En Riskli 10 OWASP Zafiyetleri İncelemesi

Yazılım Mühendisliği Ana Bilim Dalı

Dönem Projesi

Doğukan Aslan

ORCID 0000-0002-5655-7372

Proje Danışmanı: Prof. Dr. Femin Yalçın Küçükbayrak

Haziran 2024

En Riskli 10 OWASP Zafiyetleri İncelemesi

Öz

OWASP (Açık Web Uygulama Güvenliği Projesi) adlı topluluk, hacker olarak adlandırılan kötü niyetli kişilere karşı mücadele etmek için kurulmuştur. OWASP, en büyük güvenlik açıklarını inceleyerek belirli kriterlere göre sıralar. Bu çalışmada, en tehlikeli açıklar incelenmiştir. "Injection" adı verilen bu açıklıklar genellikle kullanıcı tarafından sağlanan ve denetlenmeyen verilerden kaynaklanır. Saldırganlar sistemlere beklenmedik veriler gönderebilir ve yetkisi olmayan verilere izinsiz erişebilir. "Broken Authentication" olarak bilinen açıklık ise genellikle kimlik doğrulama, oturum açma gibi işlevlerin yanlış kullanılması sonucunda ortaya çıkar. Saldırganlar genellikle bu tür saldırılar için "Brute Force" yöntemini kullanırlar. "Sensitive Data Exposure" açıklığı, şifrelenmemiş veya eski varsayılan şifreleme algoritmalarının kullanıldığı verilerin ortaya çıkmasına neden olur. "XML External Entities" açığı, eski veya yanlış yapılandırılmış XML dosyalarından kaynaklanır. "Broken Access Control" açığı, kullanıcı haklarının yeterince kontrol edilmemesi sonucunda ortaya çıkar. "Security Misconfiguration" açığı, hizmet yapılandırmasının yanlış veya eksik olmasından kaynaklanır. "Cross Site Scripting" açığı kullanıcı tarafından sağlanan verilerin tam olarak denetlenmeden veya belirli bir filtreden geçirilmeden yanıt olarak gönderilmesiyle oluşur. "Insecure Deserialization" açığı uzaktan kod çalıştırılmasına neden olabilir. "Using Components with Known Vulnerabilities" açığı, kullanılan bileşenlerin eski sürümleri veya bilinen güvenlik açıkları olan sürümleri nedeniyle ortaya çıkar. "Insufficient Logging and Monitoring" açığı ise izleme yapılandırmalarıyla birlikte saldırganların daha fazla saldırı gerçekleştirmesine, sistemde daha uzun süre kalmasına ve verilere istedikleri gibi müdahale etmesine olanak tanır.

Anahtar Sözcükler: OWASP, Tehlike, Zafiyet, Açıklık, Saldırgan

The Examination of The Top 10 OWASP Vulnerabilities

Abstract

The community known as OWASP (Open Web Application Security Project) was established to combat malicious individuals termed as hackers. OWASP ranks the most significant security vulnerabilities based on specific criteria by examining them. This study focuses on the most critical vulnerabilities. These vulnerabilities, known as "Injection," often stem from unvalidated user input. Attackers can send unexpected data to systems and gain unauthorized access to sensitive data. The vulnerability known as "Broken Authentication" typically arises from misusing functions like authentication and session management. Attackers commonly employ "Brute Force" methods for such attacks. "Sensitive Data Exposure" occurs when data is exposed due to lack of encryption or the use of outdated encryption algorithms. The "XML External Entities" vulnerability arises from outdated or improperly configured XML files. "Broken Access Control" emerges from inadequate control over user rights. "Security Misconfiguration" results from incorrect or incomplete service configurations. "Cross-Site Scripting" vulnerability occurs when user-supplied data is sent as a response without proper validation or filtering. "Insecure Deserialization" vulnerability can lead to remote code execution. "Using Components with Known Vulnerabilities" arises due to the use of outdated versions of components or versions with known security flaws. The "Insufficient Logging and Monitoring" vulnerability, when combined with improper monitoring configurations, allows attackers to execute more attacks, stay within the system for longer periods, and manipulate data as they please.

Keywords: OWASP, Threat, Vulnerability, Exploit, Attacker

Teşekkür

En Riskli 10 OWASP Zafiyetleri İncelemesi, konulu çalışma İzmir Kâtip Çelebi Üniversitesi Yazılım Mühendisliği Bölümünde 2023-2024 bahar döneminde dönem projesi olarak hazırlanmıştır.

Yüksek lisans dönem projesinin planlanmasında, araştırılmasında ve yürütülmesinde desteğini eksik etmeyen danışman hocam Prof. Dr. Öğr. Femin Yalçın Küçükbayrak'a sonsuz teşekkürlerimi sunarım.

İçindekiler

Şekiller Listesi

Tablolar Listesi

Kısaltmalar Listesi

OWASP	Open Web Application Security Project
FBE	Fen Bilimleri Enstitüsü
İKÇÜ	İzmir Kâtip Çelebi Üniversitesi
ORCID	Open Researcher and Contributor ID
CMD	Command Prompt
HTTP	Hyper Text Transfer Protocol
LDAP	Lightweight Directory Access Protocol
OS	Operating System
URL	Uniform Resource Locator
SQL	Structured Query Language
XML	Extensible Markup Language
DTD	Document Type Definition
DOCTYPE	Document Type
XXE	XML External Entity
AD	Active Directory
XSS	Cross-Site Scripting
CSP	Content Security Policy
WAF	Web Application Firewall

Bölüm 1

1 GİRİŞ

Günümüzde bilgisayar sistemlerine yapılan saldırılar gün geçtikçe artıyor. Saldırılar her geçen gün daha da karmaşıklaşmaktadır. Kurum ve kuruluşlara yapılan siber saldırılar, maddi zararlara yol açmaktadır, ayrıca yapılan bu siber saldırılar kurum ve kuruluşların saygınlıklarını kaybetmesine ve bilgilerin çalınmasına sebep olmaktadır. Siber saldırılara karşı sistemlerin ve sistem yöneticilerinin önlem alması gerekmektedir. Bilgiye ulaşmanın kolaylaşmasıyla birlikte saldırganların teknik bilgisi her geçen gün artmaktadır. Siber saldırıları gerçekleştiren siyah şapkalı hackerlerin vereceği zararları engellemek amacıyla beyaz şapkalı hacker ekipleri kurulmuştur.

OWASP, Open Web Application Security Project anlamına gelmektedir. Günümüzde beyaz şapkalı hackerlerin yetiştirilmesi yeteri kadar pratik yapılabilmesi için, web uygulamalarındaki güvenlik zafiyetlerinin kapatılması ve sürekli olarak güvenliğinin sağlanması için çalışmalar yapan özgür bir topluluktur.

Bölüm 2

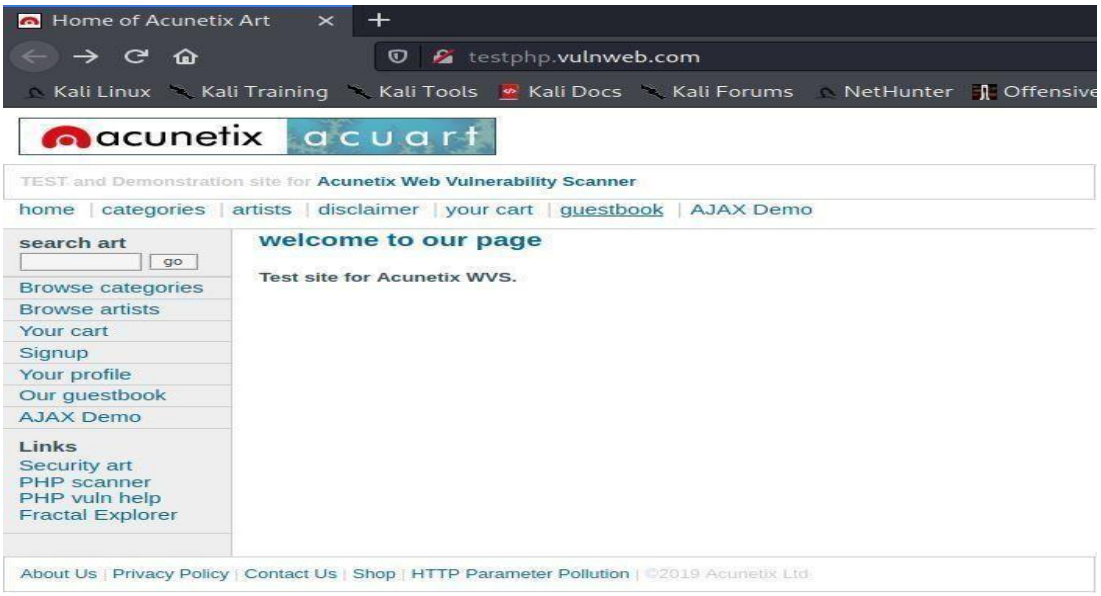
2.1 ENJEKSİYON

OWASP (Open Web Application Security Project) tarafından tanımlanan en tehlikeli 10 güvenlik açığı arasında "Injection" belirgin bir öneme sahiptir. Injection, genellikle veri tabanı sorgularına veya diğer komutlara zararlı veri enjekte edilerek gerçekleştirilen bir saldırı biçimidir. Bu, bir web uygulamasının, kullanıcı tarafından sağlanan girişleri doğru bir şekilde işlememesi veya filtrelememesi durumunda ortaya çıkar.

Örneğin, SQL Injection saldırıları, bir saldırganın bir web formu veya URL parametresi aracılığıyla SQL sorgularına zararlı kod enjekte etmesini sağlar. Uygulama girişleri uygun şekilde filtrelenmediğinde, saldırganın SQL komutları uygulanabilir hale gelir. Bu durumda saldırgan, veri tabanında istenmeyen işlemler gerçekleştirebilir, veri çalabilir veya veri tabanını tamamen ele geçirebilir. Diğer türlerde benzer durumlar meydana gelebilir. Örneğin, LDAP Injection, XML Injection veya OS Command Injection gibi türlerde, saldırganlar farklı enjekte edilen verilerle ilgili olarak veri tabanlarına, LDAP dizinlerine, XML dosyalarına veya işletim sistemine zararlı komutlar enjekte edebilirler.

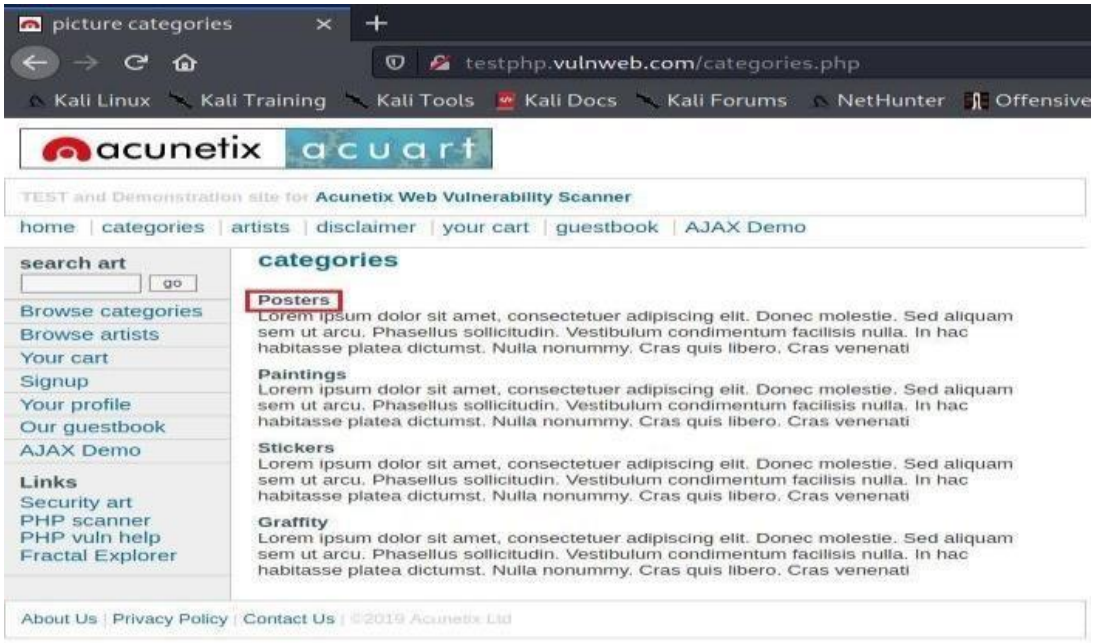
2.1.1 SQL Enjeksiyon

SQL Injection, bir web uygulamasının güvenlik açıklarından biridir ve kötü niyetli kişilerin veri tabanı işlemlerine karışmalarına olanak tanır. Bu saldırı türünde, saldırganlar web uygulamasına sağlanan giriş alanlarına veya parametrelere özel olarak hazırlanmış SQL kodları ekleyerek veri tabanı sorgularını değiştirirler. Eğer web uygulaması girişleri yeterince filtrelenmez veya doğrulanmazsa, bu SQL kodları başarıyla çalıştırılır ve saldırganlar veri tabanından istedikleri bilgilere erişebilir, veri tabanını değiştirebilir veya hatta kontrol edebilirler. SQL Injection saldırıları oldukça yaygın ve ciddi bir güvenlik tehdidi oluşturur.



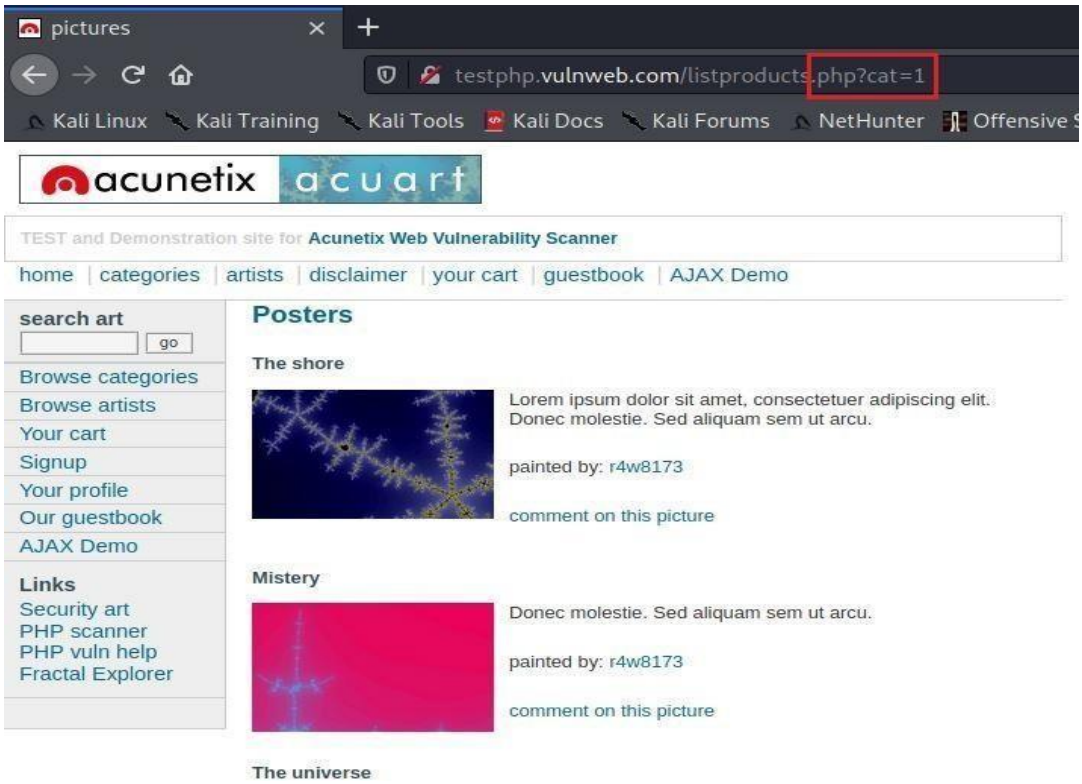
Şekil 2.1. 1: Zafiyetli (testphp.vulnweb.com) Web Sitesinin Giriş Ekranı

Şekil 2.1.1'de zafiyetli web sitesinin giriş ekranı gösterilmiştir. Bir kullanıcı adı ve şifre girişi alanı olan bir web formunda, bir saldırgan SQL Injection kullanarak giriş alanlarına eklediği özel bir SQL kodu aracılığıyla kullanıcı adı ve şifre kontrolünü atlayabilir veya kullanıcı adı ve şifre bilgilerini ele geçirebilir. Bu şekilde, saldırganın izinsiz olarak sisteme erişmesi mümkün olabilir. Bu örneği, bu tür eğitimler için planlanmış olan "http://testphp.vulnweb.com" web sitesinde görebiliriz.



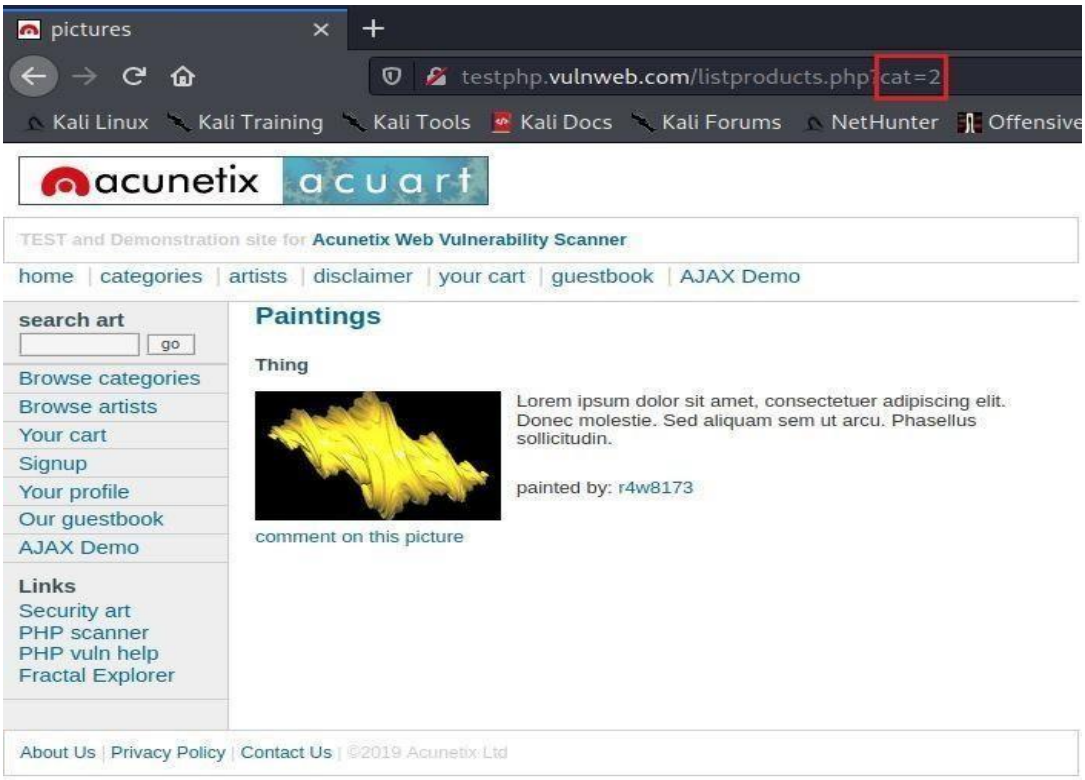
Şekil 2.1. 2: Zafiyetin Bulunması İçin Uygulanan İşlemler (A)

Şekil 2.1.2 (A)'da "Browse Categories" sekmesine tıklanıldı ve açılan ekrandan "Posters" kısmına girildi.



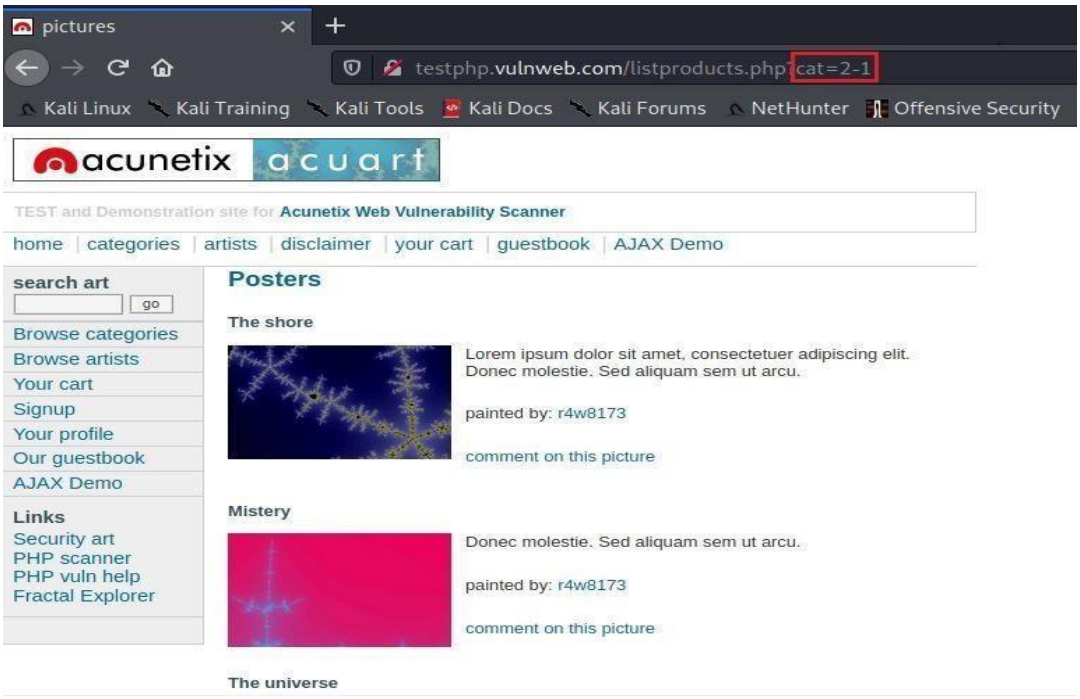
Zafiyetin Bulunması İçin Uygulanan İşlemler (B)

Şekil 2.1.2 (B)'de Posters kısmında ekrana gelen requeste bakıldı.



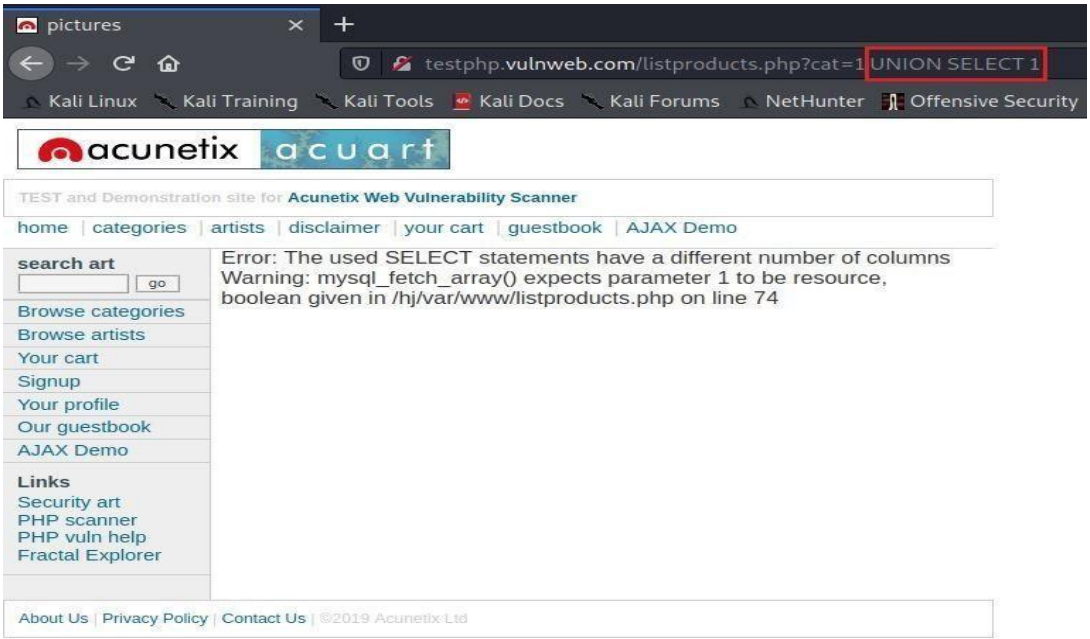
Zafiyetin Bulunması İçin Uygulanan İşlemler (C)

Şekil 2.1.2 (C)'de Request değeri 2 yapıldığında sayfada olan değişim gözlemlendi.



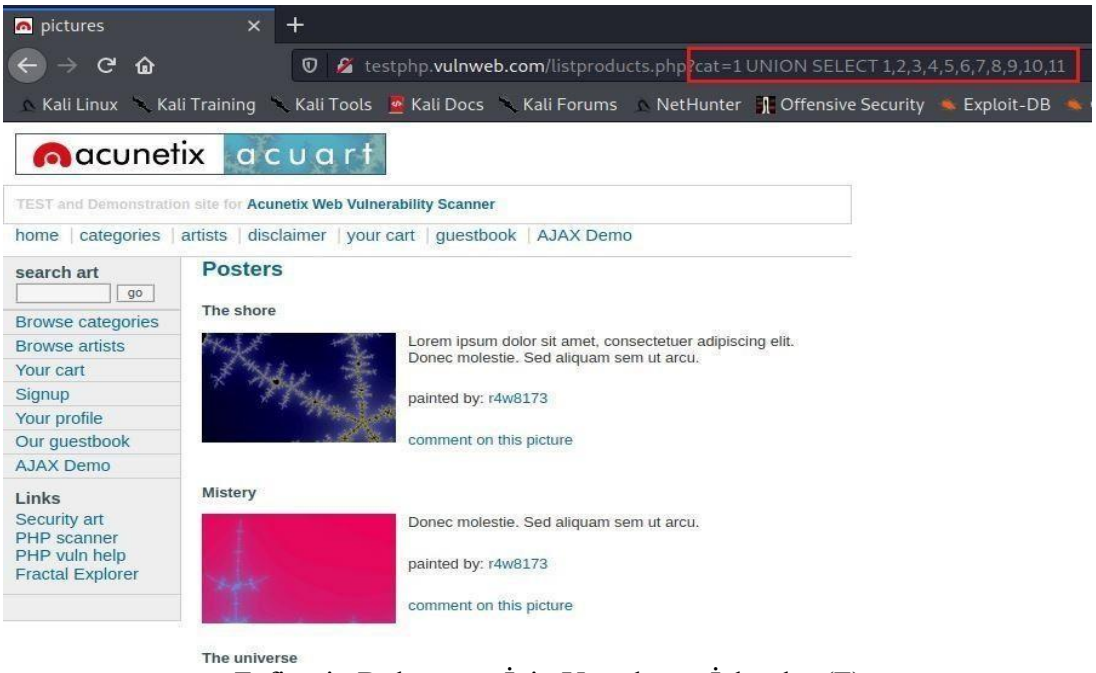
Zafiyetin Bulunması İçin Uygulanan İşlemler (D)

Şekil 2.1.2 (D)'de Request değeri '2-1' olarak değiştirildi.



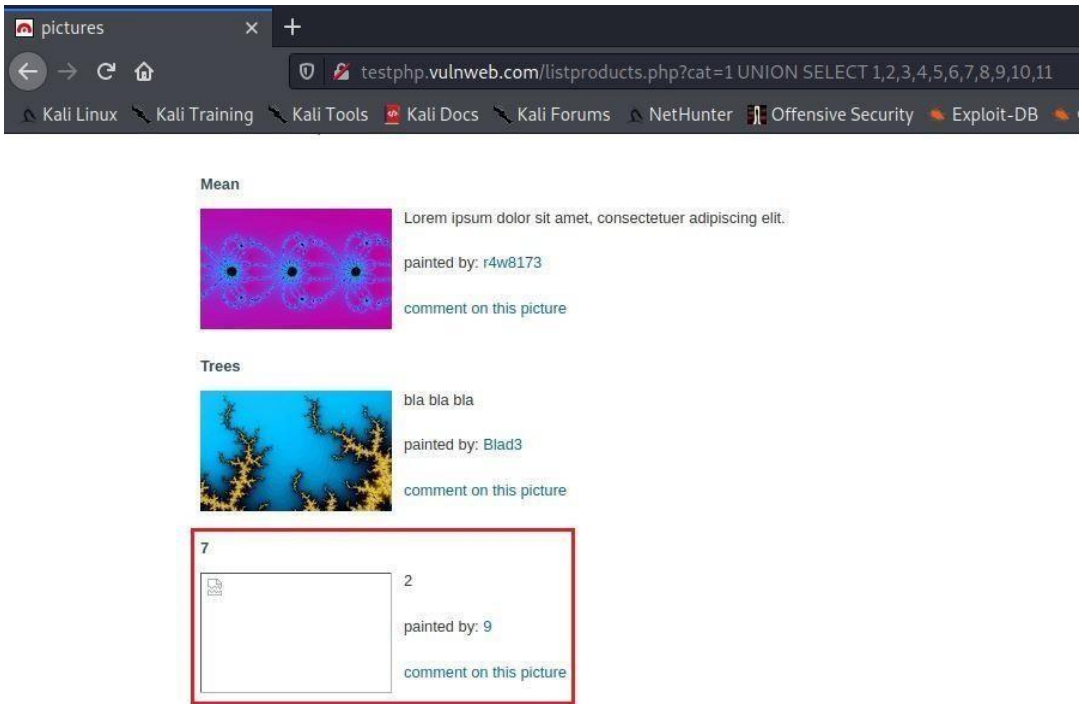
Zafiyetin Bulunması İçin Uygulanan İşlemler (E)

Şekil 2.1.2 (E)'de görüldüğü gibi tekrardan ilk sayfaya dönüldü. Çünkü veri tabanında arama yaparken '2 - 1' işlemi '1' olarak algılanır. Gözlem sonucunda, "Union" ifadesini kullanarak URL kısmında çeşitli sorgular yapılabilir. Fakat dikkat edilmesi gereken şey kolon sayılarının eşit olmasıdır. URL'ye UNION SELECT 1 sorgusu eklenip sonuç gözlemlendi ve bir hata alındı. Bu hata bize kolon sayılarının aynı olmadığını söylemektedir. Önemli olan referans noktasına dönülebilmesidir.



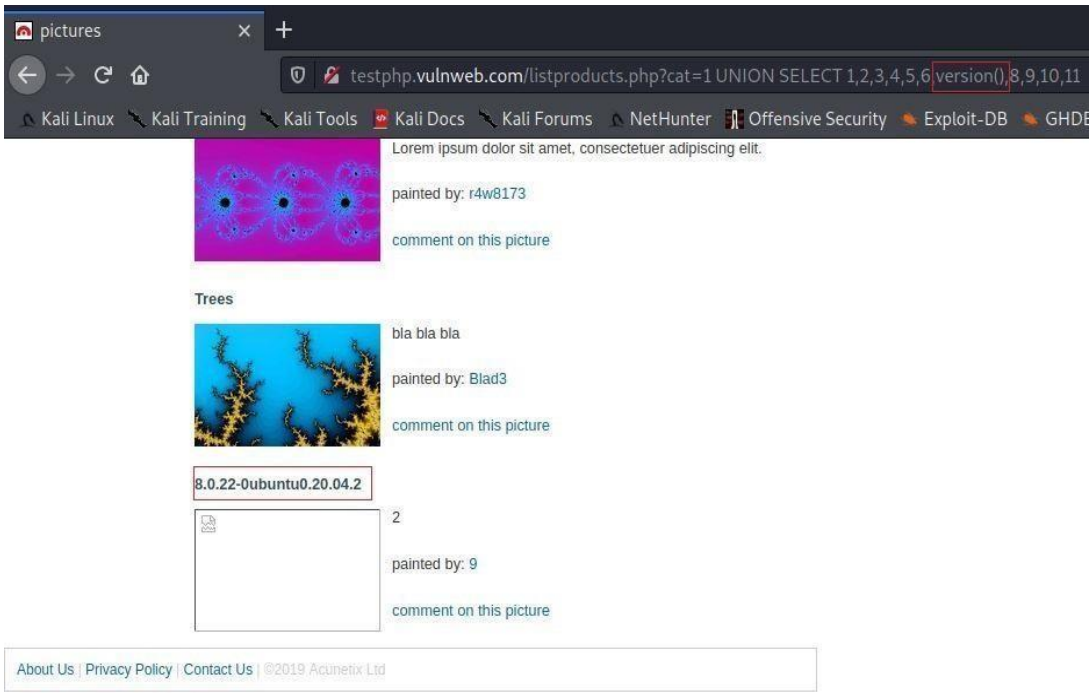
Zafiyetin Bulunması İçin Uygulanan İşlemler (F)

Şekil 2.1.2 (F)'de Yapılması gereken şey referans noktasına dönene kadar kolon sayısını arttırmaktır. Kolon sayısını arttırarak kontrol edildiği zaman, değer 11 geldiğinde referans noktasına dönüldüğü gözlemlendi.



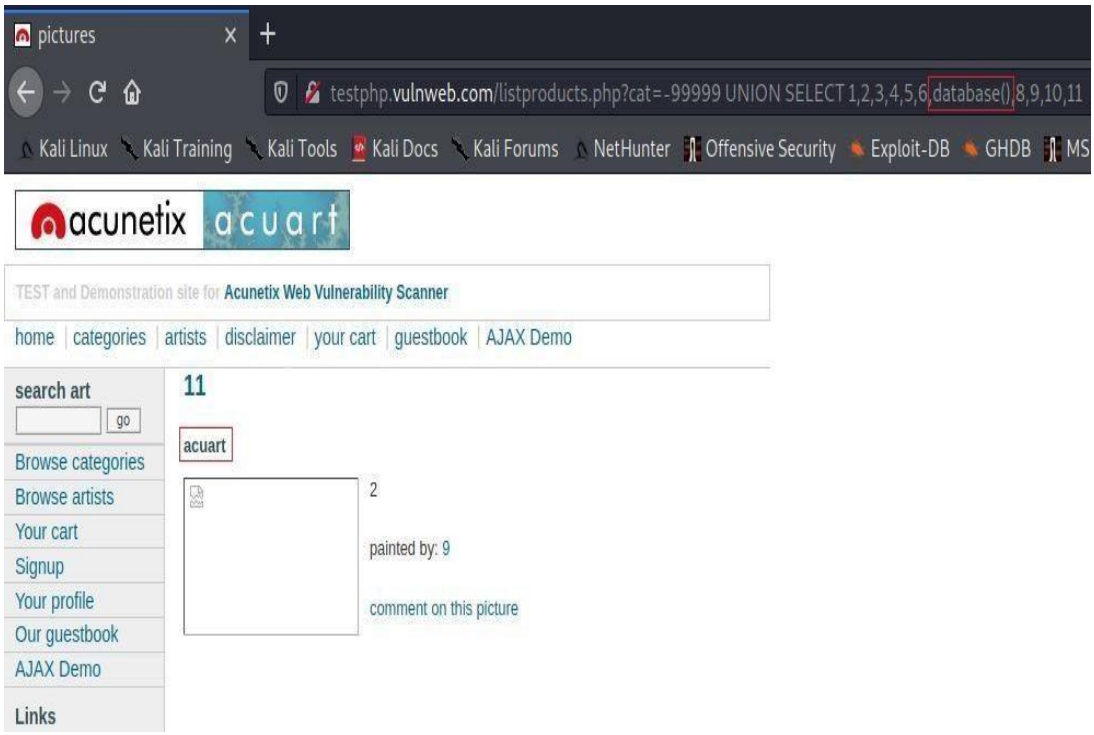
Zafiyetin Bulunması İçin Uygulanan İşlemler (G)

Şekil 2.1.2 (G)'de Sayfada aşağı doğru inildiğinde referans sayfasından farklı değerler gözükmektedir. 7, 2 ve 9 değerlerinde demek istediği, daha önce ki "SELECT" sorgusunun indislerini uygulama ve kod sayesinde ekrana yazdır.



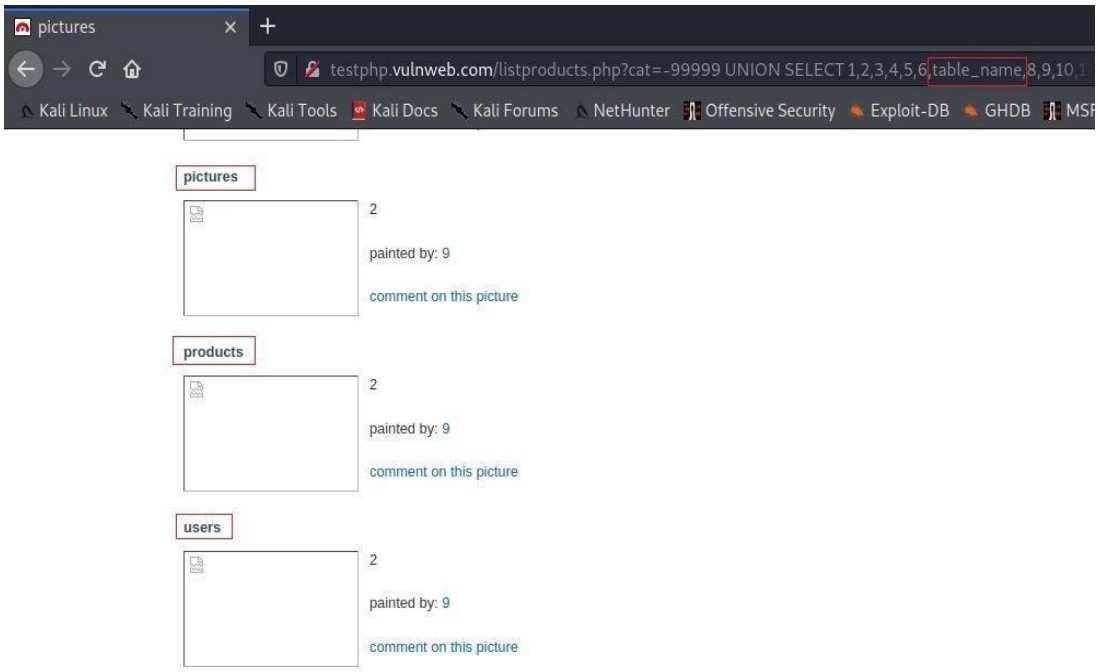
Zafiyetin Bulunması İçin Uygulanan İşlemler (H)

Şekil 2.1.2 (H)'de örnek olarak, 7. Kolonda bir yardımcı fonksiyon çalıştırılırsa, mesela "version()"fonksiyonu, sayfada bu versiyon gözükür. Veri tabanından veri çıkarmaya başlandı.



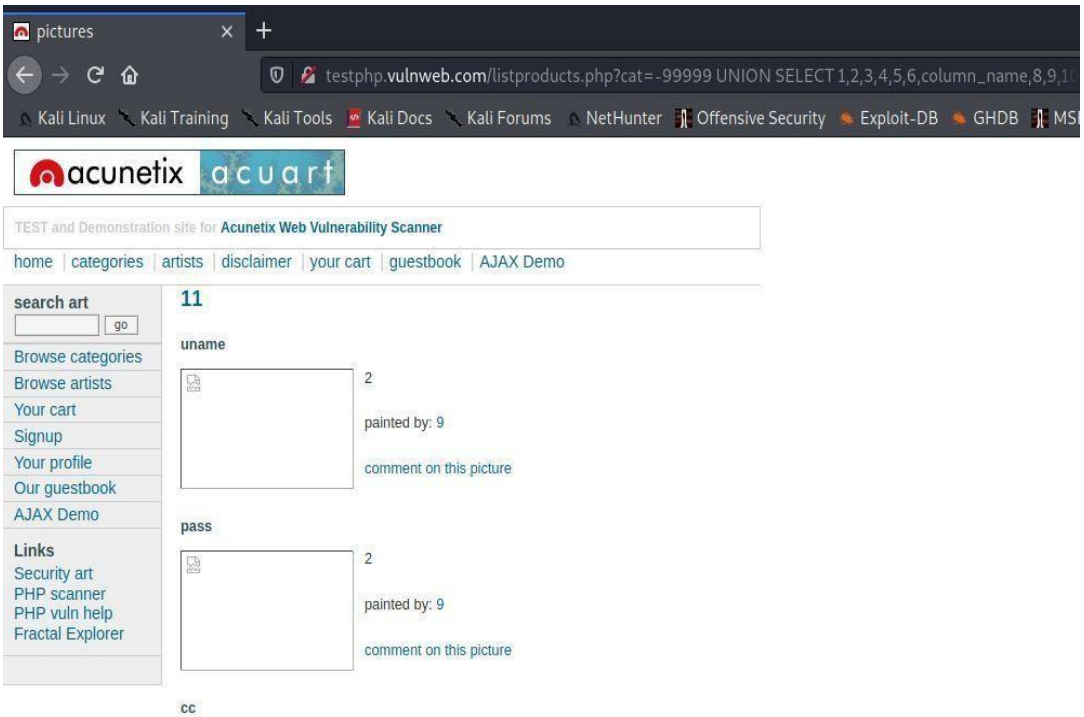
Zafiyetin Bulunması İçin Uygulanan İşlemler (I)

Şekil 2.1.2 (I)'da URL'de version() fonksiyonu yerine database() yardımcı fonksiyonunu yazılırsa, veri tabanı ismi elde edilir.



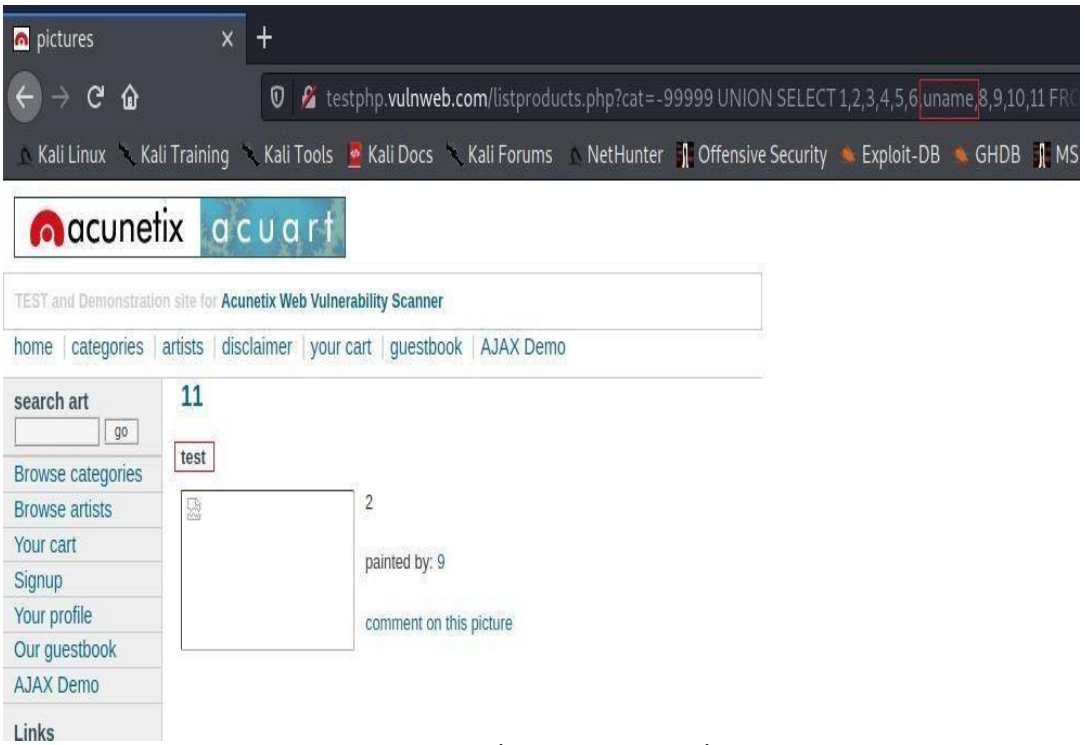
Zafiyetin Bulunması İçin Uygulanan İşlemler (İ)

Şekil 2.1.2 (İ)'de URL kısmına "FROM information_schema.tables WHERE table_schema = database()" sorgusu yazılırsa ve 7 rakamı yerine "table_names" yazıldığında sayfa veri tabanında bulunan tablo isimleri çıkar.



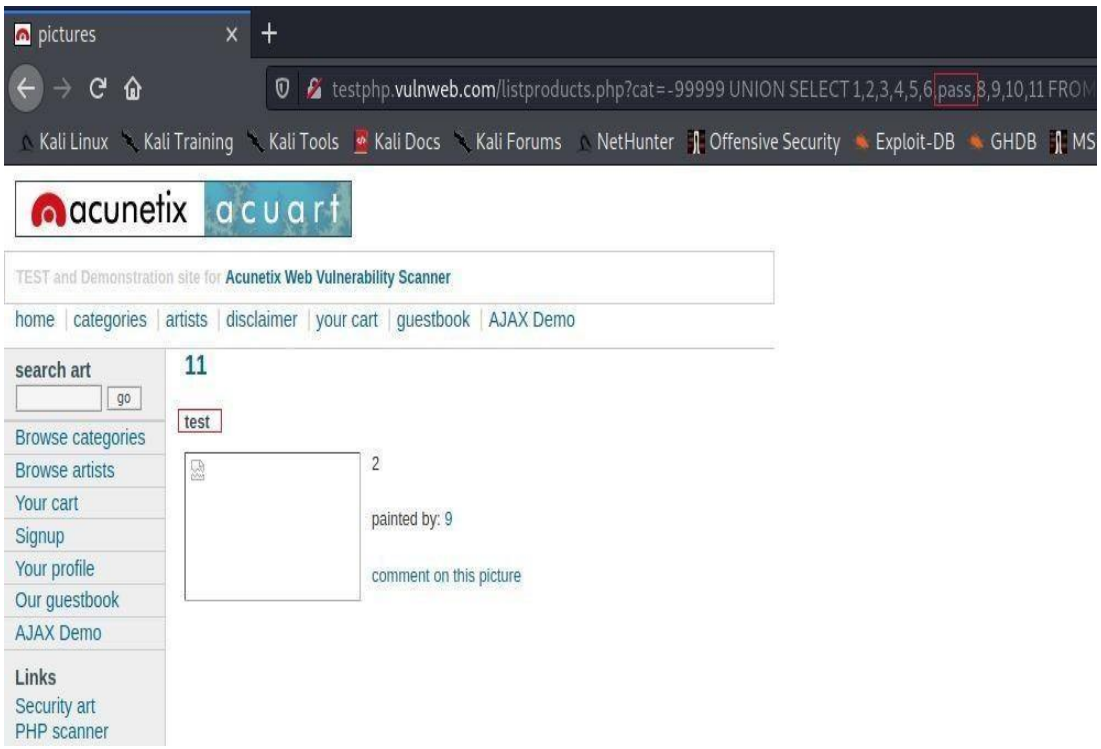
Zafiyetin Bulunması İçin Uygulanan İşlemler (J)

Şekil 2.1.2 (J)'de "FROM information_schema.columns WHERE table_name = 'users'" olarak değiştirip, 7. Kolona "column_name" yazılırsa, "users" tablosunun içeriği ekrana yansır.



Zafiyetin Bulunması İçin Uygulanan İşlemler (K)

Şekil 2.1.2 (K)'da soru sorguları değiştirilip uname ve pass'a erişim sağlanabilir. Sorguya "FROM users" 7. Kolona "uname" yazılırsa, sayfaya kullanıcı adını yazdıracaktır.



Zafiyetin Bulunması İçin Uygulanan İşlemler (L)

Şekil 2.1.2 (L)'de 7.Kolona "pass" yazılırsa, sayfaya şifreyi yazdıracaktır.

2.1.2 Komut Enjeksiyon

Command Injection, hedefte bulunan savunmasız bir uygulama aracılığıyla ana bilgisayar işletim sisteminde rastgele komutların yürütülmesidir ve kötü niyetli kişilerin komut istemcisine (Command Shell) müdahale etmelerine olanak tanır. Bu zafiyetin sistemde kullanıcıdan input alınan her yerde bulunma ihtimali oldukça fazladır. Örneğin formlar, tanımlama bilgileri ve HTTP üst bilgileridir. Bu zafiyet kullanıcının girmiş olduğu verinin yanında Linux ya da Windows komutları çalıştırmasına olanak sağlar. Zafiyetin var olduğu sistemde bir arama kutusu veya URL parametresi gibi giriş alanları olan bir web uygulamasında, bir saldırgan Command Injection kullanarak giriş alanlarına eklediği özel bir komut aracılığıyla sistem komutlarını çalıştırabilir. Bu şekilde, saldırgan dosya indirebilir, sistem dosyalarını okuyabilir veya değiştirebilir, hatta sistem üzerinde yetki kazanabilir. Bu zafiyet cmd veya Linux terminalinde komut çalıştırmaya benzemektedir.

2.1.2.1 Komut Enjeksiyon Karakter Kullanımı

Command Injection saldırılarında kullanılan bazı yaygın karakterler şunlardır:

- ";" (Noktalı virgül): Komutları ayırmak ve ardışık komutlar eklemek için kullanılabilir.
- "|" (Dikey çizgi): Komutların çıktılarını başka bir komuta yönlendirmek için kullanılabilir.
- "&" (Ve işareti): Komutları birleştirmek veya arka planda çalıştırmak için kullanılabilir.
- "`" (Geri açılı tırnak): İçine alınan komutları çalıştırmak için kullanılabilir.

Örneğin, bir saldırgan aşağıdaki gibi bir URL'de Command Injection saldırısı gerçekleştirebilir: http://example.com/search?query=test;ls%20-la

Bu URL'de, "test" kelimesi aranırken, ";" karakteri ile ayrılan bir başka komut eklenmiştir ("ls -la" komutu), böylece bu komut da çalıştırılacaktır. Bu şekilde, saldırgan sistemin dosya listesini alabilir.

Şekil 2.1. 3: Command Zafiyeti Bulunan Site (A)

Command Zafiyeti Bulunan Site (B)

Şekil 2.1.3 (B)'da "Lookup" butonuna basıldığında domaine ait bilgiler elde edilmektedir. Bazı command injection karakterleri kullanarak Linux komutları çalıştırılmıştır.

Command Zafiyeti Bulunan Site (C)

Şekil 2.1.3 (C)'de öncelikle "&" ifadesi kullanılarak "id" komutu çalıştırılmıştır. Sonuç olarak "www-data" kullanıcısının id bilgisi elde edilmiştir.

Command Zafiyeti Bulunan Site (D)

Şekil 2.1.3 (D)'de Bir başka ifade olan ";" ifadesini denenmiştir. Bu ifadeyi kullanılarak "cat" komutu ile "/etc/passwd" dosyası okunmuştur.

2.1.2.2 Komut Enjeksiyon Zafiyetinin Önlenmesi

Command Injection zafiyetlerini engellemek için aşağıdaki adımlar atılabilir:

1. Kullanıcı Girişlerinin Onaylanması ve Filtrelenmesi: Web uygulamaları, kullanıcı girişlerini denetlemeli ve güvensiz karakterleri elemelidir. Kullanıcı girişlerinin temizlenmesi veya kabul edilmemesi gereken karakterlerin reddedilmesi, Command Injection saldırılarını durdurur.
2. Parametrize Sorguların İncelenmesi: SQL Injection'da olduğu gibi, Command Injection saldırılarını önlemek için parametrize sorgular kullanılmalıdır. Parametrize sorgular, kullanıcı girişlerini sorguya doğrudan eklemek yerine, ayrı parametreler olarak işlenir ve bu şekilde saldırganların zararlı komutlarını eklemesi zorlaşır.
3. En Az Ayrıcalık İlkesi: Web uygulamaları, işletim sistemi komutlarını çalıştırmak için en düşük ayrıcalığa sahip kullanıcı hesaplarını kullanmalıdır. Bu şekilde, saldırganların erişebileceği zararlı işlemler sınırlanır.
4. Güvenlik Denetimleri: Web uygulamaları, düzenli olarak kullanıcı girişlerini ve uygulama içi iletişimi kontrol etmeli ve güvenlik açıkları tespit edildiğinde hızlıca düzeltmelidir.
5. Yazılım Güncellemelerinin Yapılması: İşletim sistemi ve web uygulaması yazılımları güncel tutulmalıdır. Yazılım güncellemeleri genellikle güvenlik açıklarını düzeltir ve Command Injection gibi saldırılara karşı daha sağlam bir koruma sağlar.
6. Güvenlik Duvarının Kullanılması: Web uygulamaları için bir güvenlik duvarı konumlandırılabilir. Güvenlik duvarı, gelen istekleri izler ve potansiyel saldırıları engeller veya filtreler.
7. Eğitim ve Farkındalık: Geliştiricilere ve sistem yöneticilerine Command Injection ve diğer güvenlik zafiyetleri hakkında eğitim verilmeli ve güvenlik önlemleri konusunda farkındalık oluşturulmalıdır.

Zafiyetin oluşum nedeni sistem komutlarının kullanımından kaynaklanmaktadır. En mantıklı korunma yöntemi sistem komutlarının kullanımına izin verilmemesidir ama eğer bir nedenden ötürü izin verildiyse yukarıdaki gibi bazı önlemler alınabilir. Ayrıca aşağıdaki önlemler de alınabilir:

- Alınan girdi formatının kontrolünün sağlanması ve yetki analizinin yapılması.
- Sistem üzerinde girdileri değerlendirmek için “Whitelist, Blacklist” gibi yapıların oluşturulması.
- Girdi işleminin yalnızca alfa numerik karakterlerden oluşması.
- Özel karakterlerin site içerisinde kullanımının kısıtlanması ya da engellenmesi.

2.2 BOZUK KİMLİK DOĞRULAMA

Broken Authentication, bir web uygulamasının kimlik doğrulama ve oturum yönetimi işlemlerindeki zayıflıklardır. Bu tür zayıflıklar, kötü niyetli kişilerin kimlik doğrulama mekanizmalarını atlama veya oturumları ele geçirme yoluyla yetkisiz erişim sağlamalarına olanak tanır.

Örneğin, bir web uygulamasında kullanıcıların oturum açmak için kullandığı kimlik doğrulama mekanizmasının zayıf veya hatalı olması durumunda, saldırganlar kullanıcı adı ve parola gibi bilgileri ele geçirerek hesaplara yetkisiz erişim sağlayabilirler. Bu da kullanıcıların verilere veya özel bilgilere yetkisiz erişim sağlaması anlamına gelir.

2.2.1 Kimlik Doğrulaması

Kimlik bilgisi doğrulanırken, kullanıcıdan alınan kimlik bilgilerinin, giriş yapacağı sistemdeki kullanılan veri tabanında bulunan kimlik bilgileri ile karşılaştırılır. Kimlik bilgileri eşleşirse kullanıcı sisteme giriş yapabilir. Giriş yapan kullanıcının erişebileceği dizinler ve yapabileceği işlemler daha önceden hesabın sistemde belirlendiği yetki düzeyine göredir.

Kimlik bilgisi doğrulaması birden fazla yöntemle yapılabilir. Temel seviye kimlik doğrulamasında kullanıcı adı veya e-posta ve parola bilgisi kullanılır. Daha güçlü bir kimlik doğrulaması için ise kullanıcı adı ve parola dışında farklı doğrulama faktörleri kullanılır. Günümüzde kullanılan bu faktörler:

- Bilgi Faktörü: Kullanıcı adı ve parolanın yanında tek kullanımlık şifreler, pin veya gizli bir sorunun cevabı istenilebilir.
- Kalıtım Faktörü: Kişiye özel fiziksel özelliklerin kullanıldığı faktördür. Parmak izi, yüz tanıma, retina taraması örnek olarak verilebilir.
- Konum Faktörü: Sisteme sadece belirlenmiş konumlardan girme işlemidir.
- Sahiplik Faktörü: Kişiye özel belirlenmiş kart, aygıt veya cihazlardır.
- Zaman Faktörü: Genelde konumla birlikte kullanılır ve saat farkı gözetilir.

2.2.2 Kimlik Doğrulama Zafiyetinin Oluşması

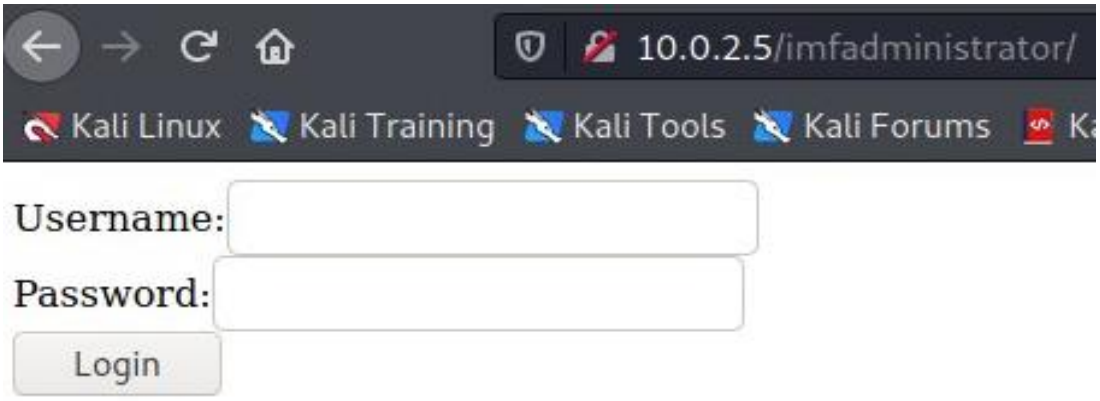
Kimlik doğrulaması zafiyetleri, çeşitli faktörlerin bir araya gelmesi sonucu ortaya çıkabilir. Bu zafiyetlerin oluşmasına neden olan bazı ana faktörler:

- Zayıf Parola Politikaları: Web uygulamaları, kullanıcıların zayıf veya kolayca tahmin edilebilir parolalar seçmelerine izin veriyorsa, bu durum kimlik doğrulaması açısından zafiyetlere yol açabilir. Kullanıcıların güçlü parolalar oluşturmalarını sağlamak için uygun politikaların uygulanması kritiktir.
- Kullanıcı Giriş Bilgilerinin Güvenliği: Kullanıcıların giriş bilgileri (kullanıcı adları, parolalar) güvenli bir şekilde saklanmadığı veya aktarılmadığı durumlarda, saldırganlar bu bilgilere erişebilir ve kimlik doğrulamasını atlayabilir. Veri tabanı saldırıları, saldırganların bu bilgilere ulaşmasına neden olabilir.
- Zayıf Oturum Yönetimi: Web uygulamaları, oturumların güvenli bir şekilde yönetilmediği zaman, zafiyetlere açık hale gelebilir. Oturum kimlikleri (session IDs) veya çerezlerin (cookies) kötüye kullanılması, oturumların ele geçirilmesine ve izinsiz erişime neden olabilir.
- Yanlış Konfigürasyonlar: Web sunucusu, veri tabanı veya diğer bileşenlerin yanlış yapılandırılması, güvenlik açıklarının meydana gelmesine yol açabilir. Örneğin, varsayılan kullanıcı adı ve parolaların kullanılması gibi hatalar, saldırganların sisteme erişmesine imkân tanır.
- Güncelleme ve Yama Eksiklikleri: Web uygulamaları ve altta yatan yazılımların düzenli olarak güncellenmemesi veya yamalanmaması durumunda, güvenlik açıkları ortaya çıkabilir ve saldırganlar bunları kullanarak kimlik doğrulamasını geçebilir veya bilgilere ulaşabilir.

2.2.3 Kimlik Doğrulama Zafiyetinin İstismar Edilmesi

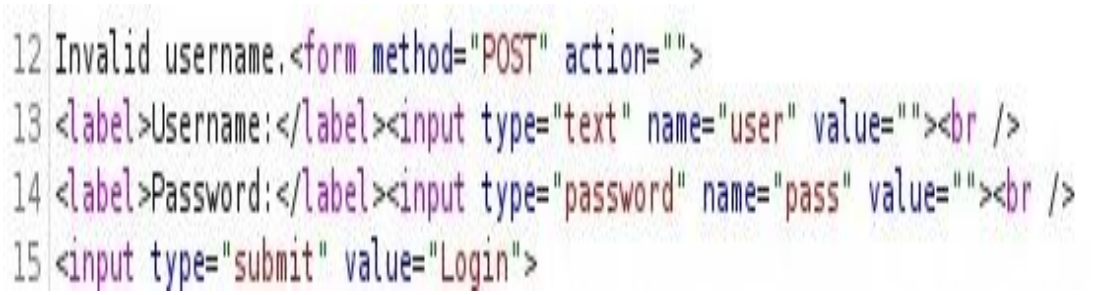
Aşağıdaki örnekte yazılımcının php strcmp fonksiyonunu uygun şekilde kullanmadığında nasıl bir zafiyet doğurduğu gösterilmiştir. Bu örnek için kurban makine olarak vulnhub sitesinden IMF:1 sanal makinesi kullanılmıştır. Saldırgan makine olarak da Kali Linux kullanılmıştır. IMF:1 bilerek zafiyet bırakılmış bir sanal

makinedir. Kurulum tamamladıktan sonra kimlik bilgilerinin istendiği bir login giriş sayfası karşılamıştır.



Şekil 2.2. 1: Zafiyet Bulunduran Sitenin Giriş Sayfası

Şekil 2.2. 2: Zafiyet Bulunduran Sitenin Giriş Sayfası Request Çıktısı



Şekil 2.2. 3: Zafiyet Bulunduran Sitenin Giriş Sayfası Response Çıktısı

Sonuçlara bakıldığında kimlik bilgilerinin user ve pass değişkenine atanarak alındığı gözükmektedir.

IMF:1 makinesinin ilgili PHP kodu incelendiğinde ilgili dizine girip cat komutu kullanarak açılan kodlar komut satırında gösterilmektedir.

```
www-data@imf:/var/www/html/imfadministrator$ cat index.php
cat index.php
<?php
session_start();
$loggedin=false;
if ($_SESSION['admin_logged_on'] == 'that is affirmative sir') {
        echo "flag3{Y29udGludWVUT2Ntcw==}<br />Welcome, ".$_POST["user"] . "<br /><a href='cms.php?pagename=home'>IMF CMS</a>";
        $loggedin=true;
} elseif (isset($_POST["user"]) && isset($_POST["pass"])) {
    $password = "398fj289fj2389fj398fjhhds^&#hkseifw3893h#(&$$*838hjf";
    sleep(3); // do not bruteforce
    if ($_POST["user"]=='rmichaels') {
        if (strcmp($password, $_POST["pass"]) == 0) {
            $_SESSION['admin_logged_on'] = 'that is affirmative sir';
            echo "flag3{Y29udGludWVUT2Ntcw==}<br />Welcome, ".$_POST["user"] . "<br /><a href='cms.php?pagename=home'>IMF CMS</a>";
            $loggedin=true;
        } else {
            echo "Invalid password";
        }
    } else {
        echo "Invalid username.";
    }

}
if($loggedin===false) {
?>
<form method="POST" action="">
<label>Username:</label><input type="text" name="user" value=""><br />
<label>Password:</label><input type="password" name="pass" value=""><br />
<input type="submit" value="Login">
<!-- I couldn't get the SQL working, so I hard-coded the password. It's still mad secure through. - Roger -->
</form>
<?php } ?>
www-data@imf:/var/www/html/imfadministrator$
```

Şekil 2.2. 4: Broken Authentication Zafiyetine Sebep Olan Kod Bloğu

Strcmp fonksiyonu parametre olarak aldığı “pass” girdi alanına karşılık gelen değer ile kullanıcının girdiği parolayı karşılaştıracaktır. Karşılaştırma sonucu “0” ise başarılı bir giriş yapılacaktır. Ancak karşılaştırma sırasında bir hata alınması durumunda fonksiyonunun geri dönüş değeri “NULL” olacaktır. Böyle bir durumda geri dönüş değeri “==”şeklinde karşılaştırılırsa, bir uyarı mesajı verilir ve bununla birlikte bir zafiyet ortaya çıkabilir. Kurban makinede kullanılan strcmp fonksiyonunda hata olması durumunda “NULL==0=TRUE” şeklinde yorumlanarak doğru sonucu verecektir. Hataya sebebiyet vermesi için “pass[]” şeklinde dizi değişkeni kullanılmıştır.

```
Pretty  Raw  \n  Actions
 1 POST /imfadministrator/ HTTP/1.1
 2 Host: 10.0.2.5
 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
 5 Accept-Language: en-US,en;q=0.5
 6 Accept-Encoding: gzip, deflate
 7 Content-Type: application/x-www-form-urlencoded
 8 Content-Length: 25
 9 Origin: http://10.0.2.5
10 Connection: close
11 Referer: http://10.0.2.5/imfadministrator/
12 Cookie: PHPSESSID=4ftt3dgb1sr7q8palnf0d39u84
13 Upgrade-Insecure-Requests: 1
14
15 user=rmichaels&pass["sefagurkanmucahit"]
```

Şekil 2.2. 5: Zafiyet Bulunduran Site Sunucusuna Gönderilen Payload

2.2.3 Kimlik Doğrulama Zafiyetinin Önlenmesi

Kimlik doğrulama zafiyetlerini önlemek için şu önlemler alınabilir:

- Sağlam Parola Kuralları: Kullanıcıları güçlü ve karmaşık parolalar kullanmaya teşvik etmek için uygun parola politikaları uygulanmalıdır. Bu kurallar, uzunluk, büyük/küçük harf kullanımı, rakam ve özel karakter gereksinimlerini içerebilir.
- Çift Faktörlü Kimlik Doğrulama (MFA): Çift faktörlü kimlik doğrulama yöntemleri, kullanıcıların parola dışında başka bir doğrulama adımı atmalarını gerektirir. Bu, saldırganların sadece parolayı ele geçirmesi durumunda bile sistem erişimini engeller.
- Oturum Zamanlama ve Yönetiminin İyileştirilmesi: Oturum süreleri kısıtlanmalı ve kullanıcılar oturumlarını kapatmayı unutsalar bile otomatik olarak oturumlarını sonlandıracak mekanizmalar kullanılmalıdır. Ayrıca, oturum kimlikleri (session IDs) ve çerezler güvenli bir şekilde saklanmalıdır.
- Giriş Denemesi Sınırları: Kullanıcıların belirli bir süre içinde belirli bir sayıda giriş denemesi yapmalarına izin veren sınırlamalar getirilmelidir. Bu, brute force saldırılarını engellemeye yardımcı olur.
- Güvenlik Duvarları ve IPS/IDS Kullanımı: Güvenlik duvarları ve Sızma Önleme Sistemleri (IPS) / Sızma Algılama Sistemleri (IDS), kimlik doğrulama isteklerini izleyebilir ve anormal aktiviteleri tespit ederek saldırıları önleyebilir.
- Güvenlik Farkındalık Eğitimleri: Kullanıcılara güvenli parola uygulamaları, phishing saldırılarına karşı dikkat etme ve kimlik doğrulama önlemlerini kullanma konularında eğitimler verilmelidir.
- Yazılım Güncellemeleri: Web uygulamaları ve altta yatan yazılımlar düzenli olarak güncellenmeli ve güvenlik yamaları uygulanmalıdır. Bu, mevcut güvenlik açıklarının giderilmesine ve yeni zafiyetlerin ortaya çıkmasının önlenmesine katkı sağlar.

2.3 HASSAS VERİ SERGİLENMESİ

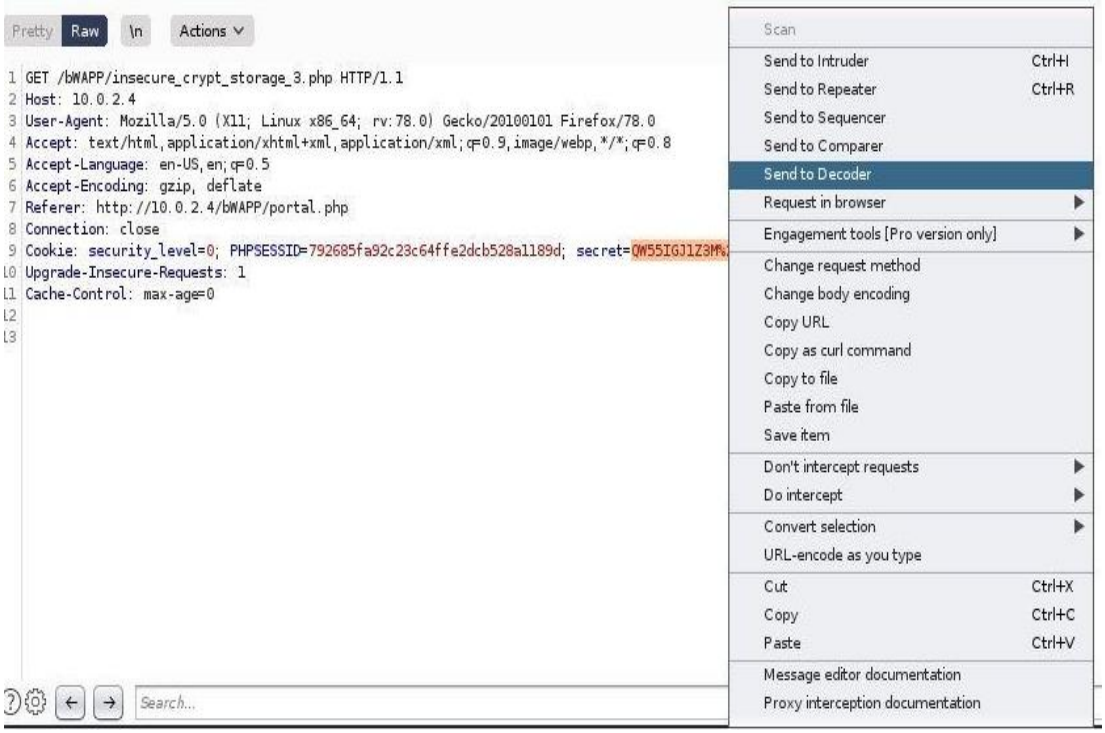
Hassas Bilgi İfşası (Sensitive Data Exposure), bir web uygulamasının hassas verileri (örneğin, kullanıcı parolaları, kredi kartı numaraları, kişisel detaylar vs.) yeterince korumadığı veya şifrelemediği durumlarda meydana gelen bir güvenlik açığıdır. Bu açık, kötü niyetli kişilerin bu verilere izinsiz erişim sağlamasına ve kullanıcıların gizli bilgilerinin ortaya çıkmasına yol açabilir. Hassas verilerin yeterince korunmaması, kullanıcı gizliliğini tehlikeye atabilir ve önemli sonuçlar doğurabilir.

2.3.1 Hassas Veri Zafiyetinin İstismar Edilmesi

Şekil 2.3. 1: bWapp Üzerinden Zafiyetli Web Sitesinin Seçilmesi

bWapp Üzerinden Zafiyetli Web Sitesinin Seçilmesi (A)

Şekil 2.3.1 (A)'da "Burp Suite" uygulaması kullanılarak, tarayıcımızın "Proxy" ayarlarını devreye sokup "Intercept is on" butonuna basıp, tarayıcı sayfamızı yenilendiğimiz zaman uygulama ekranımızda karşımıza çıkan "secret=QW55IGJ1Z3M%2F" değeri bir "Base64" kodu olabilir.



bWapp Üzerinden Zafiyetli Web Sitesinin Seçilmesi (B)

Şekil 2.3.1 (B)'de bu "secret" değeri "decoder" kısmına gönderilmiştir.

Şekil 2.3. 2: Burp Suit Yardımıyla Zafiyetin Bulunması

Şekil 2.3.2'de Decode ettikten sonra çıkan kod gösterilmiştir.

2.3.2 Hassas Veri Zafiyetinin Önlenmesi

Hassas bilgi ifşası zafiyetinin önlenmesi için şu adımlar izlenebilir:

- Veri Şifrelenmesi: Hassas veriler, saklanmadan veya aktarılmadan önce güçlü bir şekilde şifrelenmelidir. Bu, bilgilerin yetkisiz erişimden korunmasına yardımcı olur.
- Güçlü Kimlik Doğrulama: Kullanıcıların kimlik doğrulama süreçleri güçlendirilmeli ve ek güvenlik katmanları gibi çift faktörlü kimlik doğrulama yöntemleri uygulanmalıdır.
- Veri Azaltma: Uygulama, gereksiz hassas verilerin toplanmaması veya saklanmaması için gerekli önlemleri almalıdır. Yalnızca temel bilgiler toplanmalı ve işlenmelidir.
- Güvenli İletişim Yöntemleri: Hassas verilerin iletilmesi sırasında güvenli iletişim protokolleri kullanılmalıdır, örneğin, HTTPS.
- Güvenlik Duvarları ve Yazılımları: Ağ trafiği izlenerek ve anormal faaliyetler tespit edilerek hassas verilere erişimi engellemek için güvenlik duvarları ve yazılımları kullanılabilir.
- Güncelleme ve Yama Yönetimi: Web uygulaması ve altyapısı düzenli olarak güncellenmeli ve güvenlik yamaları düzenli olarak uygulanmalıdır.
- Kullanıcı Eğitimi: Kullanıcılar, güçlü parolalar kullanmaları ve güvenli internet uygulamaları konusunda düzenli olarak eğitilmelidir.
- Güvenlik Testleri: Uygulama, hassas veri koruma düzeyini sürekli olarak değerlendirmek için düzenli güvenlik testlerine tabi tutulmalıdır.

2.4 XML HARİCİ VARLIK

XML Harici Varlık (XML External Entity), XML belgelerinde dış kaynakları belgeye dahil etmek için kullanılan bir özelliktir. Dış varlık, XML belgesinin içine yerleştirilen veya atıfta bulunulan metin veya veri parçalarını ifade eder. Bu varlıklar, genellikle bir URL veya dosya yolu ile tanımlanır ve bir XML belgesine dahil edilerek içeriğe eklenir.

Bu özellik, XML belgesinin dinamik olarak başka kaynaklara bağlanmasına olanak tanır, böylece belge içeriği farklı kaynaklardan alınabilir veya güncellenebilir. Ancak, kötü niyetli durumlarda, dış varlıkların belgeye eklenmesi güvenlik riski oluşturabilir. Örneğin, XML Harici Varlık saldırıları, dış varlıkların kötü amaçlı içerikleri veya hassas bilgileri içerecek şekilde kullanılmasıyla gerçekleşebilir. Bu tür saldırılar, güvenlik açıklarına neden olabilir ve hassas bilgilerin ifşasına yol açabilir.

2.4.1 Genişletilebilir İşaretleme Dili

XML (Genişletilebilir İşaretleme Dili), bilgilerin yapılandırılmış biçimde taşınması için kullanılan bir işaretleme dilidir. Metin tabanlı bir formatta olduğundan, verilerin düzenlenmesi, saklanması ve taşınması için yaygın olarak kullanılır. XML, ağ üzerinde veri alışverişi yapmak, veri tabanlarında veri depolamak, yapılandırılmış veri alışverişi ve diğer birçok uygulama alanında kullanılmaktadır.

XML belgeleri, özel etiketlerle tanımlanmış yapılara sahiptir. Her etiket, belgedeki bir öğeyi veya veri parçasını temsil eder. Örneğin, bir XML belgesinde bir kitap kataloğunu temsil etmek için her kitap bir etiket içine yerleştirilebilir. Bu etiketlerin içeriği, kitap adı, yazarı, yayınevi vb. gibi bilgileri içerebilir. XML, verilerin hiyerarşik ve yapısal bir şekilde düzenlenmesine olanak tanır.

XML'in esnekliği, veri türlerini ve yapılarını tanımlamanın yanı sıra belirli bir uygulama veya kullanım senaryosuna uyacak şekilde genişletilebilmesine olanak tanır. Bu nedenle, "genişletilebilir" (extensible) adını almıştır. Örneğin, bir belgeyi belirli bir endüstri standardına veya belirli bir uygulama gereksinimine uyacak şekilde özelleştirebilirsiniz.

XML, HTML'den farklıdır; HTML, belgelerin web tarayıcılarında görüntülenmesi için kullanılan bir işaretleme dilidir, XML ise verilerin tanımlanması ve taşınması için kullanılır.

2.4.2 Belge Türü

Belge Türü (DOCTYPE), bir XML veya HTML belgesinin türünü ve sürümünü tanımlayan bir bildirimdir. Bir belgenin DOCTYPE bildirimi, belgenin hangi standart veya dilin izlendiğini belirtir ve belgenin tarayıcıda nasıl işleneceğini tanımlar. DOCTYPE bildirimi genellikle belgenin başında bulunur ve aşağıdaki formatta olabilir:

<!DOCTYPE html>

Bu örnek, bir HTML5 belgesinin DOCTYPE bildirimini temsil eder. HTML belgelerinde DOCTYPE bildirimi, belgenin hangi HTML sürümünü takip ettiğini belirtir ve tarayıcının belgeyi nasıl işleyeceğini belirler. Örneğin, HTML5 belgeleri için DOCTYPE bildirimi, tarayıcıya belgenin HTML5 standartlarına göre işlenmesini söyler. DOCTYPE bildirimi, tarayıcıya belgenin hangi standartlara uygun olduğunu bildirerek, doğru bir şekilde işlenmesini ve görüntülenmesini sağlar. Bu nedenle, DOCTYPE bildiriminin belge içinde doğru bir konumda olması önemlidir.

2.4.3 Belge Türü Tanımı

Belge Türü Tanımı (DTD), XML belgelerinin yapılarını ve niteliklerini belirleyen bir belge türüdür. DTD, XML belgelerinde kullanılan etiketlerin ve niteliklerin düzenini, belgedeki geçerli öğeleri ve öğelerin nasıl ilişkilendirildiğini tanımlar. Ayrıca, bir XML belgesinin doğruluğunu kontrol etmek ve belgenin geçerli olup olmadığını belirlemek için kullanılır.

DTD'ler, XML belgelerinin belirli bir yapıya sahip olmasını sağlar ve bu belgelerin nasıl biçimlenmesi gerektiğini açıklar. Örneğin, bir DTD, bir kitap kataloğu XML belgesinde her bir kitabın başlık, yazar, yayınevi vb. gibi öğelerini ve bu öğelerin birbiriyle olan ilişkilerini tanımlayabilir.

DTD'ler, bir XML belgesinin geçerliliğini sınayabilir. Belge, DTD tarafından tanımlanan düzene ve kurallara uygunsa, belge geçerli kabul edilir. Aksi takdirde, belge geçersiz kabul edilir ve DTD tarafından belirtilen standartlara uymadığı için işlenemez.

2.4.4 Varlık

Bir varlık (Entity), bir doküman içinde bulunan ve genellikle tekrar kullanılabilir ve değiştirilebilir bir bölümü ifade eder. XML belgelerinde, bir varlık, bir karakter, bir dize veya daha karmaşık bir yapı olabilir. XML'de, bir varlık "&" işaretiyle başlar ve ";" ile sonlanır. XML belgelerinde, bu tür varlıklar, belge içindeki özel karakterleri temsil etmek için kullanılır ve XML belgesinin geçerliliğini sağlamak için önemlidir. Varlıklar ayrıca XML belgelerinde yeniden kullanılabilir yapılar veya verileri tanımlamak için de kullanılabilir. Örneğin, bir XML belgesinde sık sık kullanılan bir ifadeyi bir varlık ile tanımlamak, belgenin okunabilirliğini artırabilir ve tekrar eden kodları azaltabilir.

2.4.5 XXE Zafiyetinin Oluşması ve İstismar Edilmesi

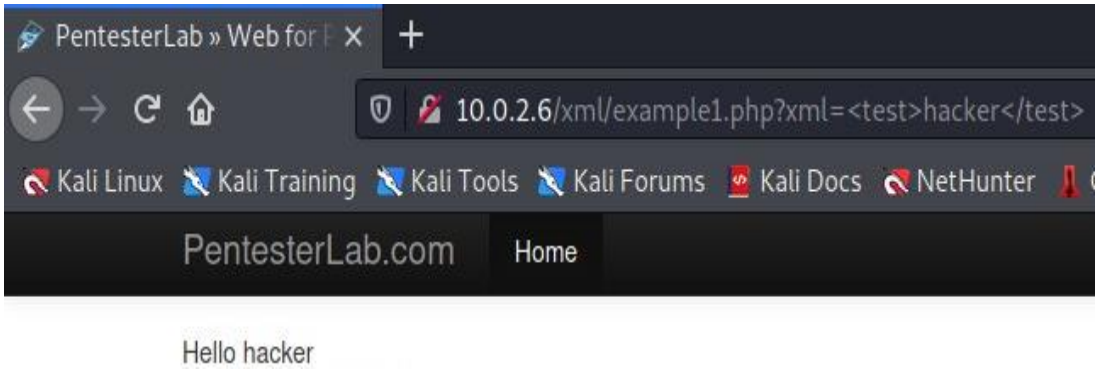
XXE (XML External Entity) açığı, XML işleme süreçlerinde ortaya çıkan bir güvenlik zafiyetidir. Bu açık, bir saldırganın XML dış varlıklarını kötüye kullanarak sunucuya izinsiz erişim sağlamasına veya hassas verilere ulaşmasına neden olabilir.

XXE zafiyeti genellikle bir XML belgesinin işlenmesi sırasında görülür. XML'de, belgeyi oluşturan XML dosyasına dış kaynaklardan veri almak için referanslar içeren dış varlıklar bulunur. Bu varlıklar, belge işlendiğinde belirtilen URL veya dosya yolu aracılığıyla yüklenir. Ancak, bu durum, saldırganların dış varlıkları kötü niyetli amaçlar için kullanmasına olanak tanır. XXE zafiyeti genellikle şu nedenlerden kaynaklanır:

- Varsayılan Dış Varlık Ayarları: XML işleyici ayarlarının varsayılan olarak dış varlıkları işlemeye izin vermesi durumunda, saldırganlar dış varlıkları kötü amaçlı içerikleri yüklemek için kullanabilirler.

- Kullanıcı Girdilerinin Yeterince Filtrelenmemesi: Kötü niyetli kullanıcı girişlerinin yeterince filtrelenmemesi durumunda, saldırganlar dış varlıkları kontrol edebilir ve kötü amaçlı içerikleri yüklemek için kullanabilirler.
- Zayıf XML İşleyici Ayarları: XML işleyici ayarlarının zayıf veya hatalı yapılandırılması, saldırganların dış varlıkları kötü amaçlar için kullanmasına imkân tanır.

XXE (XML External Entity) zafiyetinin istismarı, saldırganların XML dış varlıklarını kötüye kullanarak hedef sistemlere yetkisiz erişim sağlamak veya hassas bilgilere erişmek için kullanabilecekleri çeşitli yöntemler içerir.



Şekil 2.4. 1: XXE Zafiyeti Barındıran Sitenin Anasayfa Çıktısı

Şekil 2.4.1'de Zafiyet bulunduran makine olarak Pentester Lab tarafından geliştirilen Web For Pentester zafiyetli makinayı kullanılmıştır. Url incelendiğinde xml adında bir parametre aldığını ve test tagleri arasında yazılan verinin xml parametresine atanarak sayfaya basıldığı gösterilmiştir.

Şekil 2.4. 2: XXE Zafiyeti Barındıran Sitenin Passwd Çıktısı

Şekil 2.4.2'de gözüken bu durum gerekli payload kullanılarak gönderildiğinde zafiyet ortaya çıkmaktadır.

2.4.5 XXE Zafiyetinin Önlenmesi

XXE Zafiyetinin önlenmesi için aşağıdaki güvenlik önlemleri yapılmalıdır. Bunlar:

- XML İşleyici Ayarlarının Güvenli Hale Getirilmesi: XML işleyici yapılandırmalarının güvenli bir şekilde ayarlanması önemlidir. Bu, dış varlıkların etkin bir şekilde devre dışı bırakılmasını sağlar.
- Giriş Doğrulama ve Filtreleme: XML belgelerinde kullanıcı girişlerinin titizlikle doğrulanması ve filtrelenmesi gerekmektedir. Bu, kötü niyetli XML dış varlıklarının engellenmesine yardımcı olur.
- XML Güvenlik Kütüphanelerinin Kullanılması: Güvenlik odaklı XML kütüphanelerinin tercih edilmesi, XXE zafiyetlerini engellemek için etkili bir yöntemdir.
- DTD (Belge Türü Tanımı) Kullanmaktan Kaçınma: XXE açıklarını önlemek için, DTD kullanımından kaçınılmalı ve alternatif belge tanımlama yöntemleri tercih edilmelidir.
- XML Parçalama Kütüphanelerinin Güncellenmesi: Kullanılan XML parçalama kütüphanelerinin güncel ve güvenli olduğundan emin olunmalıdır. Güncellenmiş kütüphanelerin kullanılması, güvenlik açıklarının kapatılmasına yardımcı olur.
- XML İşleme İşlemlerinin Kısıtlanması: XML işleme işlemlerinin sadece güvenilir ve gerekli kaynaklardan alınan belgelerle sınırlı olması gerekmektedir.
- Güvenlik Denetimleri ve Testleri: Uygulamanın düzenli olarak güvenlik denetimlerine ve penetrasyon testlerine tabi tutulması, XXE zafiyetlerinin tespit edilmesine ve giderilmesine yardımcı olur.

Bu önlemler, XXE zafiyetlerini azaltmak ve uygulamaların güvenliğini artırmak için etkili bir yöntem sağlar.

2.5 KIRIK ERİŞİM KONTROLÜ

Kırık Erişim Kontrolü (Broken Access Control), web uygulamalarında kullanıcıların veya sistem bileşenlerinin belirli kaynaklara yetkisiz erişim sağlayabilmesine yol açan bir güvenlik açığıdır. Bu zafiyet, web uygulamasının izinleri ve erişim kontrollerini uygun şekilde uygulamadığı zaman ortaya çıkar.

Örneğin, bir web uygulamasında kullanıcılar arasında yetki farkları bulunuyorsa ve bu yetkiler yeterince denetlenmiyorsa, kullanıcılar normalde ulaşamayacakları kaynaklara erişebilirler. Bu durumda, kullanıcılar hassas verilere, yönetici arayüzlerine veya diğer önemli bileşenlere yetkisiz erişim sağlayabilirler.

2.5.1 Kırık Erişim Kontrolü Zafiyetinin Oluşması

Kırık erişim kontrolleri genellikle aşağıdaki nedenlerden dolayı oluşur:

- Zayıf Yetkilendirme ve Doğrulama Yöntemleri: Uygulama, kullanıcıların kimliklerini doğrulamak ve erişim izinlerini belirlemek için yetersiz veya hatalı yöntemler kullanıyorsa, yetki kontrolleri zayıf olabilir.
- Doğru İzin Kontrolünün Eksikliği: Uygulama, belirli kaynaklara erişim izinlerini doğru bir şekilde denetlemediğinde veya uygulamadığında, kullanıcılar yetkisiz erişim sağlayabilirler.
- Eksik İzleme ve Günlükleme: Uygulama, erişim etkinliklerini izlemiyor veya günlükleme yapmıyorsa, yetki ihlalleri tespit edilemez ve düzeltilmez.
- Varsayılan Ayarlar ve Zayıf Yapılandırma: Uygulamanın varsayılan ayarları veya yapılandırması, kullanıcıların yetkisiz erişim sağlamasına olanak tanıyabilir. Örneğin, bir yönetici arayüzünün varsayılan parolasının değiştirilmemesi, saldırganların sisteme kolayca erişim sağlamasına neden olabilir.

2.5.2 Kırık Erişim Kontrolü Zafiyetinin İstismarı

Kırık Erişim Kontrolü Zafiyetinin istismarı, yetkilendirme veya kimlik doğrulama mekanizmalarındaki hatalardan veya eksikliklerden yararlanarak belirli kaynaklara

yetkisiz erişim sağlamayı amaçlar. Bu zafiyetler genellikle aşağıdaki yöntemlerle gerçekleştirilir:

- Yetkisiz Taleplerin Yapılması: Saldırganlar, yetkisiz olarak korunan bir kaynağa erişmek için kimlik doğrulama adımlarını atlamak veya geçersiz kimlik bilgileriyle talepler göndermek gibi yöntemler kullanabilirler.
- URL Manipülasyonu: Uygulamanın URL yapısını veya parametrelerini manipüle ederek, yetkisiz olarak erişilebilecek kaynaklara erişmeye çalışabilirler. Örneğin, bir kullanıcının normalde erişemeyeceği bir sayfaya erişmek için URL'yi değiştirebilirler.
- Zayıf Oturum Yönetimi: Oturum kimlikleri üzerinde yetersiz kontroller veya tahmin edilebilir oturum kimlikleri kullanarak, yetkisiz oturumların ele geçirilmesi ve kullanılması söz konusu olabilir.
- Hata Mesajlarından Faydalanma: Uygulamanın hata mesajları, saldırganlara belirli hedeflere erişim sağlamak için bilgi sağlayabilir. Örneğin, bir yetkilendirme hatası mesajı, saldırganlara hangi kullanıcı adlarının geçerli olduğunu öğrenme imkânı verebilir.
- Doğrudan Nesne Başvurusu (Direct Object Reference): Uygulamanın içindeki kaynakları belirlemek ve doğrudan onlara erişmek için referanslar kullanılabilir. Bu, yetkilendirme kontrollerinin atlandığı veya zayıf olduğu durumlarda kullanılabilir.

2.5.3 Kırık Erişim Kontrolü Zafiyetinin Önlenmesi

Kırık Erişim Kontrolü Zafiyetini engellemek için farklı önlemler alınabilir. İlk olarak, sağlam yetkilendirme ve kimlik doğrulama sistemlerinin kullanılması esastır. Bu, kullanıcıların kimliklerini doğrulamak ve erişim izinlerini belirlemek için güvenilir ve sağlam sistemlerin tercih edilmesini sağlar. Örneğin, güçlü parola politikaları ve iki faktörlü kimlik doğrulama gibi yöntemler, uygulamanın güvenliğini artırabilir.

İkinci olarak, doğru izin kontrolünün uygulanması kritik önem taşır. Uygulama, belirli kaynaklara erişim izinlerini doğru bir şekilde denetlemeli ve uygulamalıdır. Bu, her kullanıcının sadece gereksinim duyduğu en düşük erişim izinlerine sahip olduğundan

emin olmayı içerir. Bu sayede, kullanıcılar yetkilerini aşamaz ve yetkisiz erişim sağlayamazlar.

Üçüncü olarak, gelişmiş oturum yönetimi hayati önem taşır. Oturum kimlikleri üzerinde güvenlik kontrolleri sağlanmalı ve güvenlik önlemleri alınmalıdır. Güvenli oturum yönetimi politikaları uygulanmalı ve oturum kimlikleri güçlü bir şekilde korunmalıdır. Bu, oturumlar üzerinde yetkisiz erişimi engelleyebilir.

Ayrıca, hata mesajlarından bilgi sızdırmamak önemlidir. Uygulamanın hata mesajları, kullanıcıya en az miktarda bilgi sağlamalı ve özellikle yetki ile ilgili hataların detayları dikkatlice ele alınmalıdır. Bu, saldırganların zayıf noktaları tespit etmesini güçleştirir.

Son olarak, düzenli güvenlik testleri ve denetimler yapılmalıdır. Bu, uygulamanın güvenliğini değerlendirmek ve potansiyel zafiyetleri tespit etmek için önemlidir. Güvenlik duvarları ve izleme araçları kullanılarak yetkisiz erişim girişimleri izlenebilir ve engellenebilir. Bu tür araçlar kullanılarak erken uyarılar alınabilir ve zafiyetler önceden önlenmiş olur.

2.6 GÜVENLİK YANLIŞ YAPILANDIRMASI

Güvenlik Hatalı Yapılandırma (Security Misconfiguration), sistem, sunucu veya ağda mevcut güvenlik ayarlarının yanlış ya da eksik yapılandırılması sonucunda ortaya çıkan bir güvenlik açığıdır. Bu tür hatalar genellikle sistem yöneticileri veya geliştiriciler tarafından yapılan yapılandırma hataları, varsayılan ayarların değiştirilmemesi veya güncellemelerin ihmal edilmesi gibi sebeplerle meydana gelir.

Güvenlik hatalı yapılandırmaları, saldırganların sistemlere yetkisiz erişim sağlaması, hassas verilerin ifşa edilmesi, hizmet kesintileri veya kötü amaçlı yazılımların bulaşması gibi sonuçlara neden olabilir. Bu tür bir güvenlik açığı, kuruluşları siber saldırılara ve veri sızıntılarına karşı savunmasız hale getirebilir. Bu nedenle, doğru yapılandırma ve güvenlik ayarlarının sağlanması büyük önem taşır.

2.6.1 Samba

Samba, Linux ve diğer UNIX benzeri işletim sistemlerinde Windows işletim sistemi ağ dosya ve yazıcı paylaşım protokollerini uygulamak için kullanılan bir yazılım paketidir. Samba, SMB/CIFS (Server Message Block/Common Internet File System) protokollerini destekleyerek Linux ve diğer UNIX tabanlı sistemlerin, Windows tabanlı sistemlerle etkileşimde bulunmasını sağlar.

Bu yazılım paketi, bir ağda Linux veya UNIX sistemlerinde bir dosya sunucusu veya yazıcı sunucusu olarak hizmet verebilir ve Windows istemcilerinin bu sunuculara erişmesine olanak tanır. Samba, kullanıcıların dosya ve yazıcıları ağ üzerinde paylaşmasını, ağ kullanıcılarının dosya ve yazıcı kaynaklarına erişmesini ve Windows ve Linux sistemleri arasında dosya paylaşımını kolaylaştırır.

Samba, aynı zamanda Active Directory (AD) ile entegrasyon ve dosya paylaşımı için güvenlik ve yetkilendirme özellikleri sağlayarak kurumsal ağlar için güçlü bir çözüm sunar. Bu, Linux ve UNIX sistemlerini, Windows tabanlı ağlarla entegre etmek ve dosya paylaşımı için güvenli bir ortam sağlamak için kullanılır.

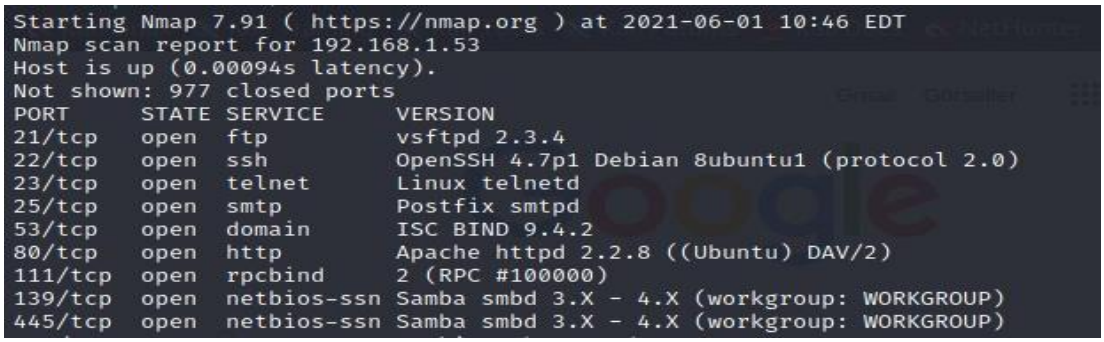
Genel olarak, Samba, açık kaynaklı bir yazılım paketi olarak, Linux ve UNIX sistemlerini Windows ağlarıyla entegre etmek ve ağ dosya ve yazıcı paylaşımı sağlamak için yaygın bir şekilde kullanılan bir araçtır.

2.6.2 Güvenlik Yanlış Yapılandırma Zafiyetinin Oluşması

Güvenlik yanlış yapılandırması, çoğunlukla sistem yöneticileri veya geliştiriciler tarafından yapılandırma hataları, varsayılan ayarların değiştirilmemesi veya güncellemelerin ihmal edilmesi gibi sebeplerle meydana gelir. Örneğin, bir sunucunun güncel olmayan veya zayıf şekilde yapılandırılmış bir işletim sistemi ile çalıştırılması, gereksiz servislerin veya bağlantı noktalarının açık bırakılması veya varsayılan şifrelerin kullanılması gibi durumlar güvenlik yanlış yapılandırmasına sebep olabilir. Bunun yanı sıra, bir uygulamanın güvenlik ayarlarının düzgün bir şekilde yapılandırılmaması veya güvenlik açıklarının düzeltilmemesi de bu tür bir zafiyetin oluşmasına neden olabilir. Bu tip durumlar, saldırganların sistemlere kolayca erişim sağlaması veya hassas verilere yetkisiz erişim elde etmesi anlamına gelir.

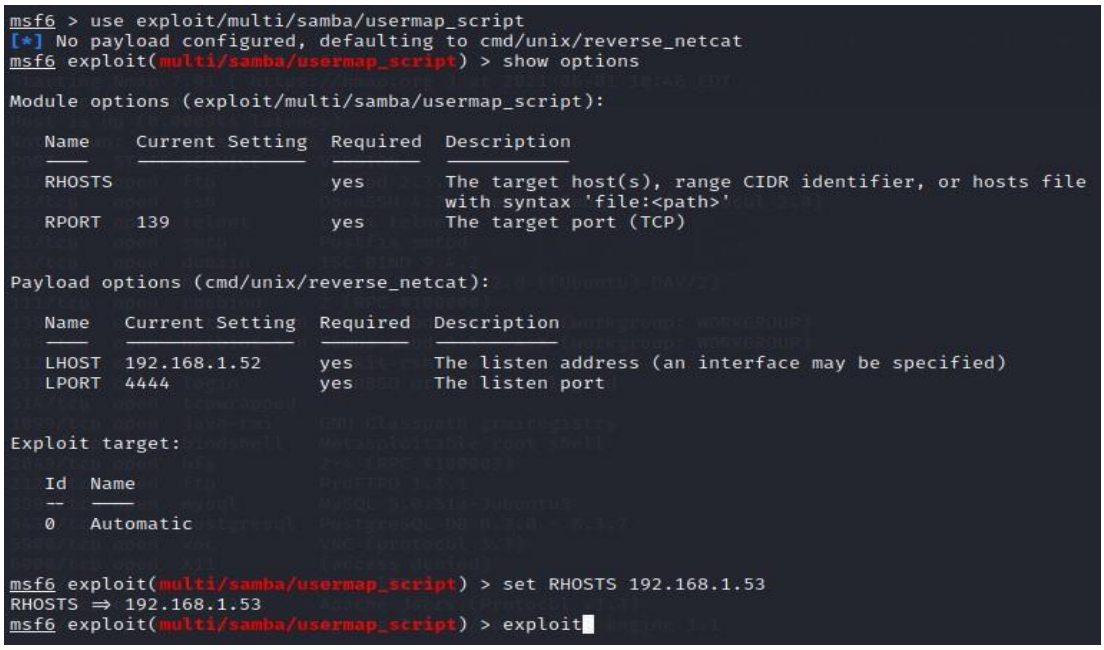
2.6.3 Güvenlik Yanlış Yapılandırma Zafiyetinin İstismarı

Arbitrary File Access Samba zafiyeti üzerinden örnek uygulama aşağıda gösterilmiştir. Bu zafiyet, sambanın bir SMB client tarafından sunulan dizin yollarının doğrulama yöntemindeki eksiklikten dolayıdır. Saldırganın kendi dizin yolunu belirleyerek istediği dosyayı çalıştırmasına olanak sağlar. Sambaya aktarılan dosyalar ve dizin adları unic_covert() ve check_name() fonksiyonlarıyla dönüştürülme işlemi yapılır ve kontrolü sağlanır. Unix_clean_name() fonksiyonu “//” ve “./” karakterlerini kaldırır. Check_name() fonksiyonu dönüştürülmüş dosya adının doğruluğunu kontrol eder ve reduce_name() fonksiyonunu çağırır ve iki kez Unix_clean_name() fonksiyonu çağırılır. Bu işlemde saldırgan istediği dizini enjekte edebilir. Aşağıdaki uygulamada saldırgan makine olarak Kali Linux, istismar edilen makine olarak metasploitable-2 kullanılmıştır.



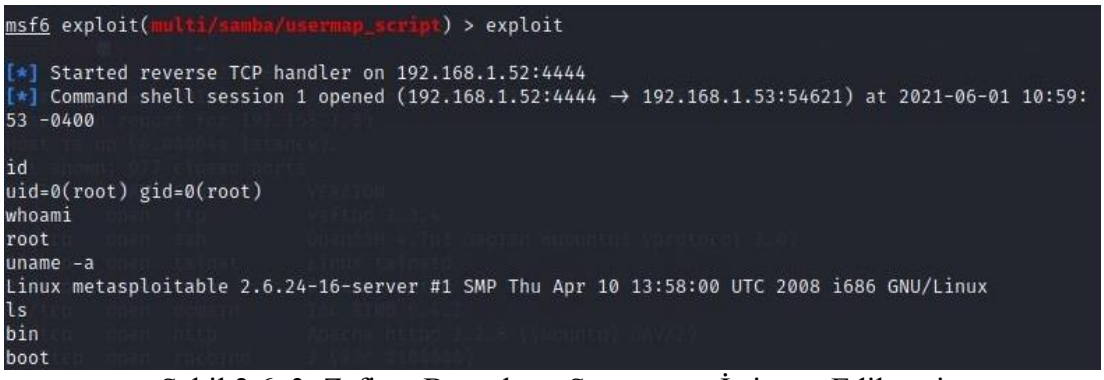
Şekil 2.6. 1: Security Misconfiguration Zafiyeti Barındıran Sunucunun Çıktısı

Şekil 2.6.1'de Nmap taraması sonucunda portun açık olduğu ve servis bilgileri elde edilmiştir.



Şekil 2.6. 2: Zafiyeti Barındıran Sunucuya Değer Seçimi

Şekil 2.6.2'de Exploit işlemi gerçekleştirilerek komut çalıştırılmıştır.



Şekil 2.6. 3: Zafiyet Barındıran Sunucunun İstismar Edilmesi

Şekil 2.6.3'de Zafiyeti barındıran sunucunun istismar edildiği gözükmektedir.

2.6.4 Güvenlik Yanlış Yapılandırma Zafiyeti Önlenmesi

Güvenlik yanlış yapılandırması zafiyetlerini önlemek için bir dizi strateji izlenmelidir. İlk olarak, şirketler, kapsamlı güvenlik politikaları ve standartları belirlemelidir. Bu politika ve standartlar, tüm sistemlerin ve uygulamaların güvenlik gereksinimlerine uygun olarak yapılandırılmasını temin eder. Bununla birlikte, güvenli yapılandırma kılavuzları kullanılmalı ve sistem yöneticileri ile geliştiricilere doğru yapılandırma adımlarını sağlamak için eğitim ve kaynaklar sunulmalıdır. Ayrıca, sistemlerin ve uygulamaların güncel olması ve güvenlik yamalarının düzenli olarak uygulanması da hayati önem taşır. Gereksiz hizmetlerin kapatılması veya devre dışı bırakılması, güçlü parola politikalarının uygulanması ve dosya/klasör izinlerinin düzenli olarak denetlenmesi gibi tedbirler de alınmalıdır. Güvenlik testleri ve denetimlerin periyodik olarak yapılması da zafiyetlerin tespit edilmesi ve düzeltilmesi için kritiktir. Son olarak, sistem yöneticileri ve geliştiriciler, güvenlik alanında eğitim ve farkındalık programlarına katılarak doğru güvenlik uygulamalarını benimsemelidirler. Bu stratejilerin bir araya gelmesi, güvenlik yanlış yapılandırması zafiyetlerini azaltmak ve organizasyonların siber saldırılara karşı daha güvenli hale gelmesini sağlamak için gereklidir.

2.7 SİTELER ARASI KOMUT ÇALIŞTIRMA

Cross-Site Scripting (XSS), web uygulamalarında oldukça yaygın görülen ve kullanıcıların tarayıcılarında zararlı kodların çalışmasına olanak tanıyan bir güvenlik açığıdır. Bu tip saldırılar, saldırganın güvenilmeyen verileri, genellikle JavaScript gibi betik dillerini ekleyerek gerçekleştirdiği durumlarda ortaya çıkar. XSS saldırıları, kullanıcı girişleri, URL parametreleri veya diğer veri alanları aracılığıyla gerçekleştirilebilir. XSS saldırılarının üç temel çeşidi vardır:

1. Depolama (Stored) XSS: Bu tür saldırılarda, saldırgan, web uygulamasının veri tabanına zararlı betikleri ekler. Bu betikler daha sonra bu verileri gören diğer kullanıcılara sunulduğunda, tarayıcıda çalışır ve saldırganın istediği eylemleri gerçekleştirir.
2. Yansıma (Reflected) XSS: Bu tür saldırılarda, saldırgan, kullanıcının tarayıcısına zararlı bir URL veya form alanı aracılığıyla betikleri ekler. Sunucu bu betikleri işler ve kullanıcının tarayıcısına geri gönderir. Kullanıcı URL'yi veya formu açtığında, betik tarayıcıda çalışır.
3. DOM XSS: Güvenlik açığının sunucu tarafında bulunan kod yerine istemci tarafında bulunmasıdır.

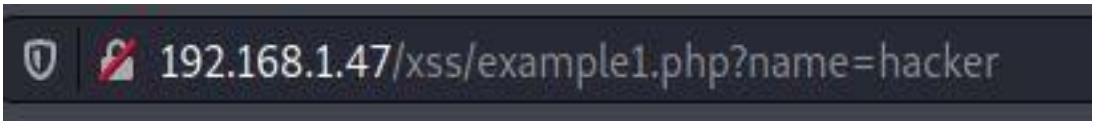
XSS saldırıları, kullanıcıların oturum bilgilerini çalmak, kullanıcıları zararlı web sitelerine yönlendirmek, sayfa içeriğini değiştirmek veya kullanıcıları başka zararlı eylemlere teşvik etmek gibi farklı kötü amaçlarla kullanılabilir.

2.7.1 Siteler Arası Komut Çalıştırma Zafiyetinin İstismarı

Cross Site Scripting (XSS) zafiyetinin istismarı, bir saldırganın kötü niyetli kodları web uygulamasına enjekte etmesi ve bu kodların son kullanıcıların tarayıcılarında çalıştırılmasıyla gerçekleşir. Bu tür bir saldırı, genellikle kullanıcı giriş alanları, form alanları veya URL parametreleri gibi web uygulamasının giriş noktaları aracılığıyla gerçekleştirilir. Saldırganlar, bu giriş noktalarına zararlı betikleri ekleyerek veya URL'lerdeki parametreleri manipüle ederek, tarayıcılarda çalıştırılmak üzere kodları enjekte ederler. Bu kötü amaçlı betikler, tarayıcılarda çalıştırıldığında, saldırganların istediği eylemleri gerçekleştirebilir, örneğin oturum bilgilerini ele geçirebilir veya

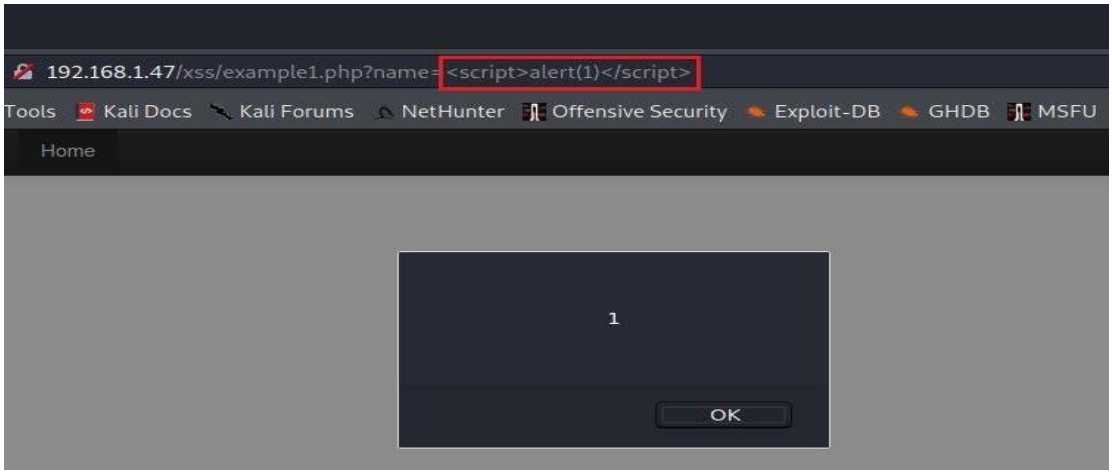
kullanıcıları kötü amaçlı web sitelerine yönlendirebilir. Ayrıca, saldırganlar, depolama XSS veya yansıma XSS gibi yöntemler kullanarak, zararlı betikleri web uygulamasının veri tabanında saklayabilir veya web uygulamasının yanıtlarında bulunan URL'lerde veya veri alanlarında kodları enjekte edebilirler. Bu yöntemler aracılığıyla, XSS zafiyeti kullanılarak web uygulamaları üzerinde çeşitli kötü amaçlar gerçekleştirilebilir.

Şekil 2.7. 1: Web For Pentester Ana Menüsü



Şekil 2.7. 2: XSS Zafiyeti (A)

Şekil 2.7.2 (A)'da Gelen HTTP'i kontrol edildiğinde "hacker" adında bir kullanıcı adı gözükmektedir. Payload yazılmasını engelleyecek herhangi bir engel bulunmadığı bilinmektedir. Daha sonra "name" requestine bir script dizini yazılmıştır.



XSS Zafiyeti (B)

Şekil 2.7.2 (B)'de Göründüğü gibi "<script>alert(1)</script>" payload girildiğinde ekranda "alert" gözükmektedir.

2.7.2 Siteler Arası Komut Çalıştırma Zafiyetinin Önlenmesi

Cross Site Scripting (XSS) açıklarını önlemek için çeşitli tedbirler alınabilir. İlk olarak, kullanıcı girişlerinin doğrulanması ve filtrelenmesi gereklidir. Bu, potansiyel olarak zararlı olan HTML etiketleri, JavaScript kodları ve diğer tehlikeli unsurların kullanıcı girişlerinden temizlenmesini sağlar. Ayrıca, web uygulamalarında güvenlik duvarları (WAF) kullanılabilir. Güvenlik duvarları, zararlı istekleri tespit ederek engelleyebilir ve böylece XSS saldırılarını önler. Doğru kodlama uygulamalarının benimsenmesi de kritik öneme sahiptir; özellikle kullanıcı girişlerinin güvenli bir şekilde işlenmesi ve sunucuya iletilmesi gerekmektedir. Content Security Policy (CSP) kullanımı da etkili bir önlemdir; CSP, tarayıcıların yürütülebilecek kaynakları denetlemesine olanak tanır, bu da XSS saldırılarını engelleyebilir. HTTPS kullanımı, iletişimin şifrelenmesini sağlayarak saldırganların ağ üzerindeki verilere erişimini zorlaştırır. Son olarak, web uygulamalarının ve kullanılan kütüphanelerin güvenlik güncelleştirmelerinin düzenli olarak izlenmesi ve uygulanması önemlidir. Bu önlemler, XSS açıklarını önlemek ve web uygulamalarını daha güvenli hale getirmek için etkili bir yol sunar.

2.8 GÜVENLİK DESERİALİZASYON

Güvensiz Deserializasyon (Insecure Deserialization), bir uygulamanın gelen verileri işlerken yeterince güvenlik kontrolleri sağlamaması sonucu ortaya çıkan bir güvenlik açığıdır. Deserializasyon, veri nesnelerinin saklanması veya iletilmesi için kullanılan veri biçimlerinden (genellikle JSON veya XML gibi) gerçek nesnelere dönüştürülmesini sağlayan bir işlemdir. Saldırganlar, bu işlemi kötüye kullanarak, uygulamanın güvenlik önlemlerini atlayabilir ve çeşitli saldırılar gerçekleştirebilirler.

Güvensiz deserializasyon saldırıları, genellikle şu şekillerde ortaya çıkar:

1. Veri Bütünlüğünün Bozulması: Saldırganlar, gelen verilerde değişiklik yaparak deserializasyon işlemini etkileyebilirler. Bu durum, uygulamanın istemeden zararlı işlemler gerçekleştirmesine veya yanıt olarak gizli bilgileri ifşa etmesine neden olabilir.
2. Doğrudan Nesne Başvurusu (Direct Object Reference): Saldırganlar, deserializasyon sürecinde kontrolsüz bir şekilde nesneleri oluşturarak, uygulamanın güvenlik kontrollerini atlayabilir ve yetkilendirme önlemlerini aşabilirler. Bu durum, yetkisiz erişim veya diğer kötü amaçlı faaliyetler için kullanılabilir.
3. Yürütülebilir Kod Enjeksiyonu: Saldırganlar, gelen verilere yürütülebilir kod parçaları ekleyerek, uygulamanın sunucu tarafında kod çalıştırmasını sağlayabilirler. Bu, sunucuda istenmeyen işlemleri gerçekleştirmek veya saldırganın istediği kodları çalıştırmak için kullanılabilir.

2.8.1 Serileştirme ve Deserializasyon

Serileştirme, bir nesnenin veri akışı veya dosya gibi saklama formatlarına dönüştürülmesi işlemidir. Bu süreç, genellikle veri iletilirken veya depolanırken nesnenin durumunu ve yapısını belirli bir biçime dönüştürür. Serileştirme, nesnelerin ağ üzerinden taşınması veya veri tabanında saklanması gibi durumlarda yaygın olarak kullanılır.

Deserileştirme, serileştirme işleminin tersidir. Yani, bir veri akışı veya dosyadan alınan bilgilerin orijinal nesne formuna dönüştürülmesidir. Bu işlem genellikle, serileştirme işlemi öncesindeki nesnenin aynısını oluşturur. Deserileştirme, veri akışı veya dosyadan alınan verilerin, program tarafından kullanılabilir nesneler haline getirilmesini sağlar.

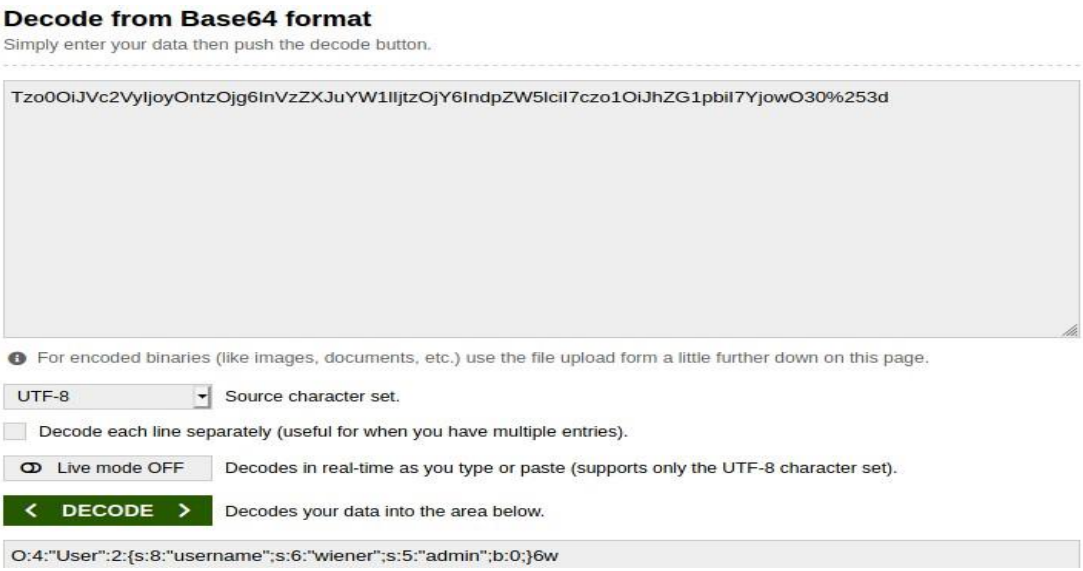
Serileştirme ve deserializasyon, nesne tabanlı programlamada veri taşıma, iletişim ve kalıcı saklama gibi birçok senaryoda kullanılır. Örneğin, bir nesnenin ağ üzerinde başka bir uygulamaya taşınması veya bir veri tabanında saklanması için serileştirme kullanılabilirken, ağdan gelen verilerin tekrar nesne formuna dönüştürülmesi için deserializasyon kullanılır. Bu işlemler, veri bütünlüğünün korunması ve farklı sistemler arasında veri alışverişi sağlamak için kritiktir.

2.8.2 Güvensiz Deserializasyon Zafiyetinin İstismarı

Bu zafiyetin istismar edilmesi için saldırgan makine olarak Kali Linux kullanılmıştır. İstismar edilebilmesi için Web Security Academy sunucusu kullanılmıştır. Bu zafiyet için Web Security Academy wiener kullanıcı adını ve peter parolasının verildiği bilinmektedir. Serialization edilmiş verileri deserialization işlemine tabi tutarak admin kullanıcısı olunmaya çalışılmıştır.

Şekil 2.8. 1: Insecure Deserialization Zafiyeti Barındıran Login Sayfası Çıktısı

Şekil 2.8.1'de Giriş yapıldıktan sonra request incelendiğinde cookie session bilgisinin base64 ile encode edildiği gözükmektedir.

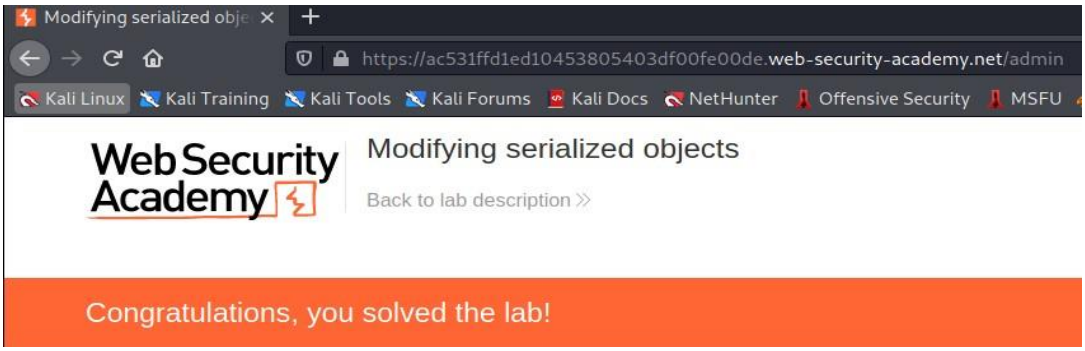


Şekil 2.8. 2: Zafiyeti Barındıran Sitenin Session Bilgisinin Encode Edilmesi

Şekil 2.8.2'de Encode edilen veri decode edilerek oturum bilgisinin serializate edildiği gözükmektedir. Admin değişkeni default olarak binary formatta 0 olduğu gözükmektedir.

Şekil 2.8. 3: Zafiyeti Barındıran Sitenin Admin Girişi Çıktısı

Şekil 2.8.3'de Değer 1 yapılarak base64 formatında encode edildiği gözükmektedir.



Şekil 2.8. 4: Insecure Deserialization Zafiyeti Barındıran Siteye Başarılı Giriş

Şekil 2.8.4'de Gönderilen request sonucunda admin olarak giriş sağlanarak zafiyetin istismar edildiği gösterilmiştir.

2.8.3 Güvensiz Deserializasyon Zafiyetinin Önlenmesi

Güvenli olmayan deserializasyon zafiyetlerini engellemek için çeşitli adımlar atılabilir. Başlangıçta, güvenilir serileştirme ve deserializasyon kütüphaneleri tercih edilmelidir. Güvenlik odaklı kütüphaneler, kötü niyetli verilerin güvenle işlenmesini sağlamak için kritiktir. Ek olarak, giriş doğrulama ve yetkilendirme kontrolleri uygulanmalıdır. Gelen veriler titizlikle doğrulanmalı ve gereksiz veya potansiyel tehlikeli verilere izin verilmemelidir. Bu, kötü niyetli verilerin deserializasyon sürecinde istismar edilmesini önleyebilir. Veri bütünlüğünün sağlanması da hayati öneme sahiptir. Gelen verilerin bütünlüğü doğrulanmalı ve değiştirilmiş veya bozulmuş verilerin deserializasyon sürecine tabi tutulması engellenmelidir. İhtiyaç halinde, güvenlik duvarları (WAF) kullanılabilir. WAF'ler, kötü niyetli girişlerin tespit edilmesini ve engellenmesini sağlayarak güvensiz deserializasyon saldırılarını engelleyebilir. Son olarak, güncellemelerin düzenli olarak takip edilmesi ve uygulanması gereklidir. Serileştirme ve deserializasyon işlemlerini yapan kütüphaneler ve bileşenler, güvenlik güncelleştirmeleriyle düzenli olarak güncellenmelidir.

2.9 BİLİNEN GÜVENLİK AÇIĞINA SAHİP BİLEŞENLERİ

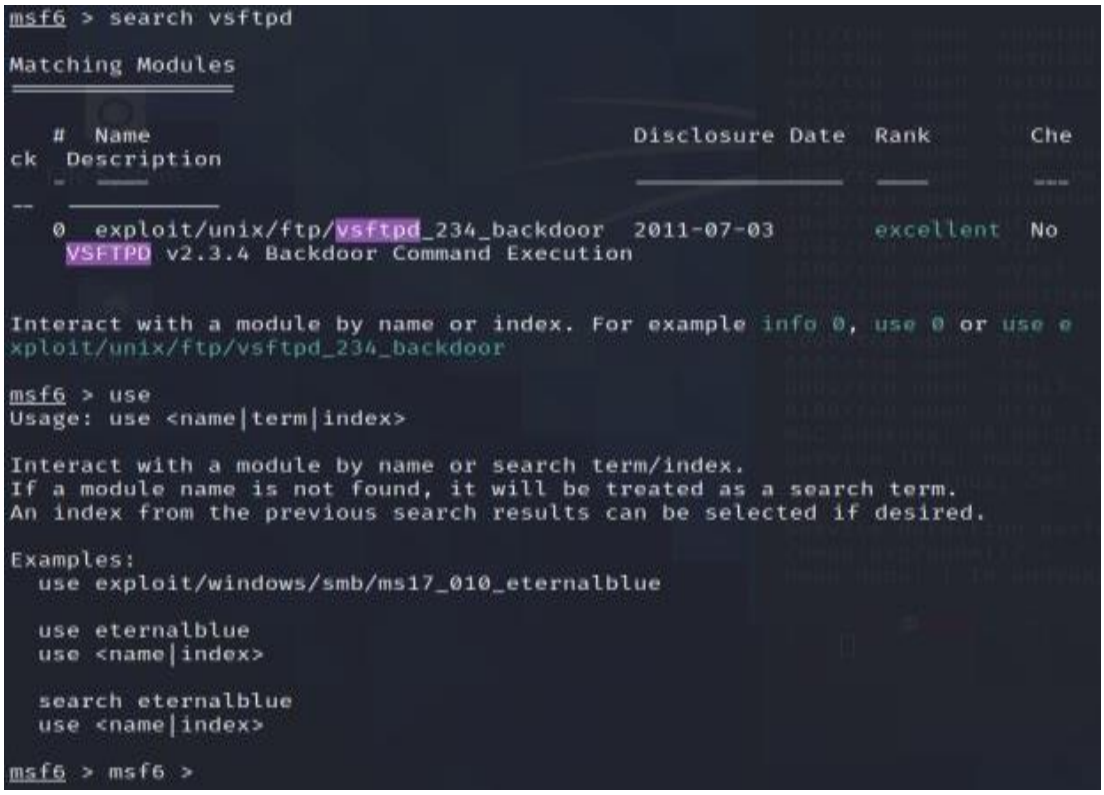
Günümüzde sürekli kullanıcıları zora sokan siber güvenlik açıkları ortaya çıkmaktadır. Bu tehditlerin çoğu, daha önce savunmasız olduğu bilinen veya zamanında uygulanmayan güncellemeler nedeniyle daha sonra savunmasız hale gelen kütüphane ve framework gibi bileşenlerin kullanılması gibi durumlar yüzünden bu zafiyetler doğmaktadır. Saldırganlar, kusurları bulmak için basitçe otomatik tarama araçlarını kullanabilir veya uygulamanın manuel analizini yapabilir ve belirli bir uygulamanın daha önce savunmasız bulunan bir bileşeni kullandığını bulabilirlerse, bu güvenlik açığından kolayca yararlanmaya çalışabilirler. Genellikle bağımlılıkları bulmak için bilinen html öğelerini kontrol etme, hataları tetikleme, zorunlu tarama vb. gibi parmak izi yöntemlerini kullanır.

2.9.1 Bilinen Güvenlik Açığı Zafiyeti İstismarı

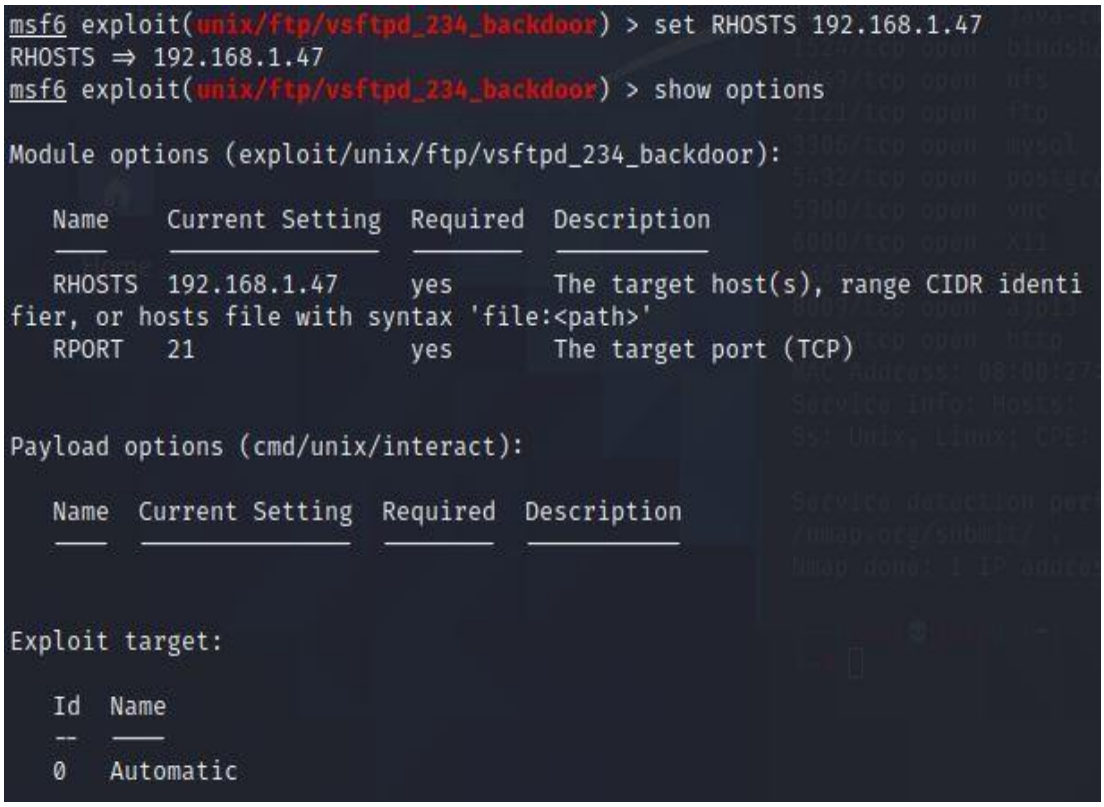
```
Nmap scan report for 192.168.1.47
Host is up (0.00010s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login       OpenBSD or Solaris rlogind
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
```

Şekil 2.9. 1: Zafiyetin Terminalde Bulunması (A)

Şekil 2.9.1 (A)'da bir "nmap" taraması uygulanmaktadır. "Nmap" ağ tarama ve zafiyet tespiti için kullanılan açık kaynaklı bir araçtır.



Zafiyetin Terminalde Bulunması (B)



Zafiyetin Terminalde Bulunması (C)

Şekil 2.9.1 (B) ve Şekil 2.9.1 (C)'de FTP servisinde çalışan "vsftpd" için bir arama yapıldığı gözükmektedir. Gerekli modülün seçimi yapılmaktadır. Exploit işleminden sonra backdoor oluşmaktadır.

2.9.2 Bilinen Güvenlik Açığı Zafiyeti Önlenmesi

Bileşenlerde bilinen güvenlik açıklarının kullanılması zafiyetini önlemek için birtakım stratejiler izlenebilir. İlk olarak, periyodik güvenlik incelemeleri yapılmalıdır. Bu incelemeler, kullanılan bileşenlerin ve kütüphanelerin güvenlik açıkları için taranması ve analiz edilmesini içerir. Güvenilir kaynaklardan yazılım temini de önemlidir. Bileşenlerin güvenilir kaynaklardan elde edilmesi veya indirilmesi, güvenlik açıklarının azaltılmasına yardımcı olabilir. Ayrıca, güvenlik güncellemelerinin düzenli olarak izlenmesi de kritiktir. Kullanılan bileşenlerin ve kütüphanelerin güncel olup olmadığı takip edilmeli ve güvenlik güncellemeleri hızlı bir şekilde uygulanmalıdır. Minimal ve gereksiz bileşen kullanımı da hayati öneme sahiptir. Yalnızca gereksinim duyulan bileşenlerin tercih edilmesi, saldırı yüzeyini azaltabilir. Otomatik güvenlik denetimlerinin kullanılması, bileşenlerin güvenlik açıklarının otomatik olarak tespit edilmesine ve değerlendirilmesine yardımcı olabilir. Bileşenlerin kaynağının doğrulanması da önemlidir. Güvenilirliği doğrulanmamış bileşenlerin kullanılmasından kaçınılmalı ve kaynak kodları titizlikle incelenerek güvenilirlikleri kontrol edilmelidir. Son olarak, kullanılan bileşenlerin güvenlik politikaları ile uyumlu olması sağlanmalıdır. Bu önlemler, bileşenlerde bilinen güvenlik açıklarının kullanılması zafiyetini azaltabilir ve uygulamanın güvenlik seviyesini artırabilir.

2.10 YETERSİZ LOGLAMA ve İZLEME

Yetersiz Loglama ve İzleme (Insufficient Logging & Monitoring), bir uygulama veya sistemlerin, olası güvenlik olaylarını belirleme ve bu olaylara müdahale etme konusunda yetersiz loglama ve izleme sistemleriyle karşılaşması anlamına gelir. Bu durumda, ilgili uygulama veya sistem, saldırı girişimlerini, hataları veya diğer güvenlik olaylarını belirlemek için gereken loglama ve izleme mekanizmalarına sahip değildir veya mevcut olanlar yeterli değildir.

Yetersiz loglama, uygulama veya sistemdeki olayların yeterince kaydedilmemesi veya gerekli bilgilerin loglara kaydedilmemesi durumunu ifade eder. Bu durumda, potansiyel saldırılar veya hataların kökenleri geriye dönük olarak araştırılamaz veya belirlenemez. İzleme eksikliği ise, uygulama veya sistemdeki etkinliklerin gerçek zamanlı olarak izlenmemesi veya izlenen etkinliklerin yeterince detaylı bir şekilde incelenmemesi durumunu ifade eder. Bu durumda, saldırılar veya diğer güvenlik olayları hızlı bir şekilde tespit edilemez ve bu olaylara müdahale edilemez.

2.10.1 Kaba Kuvvet Saldırısı

Kaba kuvvet saldırısı (Brute Force Attack), genellikle giriş bilgilerini kırma amacıyla yapılan bir siber saldırı türüdür, ki bu genellikle kullanıcı adı ve parola gibi kimlik bilgilerini içerir. Bu saldırıda, saldırgan, bir hizmete veya oturum açma sayfasına erişmek için otomatik bir program veya yazılım aracılığıyla kullanıcı adı ve parola gibi giriş bilgilerini sürekli olarak denemeye çalışır.

Bu saldırılar çoğunlukla bir saldırganın sisteme erişim elde etmeye çalıştığı zamanlarda gerçekleşir, bu süreçte geniş bir parola listesini veya rastgele parolaları otomatik olarak denemek yaygındır. Saldırganlar, oturum açma formlarını veya giriş sayfalarını hedef alarak, sürekli olarak farklı kullanıcı adı ve parola kombinasyonlarını deneyerek sisteme erişmeye çalışır. Bu tür saldırılar, zaman alıcı olabilir ancak kullanılan parola zayıfsa veya saldırgan yeterince kaynak sağlayabiliyorsa etkili olabilir. Güçlü parola politikaları, çoklu oturum denemesi engellemesi ve ikinci faktör

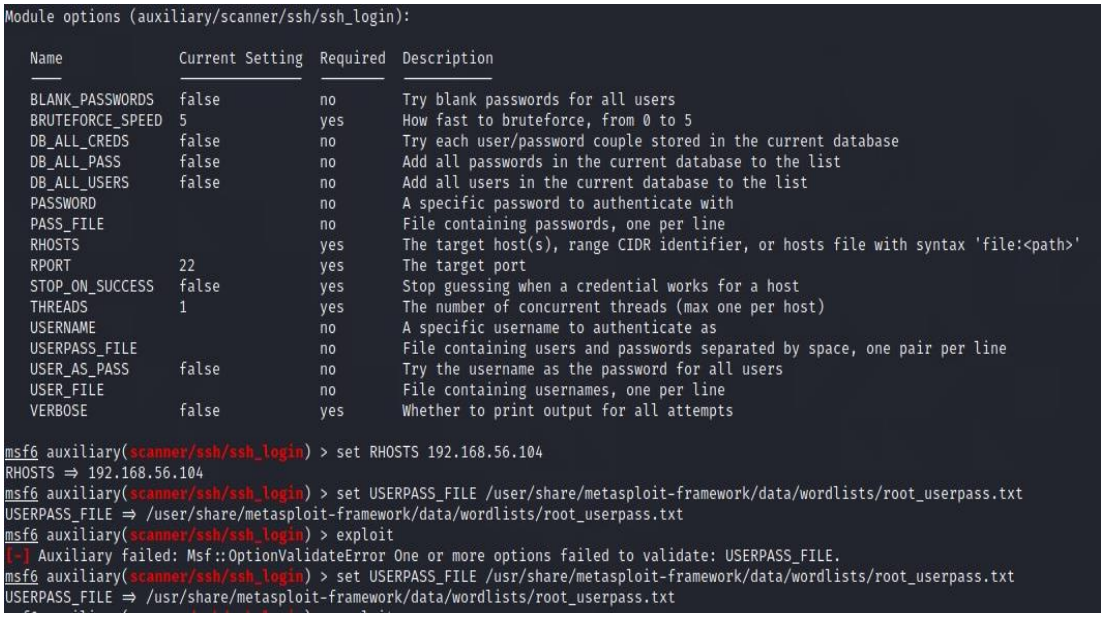
doğrulama gibi önlemler, kaba kuvvet saldırılarının etkinliğini azaltabilir veya önleyebilir.

2.10.2 Yetersiz Loglama ve İzleme İstismarı

Aşağıdaki uygulamada saldırgan makine olarak Kali Linux ve istismar edilen makine olarak metasploitable-2 kullanıldığı bilinmektedir.

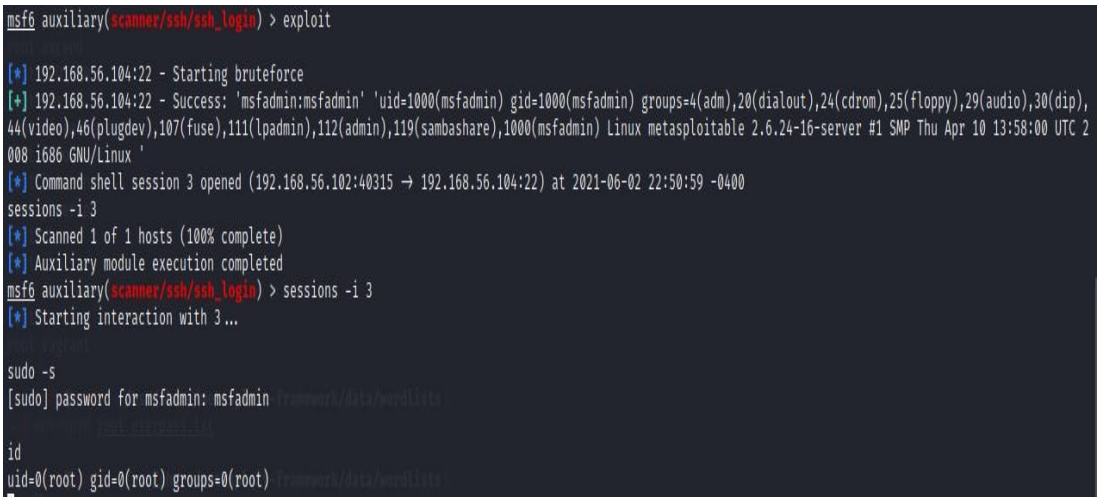
Şekil 2.10. 1: Zafiyeti Barındıran Makinenin Nmap Çıktısı

Şekil 2.10.1'de Zafiyetli makinede Nmap taraması gerçekleştirildiği gözükmektedir. Nmap çıktısında SSH servisinin çalışmakta olduğu ve portunun açık olduğu gözükmektedir. SSH için Brute Force atak saldırısını denendiği varsayılmaktadır ve Metasploit Framework kullanıldığı bilinmektedir.



Şekil 2.10. 2: Zafiyeti Barındıran Makinenin Bilgilerin Set Edilmesi

Şekil 2.10.2'de Brute Force atağı yapılırken auxiliary/scanner/ssh/ssh_login modülü kullanıldığı gözükmektedir. Modülün başarılı olabilmesi için zafiyetli makinenin ip adresi ve Brute Force atağı yapılırken Metasploit Framework ile birlikte gelen root_userpass.txt wordlist dizininin set edildiği bilinmektedir.



Şekil 2.10. 3: Zafiyetli Makinenin Sömürülmesi

Şekil 2.10.3'de Exploit edilerek brute force saldırısının başarılı olduğu gözükmektedir. Kullanıcı adı ve parolayı msfadmin olarak bulunduğu gözükmektedir. Log kaydının tutulmaması, önlem alınmaması ve izlenmemesi bu zafiyeti sebep olmuştur.

2.10.3 Yetersiz Loglama ve İzlemenin Önlenmesi

Yetersiz Günlükleme ve İzleme (Insufficient Logging & Monitoring) önlenmesi için çeşitli tedbirler alınabilir. İlk olarak, güvenlik olaylarını tespit etmek için kapsamlı bir günlükleme stratejisi benimsenmelidir. Uygulama veya sistemde meydana gelen önemli olaylar, detaylı bir şekilde kaydedilmelidir. Bu, olayların geçmişe yönelik olarak incelenmesini ve muhtemel güvenlik ihlallerinin tanımlanmasını mümkün kılar. Ayrıca, gerçek zamanlı izleme ve uyarı mekanizmaları oluşturulmalıdır. Sistemdeki anormallikler veya potansiyel güvenlik tehditleri, izleme sistemleri tarafından sürekli olarak takip edilmeli ve uygun kişilere veya sistem yöneticilerine bildirilmelidir. Bu sayede, olası saldırılar hızlı bir şekilde tespit edilip cevaplanabilir. Güvenlik olaylarını izlemek için otomatik araçlar veya yazılımlar kullanılabilir. Bu araçlar, belirlenen güvenlik politikalarına uygun olarak günlükleri inceleyebilir ve anormal aktiviteleri tanımlayabilir. Bir diğer önemli adım ise, günlük verilerinin güvenli bir şekilde saklanması ve korunmasıdır. Günlük verileri, izinsiz erişime karşı korunmalı ve gerektiğinde doğrulanabilir olmalıdır. Bu, günlük verilerinin bütünlüğünü ve güvenilirliğini sağlar. Günlükleme ve izleme süreçleri düzenli olarak gözden geçirilmeli ve geliştirilmelidir. Teknolojik ilerlemeler ve güvenlik tehditlerindeki değişiklikler dikkate alınarak, günlükleme ve izleme stratejileri periyodik olarak güncellenmelidir.

3 SONUÇ

Bu araştırmada, kullanıcının gelen verileri güvenli bir şekilde kontrol etmemesi nedeniyle zafiyetlerin ortaya çıktığı belirlenmiştir. Injection, Broken Authentication, Sensitive Data Exposure, XML External Entity, Broken Access Control, Security Misconfiguration, Cross Site Scripting, Insecure Deserialization, Using Components with Known Vulnerabilities ve Insufficient Logging and Monitoring gibi çeşitli zafiyetlerin tespit edilmiş olması, bu durumun ciddiyetini artırmaktadır. Verilerin kötü niyetli kişilerin eline geçmesiyle ciddi sonuçlara yol açabileceği vurgulanmıştır. Özellikle, Sensitive Data Exposure gibi zafiyetlerin, hassas verilerin yetkisiz kişiler tarafından ele geçirilmesine neden olabileceği belirlenmiştir. Verilerin kötü niyetli kişilere ulaşmasını engellemek için alınacak tedbirler üzerinde durulmuş ve bu tedbirlerin uygulanmıştır. Bu tedbirlerde Broken Authentication ve Broken Access Control gibi zafiyetlerin önlenmesine odaklanmıştır. Araştırma sonuçları, bu tedbirlerin etkili olduğunu ve saldırganların yöntem ve yaklaşımlarının sisteme olan etkisinin azaldığını göstermektedir.

Tablo 1: OWASP Top 10 Zafiyet Puanlaması

ZAFİYET	PUAN
Injection	8.0
Broken Authentication	7.0
Sensitive Data Exposure	7.0
XML External Entity	7.0
Broken Access Control	6.0
Security Misconfiguration	6.0
Cross Site Scripting	6.0
Insecure Deserialization	5.0
Using Components with Known Vulnerabilities	4.7
Insufficient Logging and Monitoring	4.0

Tablo 1'de bu zafiyetlerin tespit edilmesi ve araştırılması sürecinde önem sırasında göre OWASP tarafından yapılan puanlama sistemi gözükmektedir. Araştırma sürecinde, ilgili zafiyetlerin neden olduğu teknolojiler üzerinde derinlemesine bir inceleme yapılmıştır. Bu süreçte, veri tabanı sistemleri, SQL sorguları, XML, Samba, JavaScript, PHP, serileştirme işlemleri, Burp Suite gibi web uygulama güvenliğinde önemli araçlar ve Linux komutları gibi birçok teknolojiye odaklanılmıştır. Özellikle, XML External Entity ve Insecure Deserialization gibi zafiyetlerin teknik detayları incelenmiştir. Web uygulama güvenliğinin sağlanması için bu çeşitli sistem ve teknolojilerin öğrenilmesinin gerekliliği vurgulanmıştır. Araştırma, bu süreçte kriptoloji gibi konuların da önemini ortaya koymuştur.

Sonuç olarak, yapılan araştırma zafiyetlerin tespiti ve önlenmesi için alınacak önlemlerle ilgili önemli bulgular sunmuştur. Bu bulgular, güvenlik açıklarının kapatılması ve web uygulamalarının daha güvenli hale getirilmesi için yol gösterici olacaktır. Injection, Broken Authentication, Sensitive Data Exposure, XML External Entity, Broken Access Control, Security Misconfiguration, Cross Site Scripting, Insecure Deserialization, Using Components With Known Vulnerabilities ve Insufficient Logging and Monitoring gibi zafiyetlerin önlenmesi için alınacak önlemlerin önemi vurgulanmalıdır. Bu önlemler, siber güvenlikte daha sağlam bir zemin oluşturarak saldırganların etkisini azaltabilir.

4 KAYNAKÇA

Altundaş, A. (2022). *Kali Linux* (19.Baskı). İnkılap Kitapevi Yayınevi.

Aslanbakan, E. (2019). *Bilgi Güvenliği ve Hacking*. Pusula Yayınevi.

Bülbül, İ., Bingöl, P. E. (2017). *Etik Hackerlığa Giriş*. Hayy Kitap Yayınevi.

Bingöl, P. E., Bülbül, İ., Eren, V. (2018). *Etik Hackerlığa Giriş 2*. Hayy Kitap Yayınevi.

Çıtak, Ö. (2020). *Ethical Hacking Offensive & Defensive* (19.Baskı). Abaküs Yayınevi.

Eren, V., Komut, A., Çakır. Y. C. (2020). *Uygulamalı Siber Güvenlik ve Sızma Testi Eğitimleri*. Hayy Kitap Yayınevi.

Ertuğrul, İlker. (2020). *Ofansif ve Defansif Siber Güvenlik*. Dikeyeksen Yayınevi.

Friedman, A., Singer, P. W. (2015). *Siber Güvenlik ve Siber Savaş*. Buzdağı Yayınevi.

Gürel, A. (2023). *Siber Güvenlik*. Dikeyeksen Yayınevi.

Özkaya, E. (2020). *Siber Güvenlik: Saldırı ve Savunma Stratejileri*. Buzdağı Yayınevi.

Pekel, A. (2023). *Siber Güvenlik Yönetimi*. ODTÜ Yayıncılık.

Samancıoğlu, A. (2020). *Python Sıfırdan Uzmanlığa Programlama*. Unikod Yayınevi.

Trim, P. (2023). *Stratejik Siber Güvenlik Yöntemi*. Nobel Akademik Yayıncılık.