%2*, gdzie *%* jest operatorem reszty. Operacja „modulo” zwraca resztę pozostałą po podzieleniu jednej liczby przez drugą. Zatem wynikiem tej operacji jest zero, ponieważ liczba 2 całkowicie mieści się w liczbie 6 - trzy razy. Oznacza to że dzielenie *258456/2* można pominąć, ponieważ ta część równania i tak całościowo da wynik zero.

$$\text{wynik} = 0 + (300/4) * 2/3 + 121 = 0 + 75 * 2/3 + 121 = 0 + 50 + 121 = 171$$

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: *171*. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <https://uw-team.org/hackme/go171.htm>, gdzie możemy rozwiązać kolejny poziom gry.



Level #4

I co by tu teraz zrobic?

Rysunek 2.4.2. Rozwiązanie czwartego poziomu gry HACKME 1.0

## 2.5. Rozwiązanie piątego poziomu

Na tym poziomie mamy zabawę z czasem. Pierwsze linijki kodu odnoszą się do stworzenia instancji, która pozwala na pracę z danymi opisującymi datę i czas. Ponieważ nie zostały dostarczone żadne argumenty, konstruktor tworzy obiekt *Date* dla aktualnej daty oraz czasu lokalnego. Mamy również przypisanie wartości dla zmiennej *seconds* przy pomocy funkcji *getSeconds()*, która zwraca sekundy dla określonej daty w obiekcie *now* (czyli liczbę całkowitą z przedziału od 0 do 59).

Funkcja *czas()* jest związana z samym wyświetlaniem licznika czasu na stronie. Funkcja *innerHTML* pobiera zbiór, w tym wypadku liczb od 0 do 59. Natomiast *setTimeout()* uruchamia fragment kodu ( w tym wypadku funkcji *czas()*) po określonym odstępie czasu ( 1 minuta).

Zawijaj tekst ☐

```
1 <HTML>
2 <script>
3 var now = new Date();
4 var seconds = now.getSeconds();
5
6 function czas(){
7   now = new Date();
8   seconds = now.getSeconds();
9   txt.innerHTML=seconds;
10  setTimeout('czas()',1);
11 }
12
13 function sprawdz(){
14   ile=((seconds*(seconds-1))/2)*(document.getElementById('pomoc').value%2);
15   if (ile==861) {self.location.href=seconds+'x.htm'} else {alert('Zle haslo!');}
16 }
17 </script>
18 <br>Level #5
19 <h3>Zamek czasowy</h3>
20 <br><div id="txt">?</div>
21 <br>Cyfra pomocnicza: <input type="text" size=3 name="pomoc" id="pomoc"><br>
22 <br><input type="button" value="[wejdz]" onClick="sprawdz()">
23 <script>czas();</script>
24 </HTML>
25
```

Rysunek 2.5.1 Kod źródłowy piątego poziomu gry HACKME 1.0

Przejdźmy teraz do najważniejszej dla nas części kodu, która pozwoli nam wejść na kolejny poziom gry. Funkcja *sprawdz()* zaczyna się od policzenia wartości dla zmiennej *ile*. Następnie sprawdzana jest instrukcja warunkowa, gdzie wartość zmiennej *ile* przyrównywana jest do wartości **861**. Jeśli warunek jest spełniony użytkownik zostaje przekierowany na następny poziom.

Przyjrzyjmy się teraz jak liczona jest sama zmienna *ile*. Jej wartość uzależniona jest od wartości zmiennej *seconds* oraz *wartości pomocniczej* jaką podaje użytkownik.

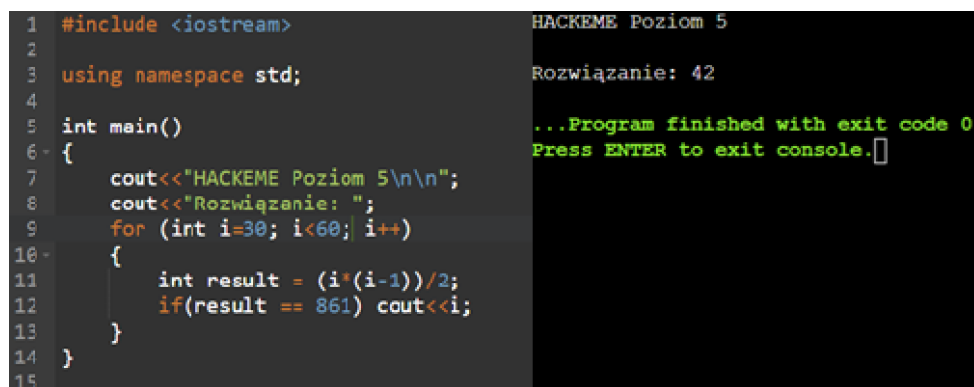
$$ile = (seconds * (seconds - 1)) / 2 * pomoc \% 2$$

Nietypowo zaczniemy analizę równania od końca, ponieważ to sporo nam uprości. Mamy wyrażenie *pomoc % 2*. Jak już wiemy z wcześniejszego przykładu jest to wyrażenie modulo, które zwraca resztę z dzielenia. W tym wypadku *pomoc* to zmienna, którą wprowadza użytkownik. Należy jednak zwrócić uwagę na to przez jaką liczbę dzielimy. Jest to liczba 2, co oznacza że każda podana liczba parzysta (czyli podzielna przez dwa) da nam wynik zero. To oznacza, że wartość zmiennej *ile* również będzie wtedy równa zero. Stąd wniosek jest taki, że należy podać *liczbę nieparzystą np. 1,3,5,...*, ponieważ to da nam wartość wyrażenia równą 1.

W takim razie zostało nam rozszyfrowanie w jakiej sekundzie należy wywołać funkcję *sprawdz()*. Możemy to zrobić na dwa sposoby. Albo policzyć na kalkulatorze ręcznie każdą parę liczb i sprawdzić, która z nich daje nam wynik równy 1722, albo możemy napisać krótki program, który policzy to za nas. Warto zauważyć również, że patrząc na liczby jakie potrzebujemy znaleźć nie warto jest sprawdzać liczb z zakresu mniejszego niż 30.

$$861 = (seconds * (seconds - 1)) / 2$$

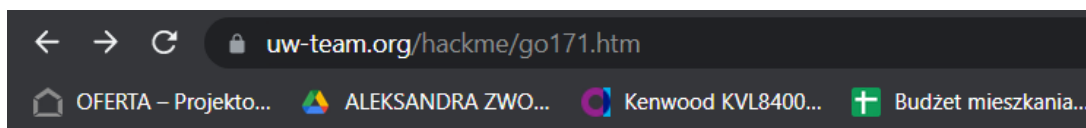
$$1722 = seconds * (seconds - 1)$$



```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout<<"HACKEME Poziom 5\n\n";
8     cout<<"Rozwiązanie: ";
9     for (int i=30; i<60; i++)
10     {
11         int result = (i*(i-1))/2;
12         if(result == 861) cout<<i;
13     }
14 }
15
```

HACKEME Poziom 5  
Rozwiązanie: 42  
...Program finished with exit code 0  
Press ENTER to exit console.

Rysunek 2.5.2. Pomocniczy skrypt napisany w C++



Level #5

## Zamek czasowy

42

Cyfra pomocnicza:

*Rysunek 2.5.3 Rozwiązanie piątego poziomu gry HACKME 1.0*

Zatem poprawną cyfrą pomocniczą jaką należało wpisać aby przejść na kolejny poziom była *liczba nieparzysta* (w moim przypadku była to cyfra 1) i wcisnąć przycisk *[wejdź]* kiedy licznik wskazywał 42 sekundę. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <https://uw-team.org/hackme/42x.htm>, gdzie możemy rozwiązać kolejny poziom gry.

## 2.6. Rozwiązanie szóstego poziomu

Ten poziom jest podobny do poziomu trzeciego. Przede wszystkim są tu stosowane funkcje *substring*, które zostały opisane właśnie w tamtym podrozdziale. Sama funkcja *sprawdz()* oprócz swojej standardowej formy, zawiera parę zmiennych pomocniczych potrzebnych do przekształcenia wartości zmiennej *hsx* oraz operacje, które ją przekształcają. To właśnie wartość kryjąca się pod tą zmienną interesuje nas najbardziej.

Zaraz po zdefiniowaniu zmiennych, mamy fragment kodu z pętlą *for*. Wartość parametru *i* jest zwiększana o 2 oczka, co oznacza że *pętla wykona się 3 razy* (dla wartości 1,3 oraz 5).

Po zwiększeniu zmiennej pomocniczej *licznik*, mamy wyrażenie warunkowe, które w zależności od tego czy *licznik* jest liczbą parzystą czy nieparzystą, odpowiednia wartość zostanie przypisana do zmiennej pomocniczej *znak*. Mianowicie, jeśli licznik jest równy 1, po wykonaniu wcześniej omówionej operacji modulo (*licznik%2*) i przyrównaniu wartości tej operacji do zera, warunek będzie nieprawdziwy. Oznacza to, że wykona się instrukcja przypisująca do zmiennej pomocniczej *znak* wartość 'x'.

Ostatnią instrukcją w pętli *for* jest aktualizacja wartości zmiennej *hsx* przez dodanie (do aktualnej wartości) stringa utworzone z funkcji *substring* oraz dodanie wartości zmiennej *znak*.

Wartości zmiennych zdefiniowanych przed pętlą for			
<i>lit = 'abcdqepolsrc'</i>	<i>lit = 'abcdqepolsrc'</i>	<i>lit = 'abcdqepolsrc'</i>	<i>lit = 'abcdqepolsrc'</i>
<i>licznik = 0</i>	<i>licznik = 0</i>	<i>licznik = 1</i>	<i>licznik = 2</i>
<i>hsx = ''</i>	<i>hsx = ''</i>	<i>hsx = 'bx'</i>	<i>hsx = 'bxd_'</i>
<i>znak = ''</i>	<i>znak = ''</i>	<i>znak = 'x'</i>	<i>znak = '_'</i>
Wykonanie operacji w pętli for			
<i>for ( i=1 ; i&lt;=5 ; i+=2 )</i>	<i>dla i = 1</i>	<i>dla i = 1</i>	<i>dla i = 1</i>
<i>licznik ++;</i>	<i>licznik = 1;</i>	<i>licznik = 1;</i>	<i>licznik = 1;</i>
<i>if ( licznik % 2 == 0 )</i>	<i>( 1%2 = 1 )</i>	<i>( 2%2 = 0 )</i>	<i>( 3%2 = 1 )</i>
<i>znak = '_ ';</i>	<i>--&gt; false</i>	<i>--&gt; true</i>	<i>--&gt; false</i>
<i>znak = 'x ';</i>	<i>znak = 'x ';</i>	<i>znak = '_ ';</i>	<i>znak = 'x ';</i>
<i>hsx+=lit.substring(i,i+1)</i> <i>+ znak;</i>	<i>lit.substring(1,2) + x</i> <i>hsx += 'b' + 'x'</i>	<i>lit.substring(3,4) + _</i> <i>hsx += 'd' + '_'</i>	<i>lit.substring(5,6) + x</i> <i>hsx += 'e' + 'x'</i>

Po opuszczeniu pętli for mamy kolejną zmianę wartości zmiennej *hsx*. Tym razem mamy użytą funkcję *substring* wykorzystującą jako parametr odniesienie do długości całego wyrażenia (*funkcja lenght*). Zwraca nam ona informację o długości stringa *hsx*, czyli podaję liczbę ilości znaków.

```

Zawijaj tekst ☐
1 <HTML>
2 <script>
3 var lit='abcdqepolsrc';
4 function sprawdz(){
5   var licznik=0;
6   var hsx='';
7   var znak='';
8   zaq=document.getElementById('haslo').value;
9   for (i=1; i<=5; i+=2){
10    licznik++;
11    if ((licznik%2)==0) {znak='_';} else {znak='x';}
12    hsx+=lit.substring(i,i+1)+znak;
13  }
14  hsx=hsx.substring(hsx.length-3,hsx.length);
15  if (zaq==hsx) {self.location.href=hsx+'.htm';} else {alert('Zle haslo!');}
16 }
17 </script>
18 <br>Level #6
19 <h3>Wprowadz haslo: </h3>
20 <br><input type="text" name="haslo" id="haslo">
21 <br><input type="button" value="OK" onClick="sprawdz()">
22 </HTML>
23

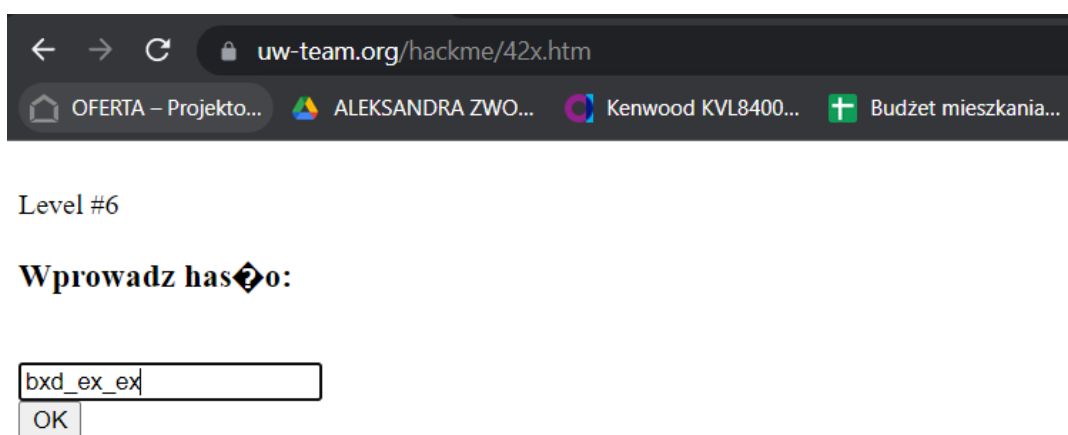
```

Rysunek 2.6.1 Kod źródłowy szóstego poziomu gry HACKME 1.

Oznacza to, że funkcja `length` wywołana dla zmiennej `hsx` zwróci wartość 6. Jest to ilość znaków występująca w `string` przypisanym do zmiennej `hsx`, już po wykonaniu pętli `for`.

Zmienna <code>hsx</code> po pętli <code>for</code>					
0	1	2	3	4	5
b	x	d	_	e	x

Skoro już znamy wartość zwróconą przez funkcję `length`, możemy rozważyć ostatnią operację na zmiennej `hsx`. Do obecnej wartości zostanie dodany ciąg znaków ze zmiennej `hsx`, gdzie miejscem startowym jest 3 pozycja (`length - 3`), a pozycją końcową będzie pozycja 6. Oznacza to, że zostanie dodany ciąg trzech ostatnich znaków zmiennej `hsx`.



Rysunek 2.6.2 Rozwiązanie szóstego poziomu gry HACKME 1.0

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: `bxd_ex_ex`. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: [https://uw-team.org/hackme/bxd\\_ex\\_ex.htm](https://uw-team.org/hackme/bxd_ex_ex.htm), gdzie możemy rozwiązać kolejny poziom gry.

## 2.7. Rozwiązanie siódmego poziomu

Na tym poziomie również główną rolę odgrywa funkcja `sprawdz()`. Na samym początku mamy przypisanie wartości wpisanej przez użytkownika do zmiennej `zsq` oraz zdefiniowanie zmiennej pomocniczej `wyn`. Kolejny blok kodu to pętla `for`, czyli najbardziej interesujący nas fragment. Ostatnią linijką funkcji `sprawdz()` jest dobrze nam znana instrukcja warunkowa, z której dowiadujemy się że wartość zmiennej `wyn` musi być równa „`plxszn_xrv`”.

```

Zawijaj tekst ☐
1 <HTML>
2 <script>
3 function sprawdz(){
4   zaq=document.getElementById('haslo').value;
5   wyn='';
6   for (i=0; i<=zaq.length-1; i++){
7     lx=zaq.charAt(i);
8     ly='';
9     if (lx=='a') {ly='z'}
10    if (lx=='b') {ly='y'}
11    if (lx=='c') {ly='x'}
12    if (lx=='d') {ly='w'}
13    if (lx=='e') {ly='v'}
14    if (lx=='f') {ly='u'}
15    if (lx=='g') {ly='t'}
16    if (lx=='h') {ly='s'}
17    if (lx=='i') {ly='r'}
18    if (lx=='j') {ly='q'}
19    if (lx=='k') {ly='p'}
20    if (lx=='l') {ly='o'}
21    if (lx=='m') {ly='n'}
22    if (lx=='n') {ly='m'}
23    if (lx=='o') {ly='l'}
24    if (lx=='p') {ly='k'}
25    if (lx=='q') {ly='j'}
26    if (lx=='r') {ly='i'}
27    if (lx=='s') {ly='h'}
28    if (lx=='t') {ly='g'}
29    if (lx=='u') {ly='f'}
30    if (lx=='v') {ly='e'}
31    if (lx=='w') {ly='d'}
32    if (lx=='x') {ly='c'}
33    if (lx=='y') {ly='b'}
34    if (lx=='z') {ly='a'}
35    if (lx==' ') {ly='_'}
36    wyn+=ly;
37  }
38  if (wyn=='plxszn_xrv') {self.location.href=wyn+'.htm';} else {alert('Zle haslo!');}
39  }
40 </script>
41 <br>Level #7
42 <h3>Wprowadz haslo:</h3>
43 <br><input type="text" name="haslo" id="haslo">
44 <br><input type="button" value="OK" onClick="sprawdz()">
45 </HTML>
46

```

**Rysunek 2.7.1** Kod źródłowy siódmego poziomu gry HACKME 1.0

Przechodząc już do samej analizy kodu w pętli *for* najpierw warto omówić ile razy sama pętla się wykona. Mamy pomocniczą zmienną *i* w celu przejścia przez całą długość hasła, które zostało podane przez użytkownika, a sama zmienna jest zwiększana o jeden. Ilość wykonań pętli odnosi się do długość zmiennej *zaq* pomniejszonej o 1. Zabieg pomniejszenia jest potrzebny, ponieważ zmienna *i* jest indeksowana od zera, a funkcja *length* podaje ilość znaków licząc od 1.

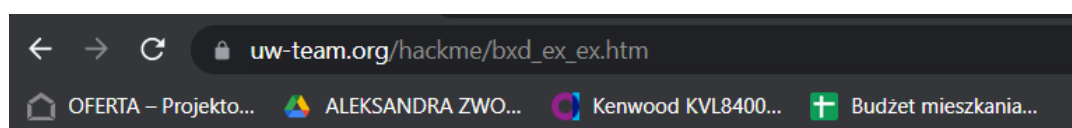
Do zmiennej pomocniczej *lx* przypisana jest wartość zwracana przez funkcję *charAt()*. Zadaniem tej funkcji jest zwrócenie określonego znaku z łańcucha znaków. Tłumacząc to na przykładzie: do zmiennej pomocniczej *lx* przypisany jest znak znajdujący się na pozycji *i* w ciągu znaków przypisanemu do zmiennej *zaq*.

Następnie mamy dużą ilość instrukcji warunkowych. Schemat tych instrukcji to sprawdzenie czy zmienna *lx* (czyli znak z łańcucha zmiennej *zaq* znajdujący się na pozycji *i*) to konkretna litera z alfabetu. Jeśli warunek jest spełniony to do zmiennej pomocniczej *ly* jest

przypisany konkretny znak. Po wykonaniu wszystkich instrukcji warunkowych mamy aktualizację zmiennej **wyn**, po przez dodanie do ciągu znaków aktualnej wartości zmiennej **ly**.

Ponieważ zmienna **wyn** musi mieć wartość „**plxszn\_xrv**”, aby przejść na kolejny poziom gry, a jest ona tworzona przez dodanie wartości **ly**, która jest tworzona na zasadzie zmiany konkretnej wartości na inną zdefiniowaną w bloku z instrukcjami warunkowymi. Zatem wystarczy odczytać te wartości w odwrotnej kolejności, co pozwoli nam wpisać odpowiednie hasło.

Odszyfrowanie wartości zmiennej wyn		
<i>ly = p</i>	<i>lf ( lx == ' k ' ) ( ly = ' p ' )</i>	<i>lx = k</i>
<i>ly = l</i>	<i>lf ( lx == ' o ' ) ( ly = ' l ' )</i>	<i>lx = o</i>
<i>ly = x</i>	<i>lf ( lx == ' c ' ) ( ly = ' x ' )</i>	<i>lx = c</i>
<i>ly = s</i>	<i>lf ( lx == ' h ' ) ( ly = ' s ' )</i>	<i>lx = h</i>
<i>ly = z</i>	<i>lf ( lx == ' a ' ) ( ly = ' z ' )</i>	<i>lx = a</i>
<i>ly = n</i>	<i>lf ( lx == ' m ' ) ( ly = ' n ' )</i>	<i>lx = m</i>
<i>ly = _</i>	<i>lf ( lx == ' ' ) ( ly = ' _ ' )</i>	<i>lx = ' '</i>
<i>ly = x</i>	<i>lf ( lx == ' c ' ) ( ly = ' x ' )</i>	<i>lx = c</i>
<i>ly = r</i>	<i>lf ( lx == ' i ' ) ( ly = ' r ' )</i>	<i>lx = i</i>
<i>ly = v</i>	<i>lf ( lx == ' e ' ) ( ly = ' v ' )</i>	<i>lx = e</i>



Level #7

**Wprowadz hasło:**


**Rysunek 2.7.2. Rozwiązanie siódmego poziomu gry HACKME 1.0**

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: **kocham cie**. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: [https://uw-team.org/hackme/plxszn\\_xrv.htm](https://uw-team.org/hackme/plxszn_xrv.htm), gdzie możemy rozwiązać kolejny poziom gry.



## 2.8. Rozwiązanie ósmego poziomu

Na samym początku mamy przypisanie wartości do dwóch zmiennych. Nie są one jednak wykorzystywane nigdzie w kodzie, dlatego zostaną one przeze mnie pominięte.

```
Zawijaj tekst ☐
1 <HTML>
2 <script>
3 var roz='dsabdkgsawqqqlsahdas'; var tmp=roz.substring(2,5)+roz.charAt(12);
4 document.write('<\s'+ 'c'+ 'r'+ 'i'+ 'p'+ 't src="%7A%73%65%64%63%78%2E%6A%73"><\s'+ 'c'+ 'r'+ 'i'+ 'p'+ 't'+ '>');
5 function sprawdz(){
6   zaq=document.getElementById('haslo').value; wyn=''; alf='qwertyuioplkjhgfdsazxcvbnm';
7   qet=0; for (i=0; i<=10; i+=2){
8     get+=10; wyn+=alf.charAt(qet+i); qet++;}
9   wyn+=eval(ax*bx*cx);
10  if (wyn==zaq) {self.location.href=wyn+'.htm';} else {alert('Zle haslo!');}
11 }
12 </script>
13 <br>Level #8
14 <h3>Wprowadz haslo:</h3>
15 <script src="%70%61%73%73%77%64.js"></script>
16 <br><input type="text" name="haslo" id="haslo">
17 <br><input type="button" value="OK" onClick="sprawdz()">
18 </HTML>
19
```

Rysunek 2.8.1 Kod źródłowy ósmego poziomu gry HACKME 1.0

Zanim przejdziemy do kolejnej linijki kodu warto wspomnieć, że do kodu załączony jest w sposób jawny plik o nazwie passwd.js. Sam plik nie zawiera nic ciekawego, co zostało pokazane na [rysunku 2.8.2](#).

```
< > ↺ uw-team.org/hackme/passwd.js
OFERTA – Projekto... ALEKSANDRA ZWO... Kenwood KVL8400... Budżet mieszkania...

// -----
// Niespodzianka!
// Tu nie ma hasła...
// szukaj dalej...
// -----
```

Rysunek 2.8.2. Dodatkowa zawartość dołączona do ósmego poziomu gry HACKME 1.0

W czwartej linijce kodu mamy załączony w nietypowy sposób dodatkowy plik. Dużo lepiej widać to kiedy patrzymy na kod zamieszczony w zakładce zbadaj element.

**`<script src="%7A%73%65%64%63%78%2E%6A%73"></script>`**

W samym pliku mamy zawarte zmienne, które będą nam potrzebne do znalezienia poprawnego hasła i ukończenia gry. Możemy od razu wyliczyć wartość dla każdej z tych zmiennych. Przy czym sama zmienna `get`, która jest tu zdefiniowana, a jej wartość zostaje zmodyfikowania w dalszej części kodu, nie wpływa w żaden sposób na samo rozwiązanie.

```

<html>
  <head>
    <script>...</script>
    ... <script src="%7A%73%65%64%63%78%2E%6A%73"></script> == $0
    <link id="chromealerabat-link" rel="stylesheet" type="text/css" href="chrome-extension://dacdinoicboceafielngnmjjplncljhj/content.css">
  </head>
  <body>
    <br>
    "Level #8 "
    <h3>Wprowadz hasło:</h3>
    <script src="%70%61%73%73%77%64.js"></script>
    <br>
    <input type="text" name="haslo" id="haslo">
    <br>
    <input type="button" value="OK" onclick="sprawdz()">

```

Rysunek 2.8.3. Kod wyświetlony po zbadaniu elementu na stronie

```

← → ↻ uw-team.org/hackme/zsedcx.js
OFERTA – Projekto... ALEKSANDRA ZWO... Kenwood KVL8400... Budżet mieszkania...

ax=eval(2+2*2);
bx=eval(ax/2);
cx=eval(ax+bx);
get=0;

```

Rysunek 2.8.4. Ukryta zawartość dodana do ósmego poziomu gry HACKME 1.0

$$ax = (2 + 2 * 2) = 6$$

$$bx = (ax / 2) = 6 / 2 = 3$$

$$cx = (ax + bx) = 6 + 3 = 9$$

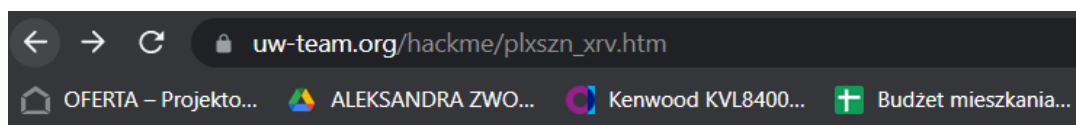
Na początku funkcji *sprawdz()* mamy zdefiniowane parę zmiennych, które przydadzą się do znalezienia poprawnego hasła. Są one wykorzystywane w *pętli for*, którą właśnie teraz omówimy. Zmienna pomocnicza *i* jest zdefiniowana na wartość zero oraz zwiększa swoją wartość o dwa. Ponieważ warunkiem ograniczającym pętlę jest wartość mniejsza lub równa dziesięć, *pętla wykona się dokładnie sześć razy*. Po wykonaniu skończeniu się *pętli for*, wartość zmiennej *wyn* jest równa ciągowi znaków w postaci „*grupjf*”.

Po wykonaniu się *pętli for* mamy ostatnią modyfikację wartości zmiennej *wyn*. Modyfikacja ta wykorzystuje wcześniej wspomniane przeze mnie zmienne *ax*, *bx* oraz *cx*. Pojawia się również przemilczane wcześniej *eval()*. Funkcja ta oblicza ciąg i używa go jako kod skryptu do wykonania. W naszym przypadku oblicza iloczyn wartości *ax*, *bx* i *cx* (otrzymujemy wartość 162).

**Przykład:** *eval („ x=1; y=10; document.write( x\*y ) ”);*    **WYNIK:** 10

**Przykład:** *console.log( eval ( new String ( ' 2 + 2 ' ) ));*    **WYNIK:** 2 + 2

Wykonanie operacji w pętli for			
Zmienna <i>alf</i> = 'qwertyuioplkjhgfdsazxcvbnm'			
	<i>wyn</i> += <i>alf.charAt (qet + i)</i>	<i>wyn</i> = ''	<i>qet</i> = 0
dla <i>i</i> = 0	<i>wyn</i> += <i>alf.charAt ( 0 )</i>	<i>wyn</i> = 'q '	<i>qet</i> = 1
dla <i>i</i> = 2	<i>wyn</i> += <i>alf.charAt ( 3 )</i>	<i>wyn</i> = 'qr '	<i>qet</i> = 2
dla <i>i</i> = 4	<i>wyn</i> += <i>alf.charAt ( 6 )</i>	<i>wyn</i> = 'qru '	<i>qet</i> = 3
dla <i>i</i> = 6	<i>wyn</i> += <i>alf.charAt ( 9 )</i>	<i>wyn</i> = 'grup '	<i>qet</i> = 4
dla <i>i</i> = 8	<i>wyn</i> += <i>alf.charAt ( 12 )</i>	<i>wyn</i> = ' grupj '	<i>qet</i> = 5
dla <i>i</i> = 10	<i>wyn</i> += <i>alf.charAt ( 15 )</i>	<i>wyn</i> = ' grupjf '	<i>qet</i> = 6



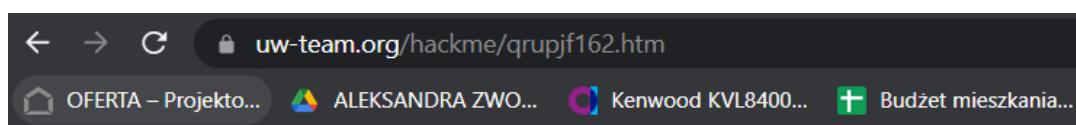
Level #8

Wprowadz hasło:



Rysunek 2.8.5. Rozwiązanie ósmego poziomu gry HACKME 1.0

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: **grupjf162**. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <https://uw-team.org/hackme/grupjf162.htm>, gdzie dostajemy informację o ukończeniu gry.



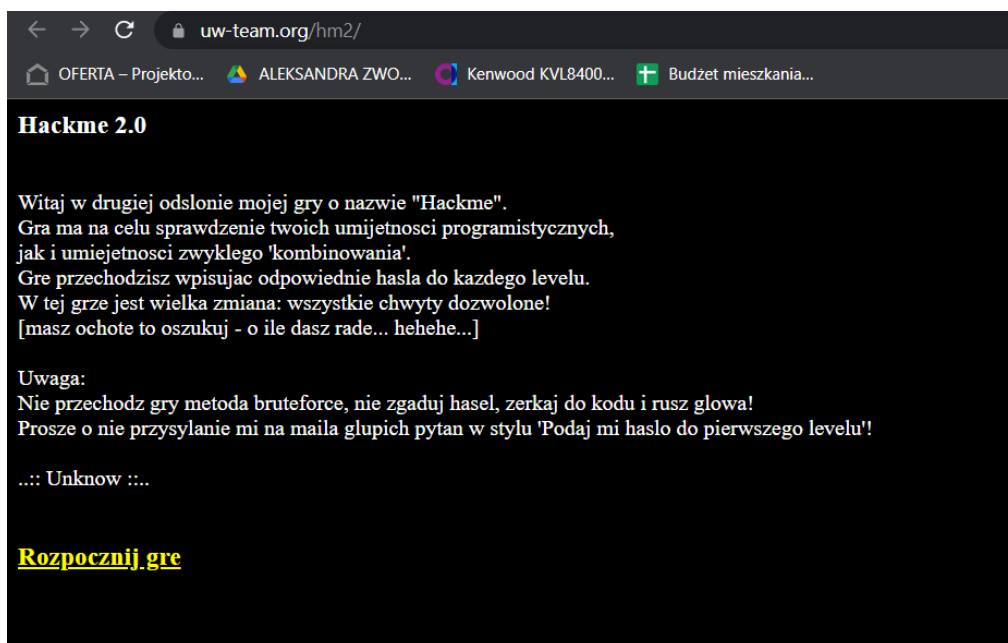
**You win!**

Gratulacje!  
Właśnie przeszedłeś grę Hackme 1.0 by Unknow!  
Gratuluje cierpliwości :)

Rysunek 2.8.6. Komunikat powiadamiający o ukończeniu gry HACKME 1.0

## 3. Druga gra programistyczna

Kolejna odsłona gry już nie skupia się tak mocno na samych aspektach programowania oraz dobrego rozumienia prostych linijek kodu.



Rysunek 3.1. Strona główna drugiej gry HACKME

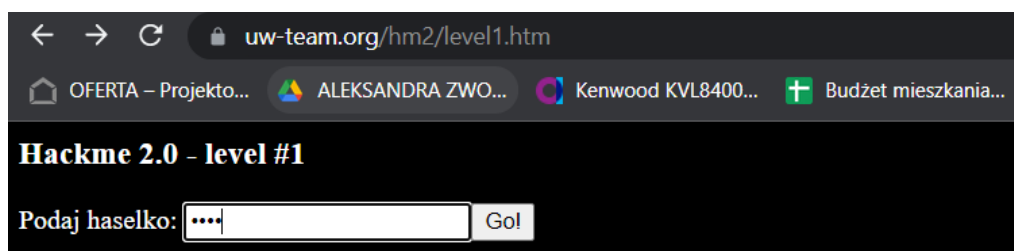
### 3.1. Rozwiązanie pierwszego poziomu

Na pierwszym poziomie drugiej gry mamy prostą *funkcję spr()* uruchamianą po naciśnięciu przycisku „Go!”. Sprawdza ona warunek czy wartość wpisana przez użytkownika jest równa wartości obiektu, którego id to „formularz”.

```
<html>
  <head>...</head>
  <body text="white" bgcolor="black" link="yellow" vlink="yellow" alink="yellow"> == $0
    <script>
      function spr(){
        if (document.getElementById('formularz').value==document.getElementById('haslo').value){
          self.location.href=document.getElementById('haslo').value+'.htm'; } else {alert('Nie, to
          nie to haselko :(');}
        }
      }
    </script>
    <h3>Hackme 2.0 - level #1</h3>
    " Podaj haselko: "
    <input type="password" name="haslo" id="haslo">
    <input value="text" name="formularz" id="formularz" type="hidden">
    <input type="button" onclick="spr()" value="Go!">
  </body>
</html>
```

Rysunek 3.1.1. Kod źródłowy pierwszego poziomu gry HACKME 2.0

Jak możemy sprawdzić obiekt o nazwie „*formularz*” i takim samym id ma wartość „*text*” i jest typu ukrytego.



Rysunek 3.1.2. Rozwiązanie pierwszego poziomu gry HACKME 2.0

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: *text*. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <http://www.uw-team.org/hm2/text.htm>, gdzie możemy rozwiązać kolejny poziom gry.

## 3.2. Rozwiązanie drugiego poziomu

W kolejnym poziomie mamy bardzo podobną funkcję *spr()* do poprzedniego poziomu. Tym razem poprawne hasło jest reprezentowane przez wartość zwracaną przez funkcję *unescape()*. Funkcja *unescape()* oblicza nowy ciąg, w którym szesnastkowe sekwencje są zastępowane znakiem, który reprezentują. Oznacza to, że wartość podaną jako argument tej funkcji musimy *zamienić z systemu szesnastkowego na odpowiadającemu mu znakowi ASCII*.

```
<html>
  <head>...</head>
  <body text="white" bgcolor="black" link="yellow" vlink="yellow" alink="yellow"> == $0
    <script>
      function spr(){
        if (document.getElementById('haslo').value==unescape('%62%61%6E%61%6C%6E%65')) {
          self.location=document.getElementById('haslo').value+'.htm'; } else { alert('Zle
          haslo!'); }
        }
      </script>
      <h3>Hackme 2.0 - level #2</h3>
      " Podaj haslo: "
      <input type="password" name="haslo" id="haslo">
      <input type="button" onclick="spr()" value="Break me!">
    </body>
  </html>
```

Rysunek 3.2.1. Kod źródłowy pierwszego poziomu gry HACKME 2.0

Poszczególne znaki są między sobą oddzielone przez znak „%”. Zamiana znaków została zaprezentowana w tabeli poniżej.

Zmienna HEX na ASCII							
HEX	62	61	6E	61	6C	6E	65
ASCII	b	a	n	a	l	n	e

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: *banalne*. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <http://www.uw-team.org/hm2/banalne.htm>, gdzie możemy rozwiązać kolejny poziom gry.

### 3.3. Rozwiązanie trzeciego poziomu

Na tym poziomie *wartość hasła została zapisana w systemie binarnym*, a wartość którą podaje użytkownik jest przekształcana z systemu dziesiętnego właśnie na binarny w celu sprawdzenia poprawności warunku *if*. Oznacza to, że aby uzyskać hasło dostępu musimy zmienić podaną wartość z systemu binarnego na dziesiętny.

```
<html>
  <head>...</head>
  <body text="white" bgcolor="black" link="yellow" vlink="yellow" alink="yellow"> == $0
    <script>
      function binary(liczba) {
        return liczba.toString(2);
      }
      function spr(){
        if (binary(parseInt(document.getElementById('haslo').value))=="10011010010") {
          self.location=document.getElementById('haslo').value+'.htm'; } else { alert('Zle!
          \nPodstawy matematyki sie klaniaja :)');}
        }
      }
    </script>
    <h3>Hackme 2.0 - level #3</h3>
    " Podaj haselko: "
    <input type="password" name="haslo" id="haslo">
    <input type="button" onclick="spr()" value="Click me baby!">
```

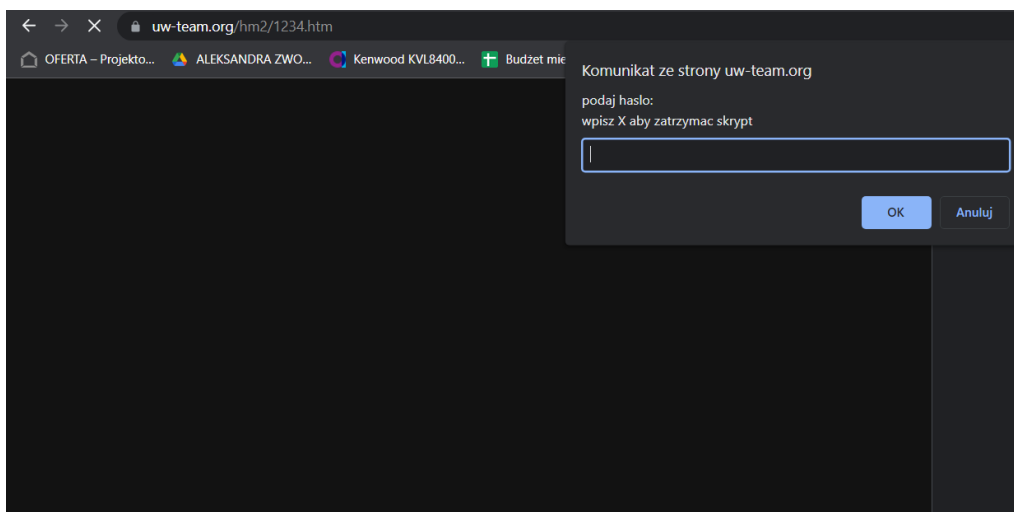
Rysunek 3.3.1. Kod źródłowy trzeciego poziomu gry HACKME 2.0

Zmienna systemu 2 na 10											
potęga	10	9	8	7	6	5	4	3	2	1	0
liczba	1	0	0	1	1	0	1	0	0	1	0
$1024 + 0 + 0 + 128 + 64 + 0 + 16 + 0 + 0 + 2 + 0 = 1234$											

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: *1234*. Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <http://www.uw-team.org/hm2/1234.htm>, gdzie możemy rozwiązać kolejny poziom gry.

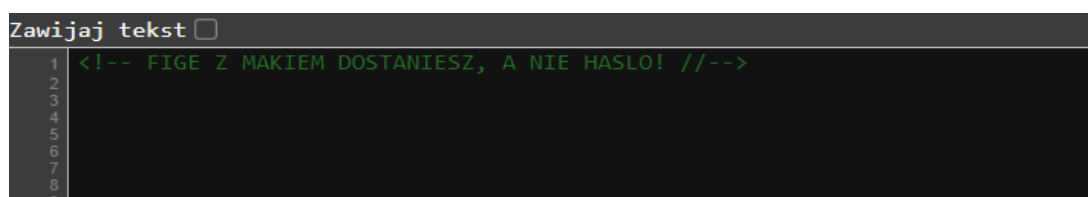
### 3.4. Rozwiązanie czwartego poziomu

Kolejny poziom był niestandardowy. Użytkownik musiał podać hasło, nie jak do tej pory na stworzonej stronie, tylko w okienku z wyświetlonym przez stronę komunikatem.



**Rysunek 3.4.1** Czwarty poziom gry HACKME 2.0

Aby wyświetlić kod należało skorzystać ze skrótu klawiszowego **CTRL + U**. Na stronie z kodem źródłowym, na pierwszy rzut oka nic się nie znajduje. Mamy jedną linijkę kodu i dużą ilość pustych linii. Jednak kiedy zjedziemy na sam dół strony znajdujemy kod z rozwiązaniem. Mamy podaną zmienną `cos` do której przypisana jest wartość funkcji `unescape()`. Z linii kodu poniżej możemy wyczytać, że użytkownik musi podać tę wartość po przeliczeniu na system szesnastkowy.



**Rysunek 3.4.2.** Kod źródłowy czwartego poziomu gry HACKME 2.0 cz.1



**Rysunek 3.4.3.** Kod źródłowy czwartego poziomu gry HACKME 2.0 cz.2

Zmienna HEX na ASCII			
HEX	32	35	38
ASCII	2	5	8

Po poznaniu wartości jaka kryła się pod funkcją `unescape()`, musimy uzyskaną wartość zamienić na wartość podaną w systemie szesnastkowym. Zamiana liczby 258 na podany system została przedstawiona w tabeli poniżej.

Zmienna systemu 10 na 16 liczby 256			
potęga	2	1	0
liczba	1	0	2
$256 + 0 + 2 = 258$			

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: [102](#). Po zatwierdzeniu hasła jesteśmy przekierowani na stronę o adresie: <http://www.uw-team.org/hm2/102.php>, gdzie możemy rozwiązać kolejny poziom gry.

### 3.5. Rozwiązanie piątego poziomu

Ten poziom jest dość nietypowy i będzie miał nietypowe rozwiązanie. Mianowicie, podczas rozwiązywania tego poziomu nie mamy dostępu do kodu źródłowego, ponieważ skrypt napisano w PHP. Autor udostępnił nam część kodu, jako wskazówkę do rozwiązania poziomu..

```

if (!isset($haslo)) {$haslo='';}
if (!isset($login)) {$login='';}
if ($haslo=="tu jest haslo") {$has=1;}
if ($login=="tu jest login") {$log=1;}
if (($has==1) && ($log==1)) { laduj nastepny level } else { powroc do tej strony }

```

Rysunek 3.5.1. Piąty poziom gry HACKME 2.0

Analizując zawarty kod widzimy, że jeśli istnieje zmienna *hasło* to jej wartość jest równa '' (czyli zmienna jest pusta) i to samo tyczy się zmiennej typu *login*, co widzimy liniijkę poniżej. Kolejne linijki kodu oznaczają, że jeśli wartość zmiennej *hasło* jest równa stringowi „*tu jest hasło*” to do zmiennej *has* zostanie przypisana wartość 1. Na podobnej zasadzie operujemy na zmiennej *login*. Ostanía linijka kodu, mówi nam, że jeśli wartość zmiennej *has* oraz *log* są równe 1 jednocześnie, to zostanie załadowany kolejny poziom gry.

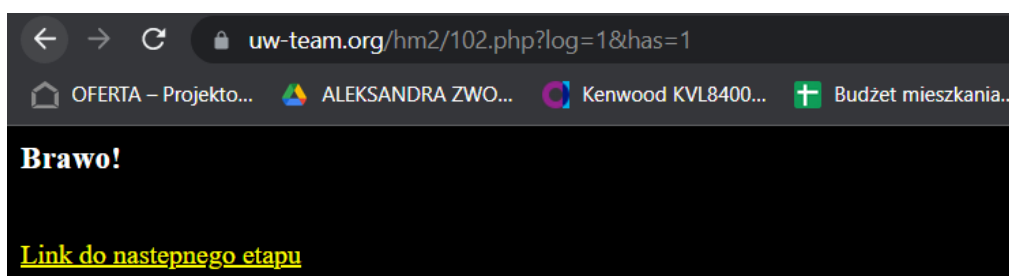




Rysunek 3.5.2 Błędne hasło i login – piąty poziom gry HACKME 2.0



Rysunek 3.5.3 Zmiana linku – rozwiązanie piątego poziomu gry HACKME 2.0



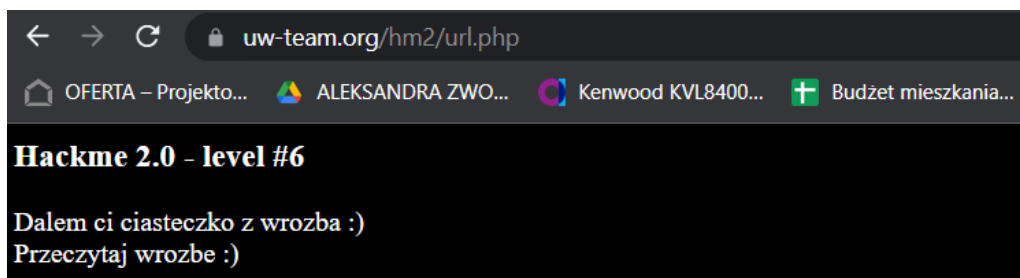
Rysunek 3.5.4 Ukończenie piątego poziomu gry HACKME 2.0

Podpowiedź kryje się w ostatniej linijce udostępnionego kodu. Do pól na login i hasło została przeze mnie wpisana wartość jeden. Po kliknięciu przycisku zaloguj, dostaliśmy komunikat, że zestaw jest niepoprawny. Warto jednak zwrócić uwagę, że zmienił się adres strony na którym się znajdowaliśmy.

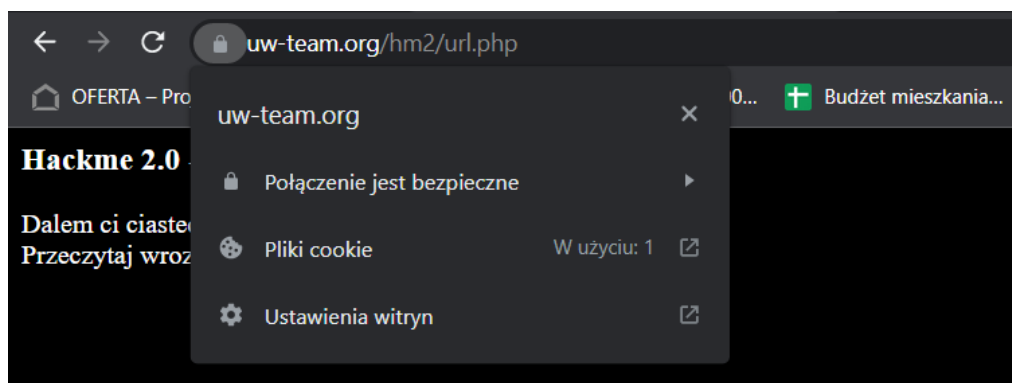
W tym momencie możemy skorzystać z podpowiedzi w ostatniej linijce i zmienić adres z linku: <http://www.uw-team.org/hm2/102.php?login=1&haslo=1> na link: <http://www.uw-team.org/hm2/102.php?log=1&has=1>. Co pozwoliło ukończyć piaty poziom.

## 3.6. Rozwiązanie szóstego poziomu

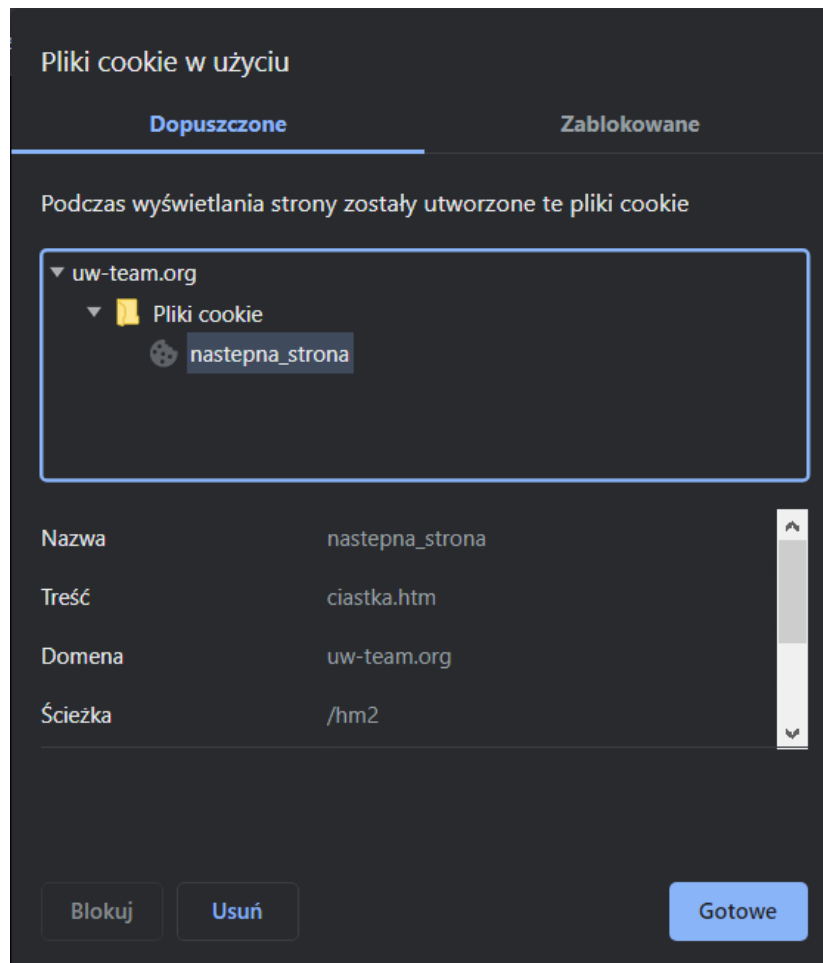
Kolejny nietypowy poziom. Po wyświetleniu jego zawartości już dostajemy podpowiedź. Ponieważ powiedzenie „Dałem ci ciasteczko z wroźbą” odnosi się właśnie do ciasteczek czyli *plików cookie*. Po zajrzeniu co się tam znajduje, widzimy plik o nazwie: *następna\_strona* o treści „*ciastka.htm*”. To właśnie jest naszym rozwiązaniem. Należy zmienić adres strony na <http://www.uw-team.org/hm2/ciastka.htm>



Rysunek 3.6.1 Szósty poziom gry HACKME 2.0



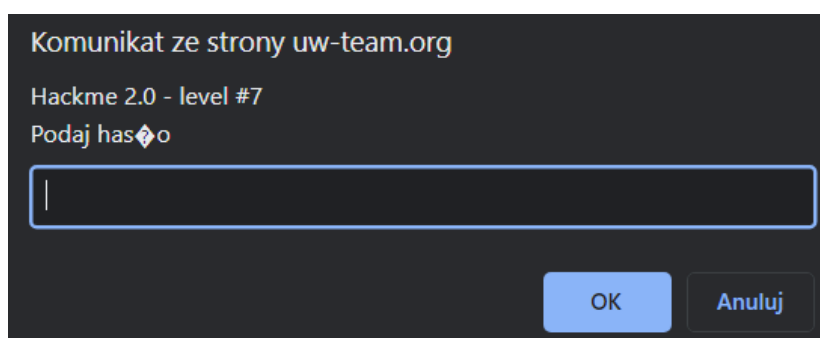
Rysunek 3.6.2 Ciasteczka dołączone do szóstego poziomu gry HACKME 2.0 cz. 1



Rysunek 3.6.3 Ciasteczka dołączone do szóstego poziomu gry HACKME 2.0 cz. 2

### 3.7. Rozwiązanie siódmego poziomu

Kolejny poziom ponownie opiera się na komunikacie, który dostajemy ze strony. W celu znalezienia hasła ponownie korzystamy ze skrótu klawiszowego **CTRL + U**.

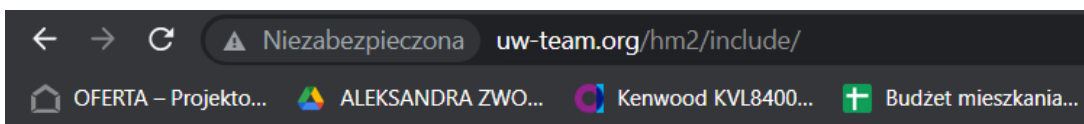


Rysunek 3.7.1. Poziom siódmy gry HACKME 2.0



W kodzie źródłowym w piątej linii kodu mamy wyraźne wskazanie hasła. Skrypt odwołuje się do pliku, który znajduje się w `/include`, a samo hasło jakie musimy podać jest dokładną nazwą tego pliku o rozszerzeniu `.js`. Aby uzyskać rozwiązanie warto wykorzystać funkcję serwera *apache*, która pokazuje listing plików w folderze (Index of ...). Już po wejściu do `/include` ukazuje nam się plik *cosik.js*, co oznacza że jako hasło należy wpisać: *cosik*.

```
Zawijaj tekst ☐
1 <html><head><title>Hackme 2.0 - by Unknow</title></head><body text="white" bgcolor="black"
2 <script>
3 strona='zle.htm';
4 haslo=prompt("Hackme 2.0 - level #7 \nPodaj hasło", "")
5 document.write('<script type="text/javascript" src="include/'+haslo+'.js"></script>');
6 onload=function(){
7 if(haslo==null) { self.location='http://www.uw-team.org/' } else location.href=strona; }
8 </script>
9 </body></html>
```

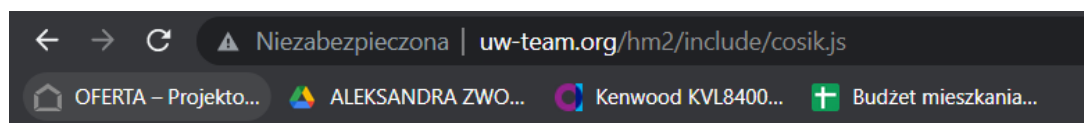
Rysunek 3.7.2. Kod siódmego poziomu gry HACKME 2.0



## Index of /hm2/include

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">cosik.js</a>	2008-11-19 16:39	21	

Rysunek 3.7.3 Zawartość katalogu /include

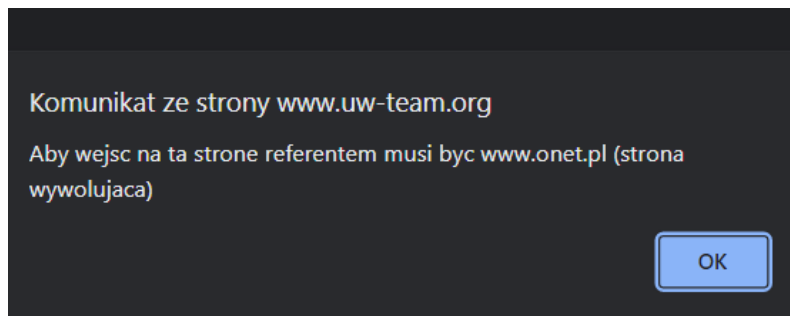


```
strona='listing.php';
```

Rysunek 3.7.4 Zawartość pliku cosik.js

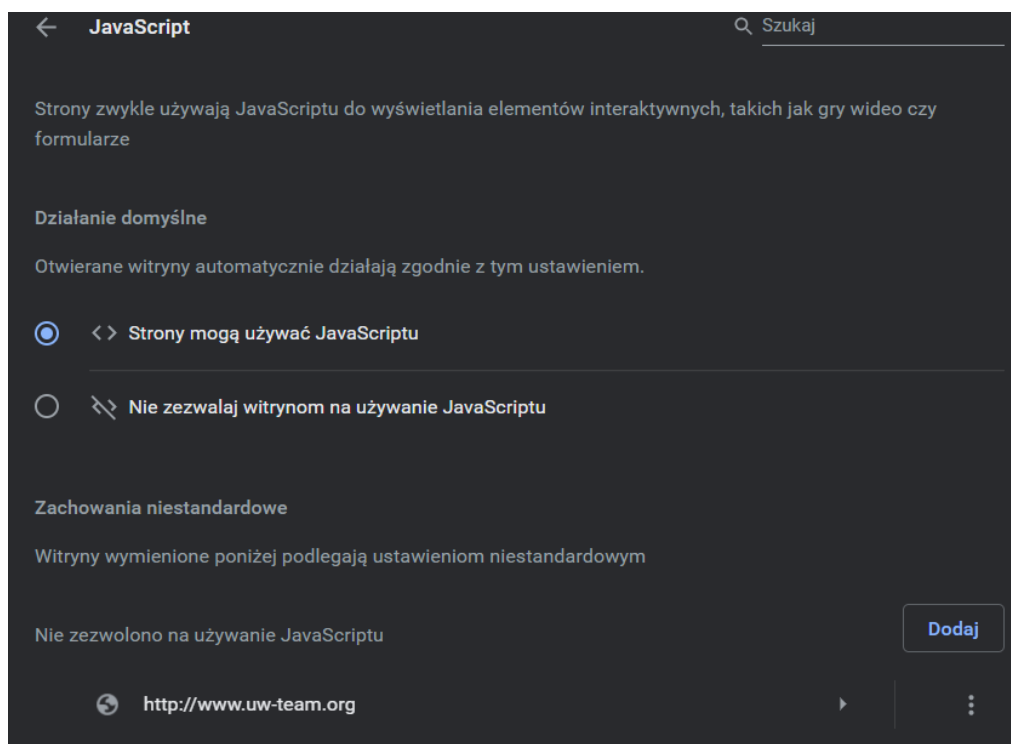
## 3.8. Rozwiązanie ósmego poziomu

Kolejny poziom podobnie jak poprzedni jest w postaci wyświetlonego komunikatu. Tym razem jednak nie mamy nigdzie do podania hasła. Tak samo jak poprzednio możemy użyć skrótu klawiszowego do wyświetlenia kodu źródłowego.

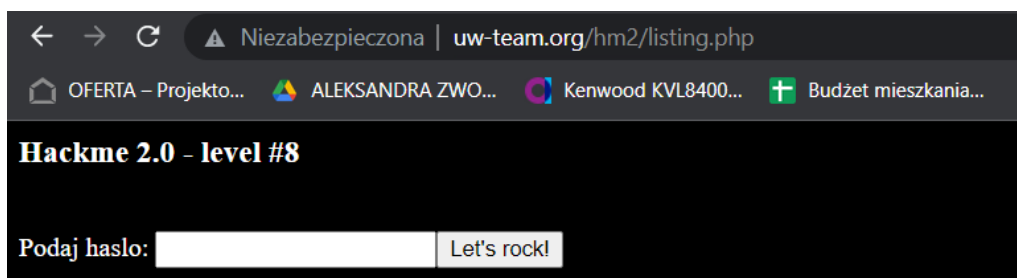


**Rysunek 3.8.1. Poziom ósmy gry HACKME 2.0**

Jednak żeby mieć gdzie wpisać hasło skorzystam z drugiej opcji, którą można było zastosować również w poprzednich przykładach czyli wyłączyć działanie JavaScript.



**Rysunek 3.8.2. Ustawienia – brak pozwolenia stronie z grą na używanie JavaScriptu**



**Rysunek 3.8.3 Wyświetlenie poziomu ósmego gry HACKME 2.0**

Po wyświetleniu kodu źródłowego, ukazuje nam się blok kodu z formatowaniem tekstu. Odczytując tekst z góry na dół, dostajemy komunikat: „*hasłem do tego etapu jest słowo kxnxgxnxaxa*”.

```

▼ <div id="ukryte" style="display:none">
  <font color="black">h</font>
  <font color="black">a</font>
  <font color="black">s</font>
  <font color="black">l</font>
  <font color="black">e</font>
  <font color="black">m</font>
  <font color="black"> d</font>
  <font color="black">o</font>
  <font color="black"> t</font>
  <font color="black">e</font>
  <font color="black">g</font>
  <font color="black">o </font>
  <font color="black">e</font>
  <font color="black">t</font>
  <font color="black">a</font>
  <font color="black">p</font>
  <font color="black">u</font>
  <font color="black"> j</font>
  <font color="black">e</font>
  <font color="black">s</font>
  <font color="black">t</font> == $0
  <font color="black"> s</font>
  <font color="black">l</font>
  <font color="black">o</font>
  <font color="black">w</font>
  <font color="black">o </font>
  <font color="black">k</font>
  <font color="black">x</font>
  <font color="black">n</font>
  <font color="black">x</font>
  <font color="black">g</font>
  <font color="black">x</font>
  <font color="black">n</font>
  <font color="black">x</font>
  <font color="black">a</font>

```

Rysunek 3.8.4 Kod poziomu ósmego gry HACKME 2.0

Zatem poprawnym hasłem jakie należało wpisać, aby przejść na kolejny poziom było: [kxnxgxnxaxa](http://www.uw-team.org/hm2/pokaz.php). Po zatwierdzeniu hasła musimy przejść na stronę o adresie: <http://www.uw-team.org/hm2/pokaz.php>, gdzie rozwiążemy kolejny etap gry.

← → ↻ Niezabezpieczona | uw-team.org/hm2/listing.php?haslo=kxnxgxnxaxa

OFERTA – Projekto... ALEKSANDRA ZWO... Kenwood KVL8400... Budżet mieszkania...

Następny etapik ukryty jest w pliku pokaz.php

**Hackme 2.0 - level #8**

Podaj hasło:

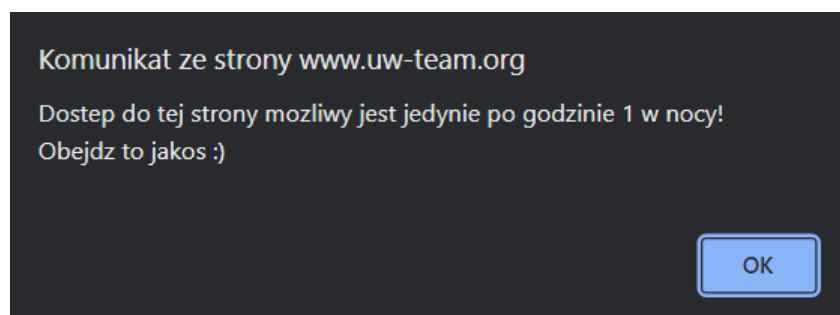
Rysunek 3.8.5 Rozwiązanie poziomu ósmego gry HACKME 2.0

## 3.9. Rozwiązanie dziewiątego poziomu

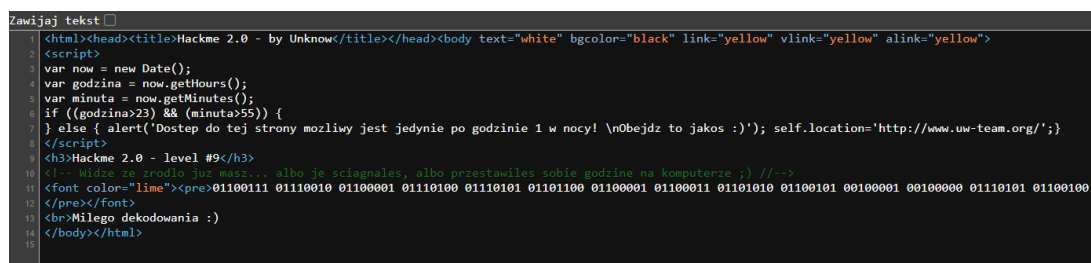
Po przejściu na wskazany wcześniej adres strony dostajemy komunikat, że dostęp do strony jest możliwy tylko po pierwszej w nocy. Są dwa możliwe scenariusze aby rozwiązać ten scenariusz.

Pierwszym jest wyświetlenie kodu źródłowego przy pomocy skrótu klawiszowego **CTRL + U**. Po wyświetleniu kodu i przeanalizowaniu działania skryptu, możemy po prostu **zmienić godzinę w komputerze** i wtedy wejść na stronę.

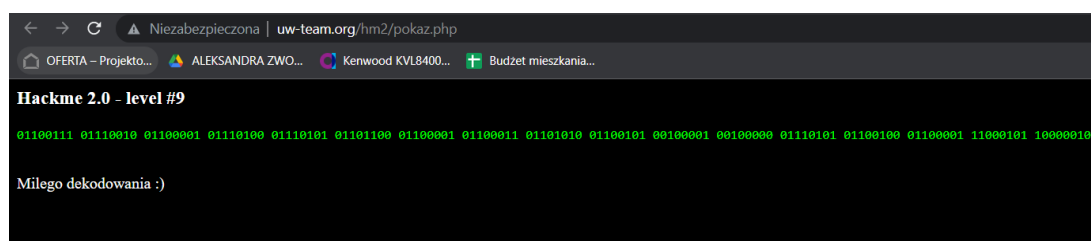
Drugą opcją było **wyłączenie możliwości uruchomienia JavaScriptu**. Powoduje to, że przechodząc na tą stronę mamy bezpośrednio wyświetloną jej zawartość.



Rysunek 3.9.1 Poziom dziewiąty gry HACKME 2.0



Rysunek 3.9.2 Kod źródłowy strony dla dziewiątego poziomu HACKME 2.0



Rysunek 3.9.3 Zawartość dziewiątego etapu gry HACKME 2.0

Po zobaczeniu treści kolejnego zadania, najprostszy sposobem będzie skorzystanie z programu online, który w szybki sposób pomoże nam odszyfrować wiadomość.

### Binary to Text Translator

Enter binary numbers with any prefix / postfix / delimiter and press the *Convert* button  
(E.g: 01000101 01111000 01100001 01101101 01110000 01101100 01100101):

Open File

Open Binary File

Paste binary numbers or drop file:

```
00100000 01000011 01101001 00100000 01110011 01101001
11000100 10011001 00100000 01110101 01101011 01101111
11000101 10000100 01100011 01111010 01111001 11000100
10000111 00100000 01110100 01100101 00100000 01110111
01100101 01110010 01110011 01101010 01100101 00100000
01001000 01100001 01100011 01101011 01101101 01100101
00101110
```

Character encoding (optional)

ASCII/UTF-8

Convert

Reset

Swap

gratulacje! udało Ci się ukończyć tę wersję Hackme.

**Rysunek 3.9.4. Odszyfrowanie ciągu binarnego**



## Spis ilustracji

Rysunek 2.1. Strona główna pierwszej gry HACKME.....	3
Rysunek 2.1.1. Kod źródłowy pierwszego poziomu gry HACKME 1.0.....	4
Rysunek 2.1.2. Rozwiązanie pierwszego poziomu gry HACKME 1.0.....	5
Rysunek 2.2.1. Kod źródłowy drugiego poziomu gry HACKME 1.0.....	6
Rysunek 2.2.2. Dodatkowa zawartość dodana do drugiego poziomu gry HACKME 1.0.....	6
Rysunek 2.2.3. Rozwiązanie drugiego poziomu gry HACKME 1.0.....	6
Rysunek 2.3.1. Kod źródłowy trzeciego poziomu gry HACKME 1.0.....	7
Rysunek 2.3.2. Rozwiązanie trzeciego poziomu gry HACKME 1.0.....	8
Rysunek 2.4.1 Kod źródłowy czwartego poziomu gry HACKME 1.0.....	9
Rysunek 2.4.2. Rozwiązanie czwartego poziomu gry HACKME 1.0.....	10
Rysunek 2.5.1 Kod źródłowy piątego poziomu gry HACKME 1.0.....	10
Rysunek 2.5.2. Pomocniczy skrypt napisany w C++.....	11
Rysunek 2.5.3 Rozwiązanie piątego poziomu gry HACKME 1.0.....	12
Rysunek 2.6.1 Kod źródłowy szóstego poziomu gry HACKME 1.....	13
Rysunek 2.6.2 Rozwiązanie szóstego poziomu gry HACKME 1.0.....	14
Rysunek 2.7.1 Kod źródłowy siódmego poziomu gry HACKME 1.0.....	15
Rysunek 2.7.2. Rozwiązanie siódmego poziomu gry HACKME 1.0.....	16
Rysunek 2.8.1 Kod źródłowy ósmego poziomu gry HACKME 1.0.....	17
Rysunek 2.8.2. Dodatkowa zawartość dołączona do ósmego poziomu gry HACKME 1.0.....	17
Rysunek 2.8.3. Kod wyświetlony po zbadaniu elementu na stronie.....	18
Rysunek 2.8.4. Ukryta zawartość dodana do ósmego poziomu gry HACKME 1.0.....	18
Rysunek 2.8.5. Rozwiązanie ósmego poziomu gry HACKME 1.0.....	19
Rysunek 2.8.6. Komunikat powiadamiający o ukończeniu gry HACKME 1.0.....	19
Rysunek 3.1. Strona główna drugiej gry HACKME.....	20
Rysunek 3.1.1. Kod źródłowy pierwszego poziomu gry HACKME 2.0.....	20
Rysunek 3.1.2. Rozwiązanie pierwszego poziomu gry HACKME 2.0.....	21
Rysunek 3.2.1. Kod źródłowy pierwszego poziomu gry HACKME 2.0.....	21
Rysunek 3.3.1. Kod źródłowy trzeciego poziomu gry HACKME 2.0.....	22
Rysunek 3.4.1 Czwarty poziom gry HACKME 2.0.....	23
Rysunek 3.4.2. Kod źródłowy czwartego poziomu gry HACKME 2.0 cz.1.....	23
Rysunek 3.4.3. Kod źródłowy czwartego poziomu gry HACKME 2.0 cz.2.....	23
Rysunek 3.5.1. Piąty poziom gry HACKME 2.0.....	24
Rysunek 3.5.2 Błędne hasło i login – piąty poziom gry HACKME 2.0.....	25
Rysunek 3.5.3 Zmiana linku – rozwiązanie piątego poziomu gry HACKME 2.0.....	25
Rysunek 3.5.4 Ukończenie piątego poziomu gry HACKME 2.0.....	25
Rysunek 3.6.1 Szósty poziom gry HACKME 2.0.....	26
Rysunek 3.6.2 Ciasteczka dołączone do szóstego poziomu gry HACKME 2.0 cz. 1.....	26
Rysunek 3.6.3 Ciasteczka dołączone do szóstego poziomu gry HACKME 2.0 cz. 2.....	27
Rysunek 3.7.1. Poziom siódmy gry HACKME 2.0.....	27
Rysunek 3.7.2. Kod siódmego poziomu gry HACKME 2.0.....	28
Rysunek 3.7.3 Zawartość katalogu /include.....	28

<i>Rysunek 3.7.4 Zawartość pliku cosik.js.....</i>	<i>28</i>
<i>Rysunek 3.8.1. Poziom ósmy gry HACKME 2.0.....</i>	<i>29</i>
<i>Rysunek 3.8.2. Ustawienia – brak pozwolenia stronie z grą na używanie JavaScriptu .....</i>	<i>29</i>
<i>Rysunek 3.8.3 Wyświetlenie poziomu ósmego gry HACKME 2.0.....</i>	<i>29</i>
<i>Rysunek 3.8.4 Kod poziomu ósmego gry HACKME 2.0.....</i>	<i>30</i>
<i>Rysunek 3.8.5 Rozwiązanie poziomu ósmego gry HACKME 2.0.....</i>	<i>30</i>
<i>Rysunek 3.9.1 Poziom dziewiąty gry HACKME 2.0 .....</i>	<i>31</i>
<i>Rysunek 3.9.2 Kod źródłowy strony dla dziewiątego poziomu HACKME 2.0.....</i>	<i>31</i>
<i>Rysunek 3.9.3 Zawartość dziewiątego etapu gry HACKME 2.0.....</i>	<i>31</i>
<i>Rysunek 3.9.4. Odszyfrowanie ciągu binarnego.....</i>	<i>32</i>