Bezpieczeństwo usług sieciowych
Laboratorium 3: ROZWAL.TO

Szymon Bagiński

15 grudnia 2018

1 Cel zadania

Celem zadania było rozwiązywanie zadań typu "capture the flag" na stronie `https://stary.rozwal.to/`. Obowiązkowe były zadania z modułu **Crypto**. Niemniej jednak trzy zadania ("Zaloguj sie na konto admin.", "Nie kłam", "Mieszkam w bloku") były niemożliwe do rozwiązania z powodu błędów po stronie serwera.

Dodatkowo można było wykonać zadania typu SQL injection z innych modułów.

2 Crypto

2.1 Bob uwielbia xorować

Aby zdobyć flagę należało odkodować następujący tekst:
`GCg7Ozs7Oy01e3oNMz4gP3ogP3ovPjs2NXoZM3opMz96KDUgKSAjPCg1LTs5`
`ei4/MSkueiA7KSAjPCg1LTs0I3oqNTA/PiM0OSAjN3o4OzAuPzd0ehQ1ej41`
`OCg7dno8Njs9O3ouNWB6CBUADRsWBSEJMzQ9Nj8CNSgYIy4/GTMqMj8oJw==`

Z opisu zadania wiemy, że tekst został zakodowany poprzez operację xor z pojedynczym bajtem. Wiemy także, że flaga ma format ROZWAL_{...}. Możemy więc metodą przeglądu zupełnego spróbować zaszyfrować flagę każdym możliwym bajtem i sprawdzić czy znajduje się ona w szyfrogramie.

```
extern crate base64;

fn main() {
    let crypted_base_64 = String::from("
        GCg7Ozs7Oy01e3oNMz4gP3ogP3ovPjs2NXoZM3opMz96KDUgKSAjPCg1LTs5ei4/
        MSkueiA7KSAjPCg1LTs0I3oqNTA/
        PiM0OSAjN3o4OzAuPzd0ehQ1ej41OCg7dno8Njs9O3ouNWB6CBUADRsWBSEJMzQ9Nj8CNSgYIy4
        /GTMqMj8oJw==");

    let flaga = String::from("ROZWAL");
```

```
        for i in 0..256 {
            let byte: u8 = i as u8;
            let xored_flag: Vec<u8> = flaga.as_bytes().iter().map(|b| b ^ byte).
                collect();
            let flag_base64 = base64::encode(&xored_flag);
            if crypted_base_64.contains(&flag_base64) {
                let msg_decoded_bytes = base64::decode(&crypted_base_64).unwrap();
                let msg_decrypted_bytes: Vec<u8> = msg_decoded_bytes.iter().map(|b| b
                    ^ byte).collect();
                let msg_decoded_string = String::from_utf8(msg_decrypted_bytes).unwrap
                    ();
                println!("{:?}", msg_decoded_string);
                break;
            }
        }
    }
```

Powyższy program jest w stanie rozszyfrować wiadomość, która brzmi:
"Braaaaawo! Widze ze udalo Ci sie rozszyfrowac tekst zaszyfrowany pojedynczym bajtem. No dobra, flaga to: ROZWAL_{SingleXorByteCipher}"

2.2 Alice też xoruje

W tym zadaniu musimy zastosować inną metodę, ponieważ Alice także wykonuje operację xor, ale nie z pojedynczym bajtem. Wiemy, że gdzieś w tekście znajduje się zakodowana flaga ROZWAL_{...}. Możemy więc dla każdej możliwej pozycji flagi wykonać odwrotną operację xor, żeby zobaczyć jaki musiałby być klucz aby flaga była w danym miejscu. Jeśli kluczem jest jakieś istniejące słowo będziemy mogli nim spróbować odszyfrować całą wiadomość.

```
extern crate base64;
const crypted_base64: &str = "
    PAADREshCgcRDgZPBAoPSxgGBBEOAR0ABksVEUUeDw4YCksoBlQWAg5PBgoRGBUNAxkEGBUGSx8KGkUfDg
QHEUVLIgEWAgoDVAoFSw0NBksFBhEGBEsLGBARGBUNSUsKDQ1FBgQVGAwcDk8WHAcETwQXEQ4fBgocCgsOAAUC
ClQADQ4EABwcBQoTCksKGxUOHkVPOQoRDk8eABgRDA4ASx8dGwYDDk8EBA8PBhoCHksVEQcSSwEdAEsJFhgKSx
EOVBEZHgsaCkVLIB9JSw0DFQIKSxsbRTkkNSMkJzQUNQkCCAo9FiIGHwYAGBgKEBg=";
const flaga : &str = "ROZWAL";

fn main() {
    let cryptde_bytes = base64::decode(crypted_base64).unwrap();
    let flaga_bytes = flaga.as_bytes();
    let mut msg_decrypted_bytes: Vec<u8> = Vec::new();
    for i in 0..cryptde_bytes.len() - 5{
        let mut key: Vec<u8> = Vec::new();
        for j in 0..6 {
            key.push(cryptde_bytes[i + j] ^ flaga_bytes[j]);
        }
        let key_str = String::from_utf8(key).unwrap();
        println!("{}: {}", i, key_str);
    }
}
```

Powyższy program wypisuje dla każdej możliwej pozycji flagi, klucz jaki musiałby być użyty przy szyfrowaniu, aby flaga była właśnie w tym miejscu. Na wyjściu programu można znaleźć jedną interesującą linijkę:
`214: kkotek`

```
extern crate base64;

const crypted_base64: &str = "
    PAADREshCgcRDgZPBAoPSxgGBBEOAR0ABksVEUUeDw4YCksoBlQWAg5PBgoRG
BUNAxkEGBUGSx8KGkUfDgQHEUVLIgEWAgoDVAoFSw0NBksFBhEGBEsLGBARGBUNSU
sKDQ1FBgQVGAwcDk8WHAcETwQXEQ4fBgocCgsOAAUCClQADQ4EABwcBQoTCksKGxU
OHkVPOQoRDk8eABgRDA4ASx8dGwYDDk8EBA8PBhoCHksVEQcSSwEdAEsJFhgKSxEO
VBEZHgsaCkVLIB9JSw0DFQIKSxsbRTkkNSMkJzQUNQkCCAo9FiIGHwYAGBgKEBg=";

const flaga : &str = "ROZWAL";

fn main() {
    let cryptde_bytes = base64::decode(crypted_base64).unwrap();
    let flaga_bytes = flaga.as_bytes();
    let mut msg_decrypted_bytes: Vec<u8> = Vec::new();
    for i in 0..cryptde_bytes.len() {
        msg_decrypted_bytes.push(cryptde_bytes[i] ^ "kotek".as_bytes()[i % 5]);
    }
    println!("{}", String::from_utf8(msg_decrypted_bytes).unwrap());
}
```

Powyższy program próbuje odszyfrować wiadomość kluczem "kotek". Jego wyjściem okazało się rozwiązanie zadania:
"Wow! Jestem pod wrazeniem ze udalo Ci sie rozszyfrowac ten tekst. Musial on byc nieco dluzszy, aby mozliwe bylo przeprowadzenie efektywnego ataku. Moze jeszcze troche paddingu zeby nie bylo za trudno. Ok, flaga to ROZWAL_{AliceIsImpressed}"

2.2.1 Cweyk funcbjqlsiluqe - Szyfr podstawieniowy

W tym zadaniu należało rozszyfrować tekst w języku angielskim. Z uwagi na to, że tekst jest długi nie zostanie tutaj zaprezentowany. Można go znaleźć na stronie:

`http://training.securitum.com/rozwal/crypto/3.php`

Po konstrukcji tekstu można się zorientwać, że jest to szyfr podstawieniowy, tzn. każda litera jest zastąpiona jakąś inną według nieznanego nam słownika. Aby rozszyfrować tekst napisano program, który mapuje litery na inne. Dla wygody wszystkie litery zostały zmienione na wersaliki.

W pierwotnym tekście jest wyraźnie widoczne miejsce szukanej flagi("KUWQJG_{CpzCblbpbluiTlfdskLcQsjr}"
Zaczęto więc od podstawień liter z flagi, a dalej kierując się intuicją i wiedzą o konstrukcjach w języku angielskim powstał program ze słownikiem, zaprezentowany poniżej.

```
fn main() {
    let upper = String::from(TEXT).to_uppercase();
    let dict: HashMap<char, char> =
    [
        ('K', 'R'), ('U', 'O'), ('W', 'Z'), ('Q', 'W'), ('J', 'A'),
        ('G', 'L'), ('B', 'T'), ('C', 'S'), ('I', 'N'), ('N', 'D'),
        ('D', 'H'), ('S', 'E'), ('Y', 'F'), ('F', 'P'), ('P', 'U'),
        ('T', 'C'), ('O', 'V'), ('V', 'G'), ('E', 'Y'), ('R', 'K'),
        ('Z', 'B'), ('X', 'J'), ('L', 'I')
    ].iter().cloned().collect();

    let result: String = upper.chars().map(|character| {
        if dict.contains_key(&character) {
            return dict[&character];
        }
        else {
            return character;
        }
    }).collect();

    println!("{}", result);
}
```

Powyższy program z powodzeniem rozszyfrował tekst. Flaga w nim zawarta to:
`ROZWAL_{SubStitutionCipherIsWeak}`

2.3 Znajdź kolizję

Ze źródła strony tego zadania możemy wywnioskować, że flaga zostanie wypisana, gdy funkcja **h** zwróci wartości równe dla parametrów **s1** oraz **s2**. Parametry te można przekazać w url. W funkcji **h** widoczna jest konkatenacja podanego parametru do ciągu znaków, a następnie wyliczany jest z tej wartości skrót md5. Podane wartości nie mogą być jednak identyczne, ponieważ w takim przypadku nie dojdzie w ogóle do sprawdzania wartości powrotnych funkcji **h**. Musimy więc podać takie parametry, które nie są sobie równe, ale ich konwersja do typu string ma tę samą wartość. Można więc zastosować w tym miejscu tablice z innymi wartościami. Nie są sobie równe, ale rzutowanie do stringa tablicy w php daje zawsze słowo Ąrray". Aby dostać flagę należy więc wejść na przykład w link:

`http://training.securitum.com/rozwal/crypto/4.php?s1[]=a&s2[]=b`

Flaga: ROZWAL_{ItsYourBirthday!}

3 SQL injection

3.1 Gimnazjum SQL - Proste

Na stronie widoczne jest następujące zapytanie:

```
query('SELECT name FROM flags WHERE id='.$_GET['id']);
while($r=$res->fetchArray()) { echo $r['name']; }
```

To co podamy w parametrze **id** będzie dodane na koniec zapytania SQL, do warunku w konstrukcji WHERE. Ważne jest, żeby wyrażenie po słowie WHERE było prawdziwe. Możemy więc dopisać alternatywę logiczną, która zawsze będzie prawdziwa. Możemy więc stworzyć na przykład taki url:

```
http://training.securitum.com/rozwal/gim/gimnazjum1/?id=1%20or%201%20=%201
```

Zawiera on w sobie warunek `id=1 or 1 = 1`, który zwsze jest prawdziwy.
Otrzymana flaga: `ROZWAL_{ZjazdGimboli}`

3.2 Gimnazjum SQL - Nie takie trudne

To zadanie można rozwiązać w podobny sposób jak zadanie "Proste". Zapytanie SQL z tego zadania ogranicza jednak ilość zwracanych rekordów do jednego. Możemy więc wyświetlać tylko po jednym rekordzie z bazy na raz, ale nie wiemy jakie id ma nowa flaga. Można jednak spowodować, że instrukcja LIMIT nie będzie zawarta w zapytaniu poprzez dodanie średnika do takiej samej wartości id jak w zadaniu "Proste".

```
http://training.securitum.com/rozwal/gim/gimnazjum2/?id=1%20or%201%20=%201;
```

Nowa flaga: `ROZWAL_{OdZjazduGimboliGlowaBoli}`

3.3 Gimnazjum SQL - Troche cięższej

Nie wiem dlaczego to zadanie nazywa się "trochę cięższej", skoro można je rozwiązać w dokładnie taki sam sposób jak zadanie "nie takie trudne".

```
http://training.securitum.com/rozwal/gim/gimnazjum3/?id=1%20or%201%20=%201;
```

Nowa flaga: `ROZWAL_{SelectNameFromGimbus}`