



**INTEL RAD**  
INTELLIGENCE RESEARCH & DEVELOPMENT

# [ Web Uygulamalarında Kaynak Kod Analizi – II ]

[ Mehmet Dursun INCE < [mehmet.ince@intelrad.com](mailto:mehmet.ince@intelrad.com) > ]

[ 10 Haziran 2013 ]

# GİRİŞ

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 konseptte 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 [Web Uygulamalarında Kaynak Kod Analizi - I](#) 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

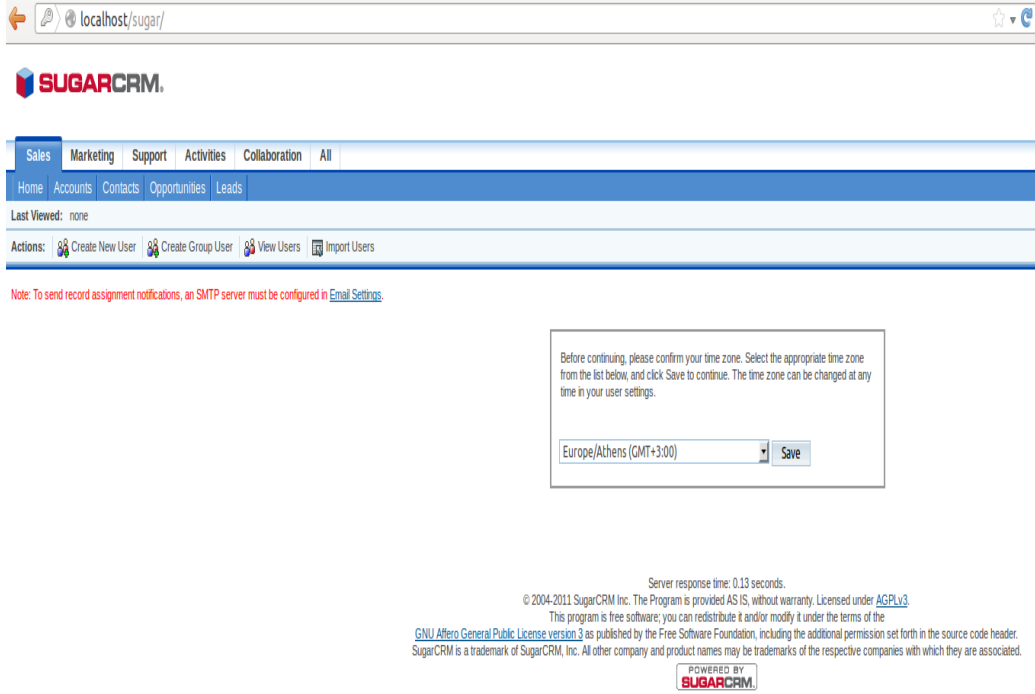
Çalışma ortamı hazırlığı için; Ubuntu üzerinde apache, mysql ve php kurulumu gerçekleştirilmiştir. Xampp gibi bütünleşik paketleri şahsen tercih etmemekteyim. Apache, mysql ve php kurulumu için gerekli adımları 1. dökümandan elde edebilirsiniz.

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.



The screenshot shows the SugarCRM installation interface. At the top, the browser address bar displays 'localhost/sugar/'. Below the browser, the SugarCRM logo is visible. The main navigation bar includes tabs for Sales, Marketing, Support, Activities, Collaboration, and All. Underneath, there are sub-tabs for Home, Accounts, Contacts, Opportunities, and Leads. The 'Last Viewed' field is set to 'none'. The 'Actions' section contains links for 'Create New User', 'Create Group User', 'View Users', and 'Import Users'. A red note states: 'Note: To send record assignment notifications, an SMTP server must be configured in [Email Settings](#).' A dialog box is open, asking the user to confirm their time zone. The text in the dialog reads: 'Before continuing, please confirm your time zone. Select the appropriate time zone from the list below, and click Save to continue. The time zone can be changed at any time in your user settings.' The dropdown menu shows 'Europe/Athens (GMT+3:00)' and a 'Save' button is next to it. Below the dialog, the server response time is shown as '0.13 seconds'. Copyright information for SugarCRM Inc. is provided, along with a link to the GNU Affero General Public License version 3. A 'POWERED BY SUGARCRM' logo is at the bottom.

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 çizgi -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/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 çıktısının 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” ile satırları getir, kalanları ekrana basma.

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

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

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$'|xargs grep '$_G
ET\|$_POST\|$_REQUEST'
```

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
search'];
/var/www/sugar/include/MySugar/MySugar.php: $category = $_RE
ategory'];
/var/www/sugar/include/MySugar/MySugar.php: if(!empty($_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
out')) {
/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 görülmektedir. 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 dosyası

```
$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']);  
test2.php         $param = $_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 açısından 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: ~  
mince@intelrad:~$ find /var/www/sugar/ -type f | grep -r '\.php$' | xargs grep '$_GET  
\|$_POST\|$_COOKIE\|$_REQUEST' | grep 'system(\|shell_exec(\| exec(\|passthru(\|un  
serialize('  
/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:           $firstrow = unserialize(ba  
se64_decode($_REQUEST['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 belirtilen 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 unserialize ve serialize metodlarına biraz daha yakından bakalım.

```
mince@intelrad: /tmp
?php
$temp = array("istanbul" => '34',
              "sakarya" => '54'
              );
echo serialize($temp);
echo "\n";

"test.php" 9L, 101C 1,1 All
```

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 oluşturulacaktır.



```
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
O:8:"Intelrad":1:{s:9:"pentester";s:12:"Mehmet INCE\n";}
```

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 fonksiyonunun 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 unserialize metoduna kullanıcı girdisi kontrol edilmeden parametre olarak atandığı tespit edilmiştir. Yani saldırgan analiz edilen yazılım genelinde ki herhangi bir sınıftan nesne türetilip bunu serialize edip elde ettiği string ifadeyi hedef uygulamaya gönderirse, bu nesne unserialize ile otomatik olarak üretilmektedir. 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.

```
mince@intelrad: /tmp
<?php
class Intelrad{
    var $pentester = "HACKER";
}

//$object = new Intelrad();

$objje = serialize(new Intelrad());
echo base64_encode($objje);

echo "\n";
-
-
-
-
"seri.php" 12L, 157C 1,1 All
```

- 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
Tzo4OiJJbnRlbHJhZCI6MTp7czo5OiJwZW50ZXN0ZXIiO3M6NjoiseFDS0VSIjt9
mince@intelrad:/tmp$
```



## 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 buldukları sınıfın değişkenleri bir dosyaya kaydediyorsa veya herhangi başka bir amaçla kullanıyorsa 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 kodlarımızı 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$ find /var/www/sugar/ -type f | grep -r '.php$' | xargs grep 'function __destruct(\| function __wakeup('
/var/www/sugar/Zend/Http/Client/Adapter/Proxy.php: public function __destruct()
/var/www/sugar/Zend/Http/Client/Adapter/Socket.php: public function __destruct()
/var/www/sugar/Zend/Http/Response/Stream.php: public function __destruct()
/var/www/sugar/Zend/Oauth/Token.php: public function __wakeup()
/var/www/sugar/modules/Import/sources/ImportFile.php: public function __destruct()
/var/www/sugar/include/tcpdf/tcpdf.php: public function __destruct() {
/var/www/sugar/include/SugarObjects/SugarSession.php: public function __destruct(){
/var/www/sugar/include/SugarTheme/SugarTheme.php: public function __destruct()
/var/www/sugar/include/SugarLogger/SugarLogger.php: public function __destruct()
/var/www/sugar/include/SugarCache/SugarCacheFile.php: public function __destruct()
/var/www/sugar/include/SugarCache/SugarCacheAbstract.php: public function __destruct()
/var/www/sugar/include/Dashlets/DashletRssFeedTitle.php: public function __destruct() {
/var/www/sugar/include/connectors/sources/default/source.php: public function __destruct(){
mince@intelrad: /tmp$
```



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         // Bug 28309 - Set the current directory to one which we expect it to be (i.e. the
313         // root directory of the install
314         set_include_path(realpath(dirname(__FILE__) . '/../..') . PATH_SEPARATOR . get_in
315         clude_path());
316         chdir(realpath(dirname(__FILE__) . '/../..'));
317
318         // clear out the cache on destroy if we are asked to
319         if ( $this->clearCacheOnDestroy ) {
320             if (is_file($GLOBALS['sugar_config']['cache_dir'].$this->getFilePath().'/pathC
321             ache.php'))
322                 unlink($GLOBALS['sugar_config']['cache_dir'].$this->getFilePath().'/pathC
323             ache.php');
324         }
325         elseif ( !inDeveloperMode() ) {
326             // only update the caches if they have been changed in this request
327             if ( count($this->_jsCache) != $this->_initialCacheSize['jsCache']
328                 || count($this->_cssCache) != $this->_initialCacheSize['cssCache']
329                 || count($this->_imageCache) != $this->_initialCacheSize['imageCache']
330                 || count($this->_templateCache) != $this->_initialCacheSize['template
331             Cache']
332             )
333             {
334                 sugar_file_put_contents(
335                 create_cache_directory($this->getFilePath().'/pathCache.php'),
336                 serialize(
337                 array(
338                 'jsCache' => $this->_jsCache,
339                 'cssCache' => $this->_cssCache,
340                 'imageCache' => $this->_imageCache,
341                 'templateCache' => $this->_templateCache,
342                 )
343                 )
344             );
345         }
346     }
347 }
```

Lütfen kodları inceleyiniz.









## 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