Udskrift sponsoreret af: Privatlivspolitik

BLOG | Kim Tiedemann

EMNER

Aften 2 - Hack 2: Cross site scripting på en portal

Af Kim Bjørn Tiedemann 8. juli 2014 kl. 11:35

Jeg har tidligere (http://www.version2.dk/blog/hvordan-jeg-paa-aftener-hackede-store-offentlige-sites-67618) skrevet, hvordan jeg på to aftener fandt to store sikkerhedshuller på to offentlige sites.

Det første site lækkede authentication cookien over http og dermed mulighed for, at en Man-In-The-Middle kunne aflæse cookien og overtage brugerens identitet.

Det andet site kan udsættes for et Cross site scripting angreb.

Cross site scripting (XSS)

Det går i al sin enkelthed ud på, at man ved at manipulere med input kan indsætte JavaScript på et site og dermed kan eksekvere JavaScript kode på sitet. Problemet skyldes at input parametre til en webside ikke saniteres og kontrolleres og dermed kan hackeren forsøge at manipulere parametrene til sitet og dermed prøve at få injectet noget Javacript.

Der er to typer cross site scripting: Reflected Cross Site Scripting hvor angrebet typisk udføres ved at sende links eller lignende til ofrene, som så ved at klikke på linket får aktiveret angrebet. Et stored Cross Site Scripting derimod er mere alvorligt, da serveren i modsætning til reflected gemmer angriberens data i databasen og dermed bliver langt flere brugere udsat for angrebet.

De første forsøg

Hvis et site har en søgefunktionalitet så er det et godt udgangspunkt at forsøge at sætte at reflected XSS angreb ind først. Det gjorde sig også gældende på dette site. Søgefunktionaliteten er implementeret ved, at der kaldes en url med en query parameter "SearchTerm", som indeholder søgeordet.

Jeg forsøger først at søge på

```
<i>rubbish</i>
```

. Dette medfører en fejl på sitet. Jeg prøver lidt forskellige kombinationer (med eller uden <, / og >) og alle medfører en fejl på sitet.

Herefter forsøger jeg bare at søge på rubbish. Det giver en fejlfri søgning. Jeg prøver herefter at se sidens kildekode og søger på rubbish, for at finde de steder på siden, hvor søgetermen benyttes.

Og voila: Den benyttes i noget JavaScript kode:

```
1  -- Some JavaScript code
2  var searchTerm = decode('rubbish');
3  -- More JavaScript code
```

Nu har vi noget

Så kunne jeg bruge et søgekriterie, hvor jeg injectede noget JavaScript kode? Og ville det have en effekt. Ja for længere nede i JavaScript koden blev searchTerm skrevet ud i et html element:

```
1  function Anonymous() {
2      $('#div').val(searchTerm);
3      $('#form').submit();
4  }
```

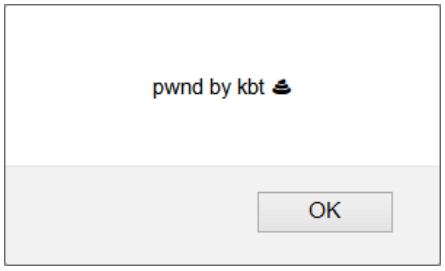
Nu skulle jeg bare finde ud af, hvordan jeg kunne injecte JavaScript ind i variablen searchTerm. Da indholdet af variablen sættes server-side og jeg kan se, at den udskrives html encodet, så kan jeg jo bare indsætte ved at JavaScript encode mit JavaScript:

```
SearchTerm=rubbish\x3cscript\x3e%20alert(\x27pwnd\x20by\x20kbt\x20\u{1F4A9}\x27)%20\x3c/script\x3e
```

Ovenstående bliver i JavaScript til

```
rubbish<script>alert('pwnd by kbt [pile_of_poo]')</script>
```

Og det skrives ud i div tagget og dermed bliver det eksekveret af browseren. Og resultatet er en alert boks med indholdet pwnd by kbt [pile_of_poo] (jeg kan desværre ikke indsætte unicode karakterer i blogposten).



Hvad kan det bruges til

Man kunne jo sende et link til en masse potentielle brugere af sitet og håbe at de klikker på det. Hvis de samtidigt er logget ind på sitet, kan man altså udføre en masse handlinger på deres vegne. Hvis det havde været en webbank kunne man instruere browseren i at submitte en formular, der fx overførte penge. Hvis det havde været en webbaseret email løsning kunne man jo slette alle personens emails eller overtage kontoen.

Et af de mest grelle eksempler på XSS er Samy MySpace worm (http://en.wikipedia.org/wiki/Samy_(computer_worm)) . Her brugte en ung mand XSS til at lave en orm, der sendte en masse venneforespørgsler af sted på det sociale site MySpace.

Hvordan kan man sikre sig?

Angrebet kan udføres fordi udviklerne af sitet ikke har været opmærksomme nok på at validere input parametre samt at sørge for den korrekte encoding af input parametre, som udskrives igen. I dette tilfælde havde udviklerne sikret, at man ikke kunne bruge < tegnet i søgestrengen. De havde også sørget for at html encode søgetermen inden den blev skrevet ud i "du har søgt på rubbish", men de havde glemt at JavaScript encoding er noget helt andet end html encoding.

Man kan bruge de forskellige frameworks måder at encode på. Fx understøtter ASP.NET MVC automatisk html encoding, så når en variabel skrives ud i html sikrer frameworket, at der automatisk encodes og dermed umuligt at injecte fx scripts. ASP.NET MVC understøtter også JavaScript encoding igennem brug af AjaxHelper klassen. Hermed sikres at det ikke er muligt at udføre ovenstående XSS angreb.

Epilog

Som jeg skrev i mit første indlæg så har jeg ikke udnyttet sikkerhedshullerne og har været i kontakt med ansvarlige for de to sites. XSS sårbarheden blev hurtigt fikset på Sundhed.dk (https://sundhed.dk) og kan ikke længere udnyttes, men desværre er det endnu ikke lykkedes det andet site at rette op på fejlen.

Vi må som udviklere tænke sikkerhed ind i løsningen og de to viste sikkerhedshuller er ikke besværlige at sikre sig i mod. Vi skal tænke sikkerhed ind i vores kode reviews, så det bliver en naturlig del af Definition-of-done: Har vi husket at sanitere input parametre? Encoder vi korrekt når vi viser output på skærmen?

SSL everywhere

Hvorfor ikke bruge https over alt på vores sites? Specielt hvis vi tillader folk at logge ind og have personificeret indhold! Der eksisterer mange myter om SSL og rigtigt mange sidder stadig tilbage med indtrykket af, at det er dyrt at anskaffe og har så stor indvirken på performance, at det er nødvendigt at anskaffe dedikeret hardware til kryptering. Ilya Grigorik (https://www.igvita.com/) fra Google har lavet sitet Is TLS fast yet (https://istlsfastyet.com/) , hvor myterne aflives. Ved at bruge SSL overalt vil vi sikre vores brugeres privatliv og vi sikrer os i mod Man-In-The-Middle attacks, hvor indholdet fra vores site manipuleres på vej mod brugeren.

Om Kim Bjørn Tiedemann

Kim er udviklingschef hos Schultz og arbejder også som løsningsarkitekt og scrum coach. Han har arbejdet med agil udvikling de seneste 5 år, teknologi er hans store passion, og i sin fritid koder han på en Windows 8-app. Han er uddannet i datalogi fra AAU.

Følg @kimtiede 96 følgere