SDU International Journal of Technological Science
Vol. 8, No 2, August 2016
pp. 1-14

ComputationalTechnologies

WEB SERVİSLERİNİN YAZILIM GÜVENLİK TESTLERİ İÇİN ÖNERİLEN HİBRİT YAKLAŞIM

Güncel SARIMAN*, Ecir Uğur KÜÇÜKSİLLE

Geliş Tarihi/ Received: 19.03.2016, Kabul tarihi/Accepted: 01.09.2016

Özet

Teknolojik gelişmelerin toplumun her kesimine hitap etmesi, çeşitli ihtiyaçlarıda beraberinde getirmektedir. Günlük hayatı zorlaştıran birçok hizmet artık web ve mobil uygulamalar ile yapılabilirken yazılımların son yıllarda çeşitlenmesiyle farklı sistemler arasında veri transferleri de ihtiyaç haline gelmiştir. Farklı sistemlerdeki veritabanlarının birbirleriyle platform bağımsız bir şekilde haberleşebilmeleri için web servisleri kullanılmaktadır. Web servislerindeki güvenlik ve gizlilik web uygulamalarında olduğu gibi oldukça önemlidir. Kullanıcılar, hayati önem taşıyan işlemleri online sistemler üzerinde, verilen hizmetlere güvenerek işlemlerini gerçekleştirmektedir. Geliştirilen web servis uygulamalarında, güvenlik önlemlerinin yazılımın ilk süreçlerinden itibaren dikkate alınması, güvenlik risklerini azaltmaktadır. Web servis uygulamalarının tek bir test modeline göre değerlendirilmesiyle, muhtemel açıklıklar yeterince tespit edilememektedir. Bu çalışmada, web servislerinin güvenliğini test etmek için geliştirilen hibrit model açıklanmaktadır. Hibrit modelde güvenlik testleri sırasında kullanılan statik ve dinamik analizin yanında gözden geçirme yöntemi dahil edilerek, otomatize araçların bulamadığı açıklıklartespit edilmektedir. Bu sayede web servislerinin geliştirilmesi sırasında dikkat edilmesi gereken bölümler tespit edilebilmektedir. Çalışma kapsamında web servislerinde olması gereken kimlik denetimi ve uygulama dillerine bağlı olarak oluşabilecek güvenlik açıkları örnek kodlarla birlikte anlatılmaktadır. Son olarak geliştirilen model ile test web servisleri, açık kaynak yazılımlar ve gözden geçirme yöntemiyle test edilerek, önerilen modelin geçerliliği test edilmektedir.

Anahtar Kelimeler: Web Servisleri, Yazılım Güvenliği, Güvenlik Testleri, Statik Kod Analizi

A HYBRID APPROACH FOR SOFTWARE SECURITY TESTS OF WEB SERVICES

Abstract

As nowadays technology appeals to all parts of the society, different needs have appeared. While web and mobile applications make daily life easier; data transfer among different systems has been a necessity with variety of web, mobile and desktop software. Web services are used for data transfer among databases of different systems and for platform independent communication. Security and privacy in web services, which connect different systems, are very important as they are in web applications. Users perform vital transactions online trusting the services. In web service applications, security risks are decreased with security precautions during the first phases of developing a software. If a software is evaluated by only one test model during security testing, potential security vulnerabilities cannot be detected adequately. In this research, a hybrid model was proposed for testing web service security. In this hybrid model, parts which need attention while developing a web service were detected with static, dynamic and code review methods during security testing. User authentication, which is a requirement in web services, and security vulnerabilities which may occur depending on programming languages were explained with sample codes. In the last section of this research, the benchmark web services were tested with the hybrid model by using open source software and the validity of the model was put forward with the results.

Keywords: Web Services, Software Security, Security Test, Static Code Analysis

*Muğla Sıtkı Koçman Üniversitesi, Bilgi İşlem Daire Başkanlığı, 48000 Muğla
E-posta: guncelsariman@mu.edu.tr

1. Giriş

Hızla artan teknolojiyle birlikte internet ve elektronik cihazların kullanımı tüm Dünyada yaygınlaşmaktadır. Web uygulamalarının günlük hayatı kolaylaştırmasıyla birlikte cep telefonlarıyla kullanılabilen mobil yazılımlar günümüzün vazgeçilmez bir parçası olmuştur. Son zamanlarda bankacılık işlemleri, kamusal hizmetler, online alışveriş gibi kullanıcı güvenliği ve gizliliğinin önemli olduğu işlemler internet üzerinde kullanıcı kontrolünden bağımsız bir şekilde çalışmaktadır. Bu noktada kullanıcılar hizmetlere güvenerek işlemlerini yürütmektedirler. Yazılım geliştirme sürecinin en başından beri dikkate alınması gereken güvenlik yaklaşımı internetin gelişmesi, web ve mobil uygulamaların yaygınlaşması ile kendini daha da belirgin hale getirmiştir. Güvenli yazılım, ulaşabildiği verinin bütünlüğünü, mahremiyetini korurken, bilgiye erişimin devamlılığını da sağlayabilmelidir. Uygulama güvenliğinde temel prensip, güvenlik önlemlerinin veya denetimlerin uygulama geliştirme safhasının ilk adımlarında dikkate alınmasıdır. Geliştirmenin ilk aşamalarında kimlik doğrulama, çalışma kapsamında ortaya çıkabilecek güvenlik açıklarının gereksinimleri ve kod denetimlerinin yapılması önem arz etmektedir.

Farklı uygulama platformlarında geliştirilen yazılımlar çeşitli amaçlarla kullanılabilmektedir. Masaüstü uygulaması olarak çalışan yazılımların yanında son zamanlarda daha sık kullanılan ve masaüstü uygulamaların yerini alan web siteleri, tablet ve cep telefonlarında kullanılabilen mobil yazılımların önemi artmaktadır. Farklı sistemlerdeki veritabanlarının birbirleriyle veri alışverişi yapabilmeleri ve platform bağımsız bir şekilde haberleşebilmeleri için web servisleri kullanılmaktadır. Web servisleri internet dünyasında XML (Extensible Markup Language) ve JSON (JavaScript Object Notation) standartı ile veri aktaran uygulamalardır. Web servisleri, Http protokolü ile XML gönderip alarak iki uzak cihaz arasındaki iletişimi sağlayan bir haberleşme yöntemidir. SOAP (Simple Object Access Protocol) ile web servisleri veya buna benzer dağıtık sistemler arasındaki iletişimin belirli bir standarda göre yapılması amaçlanmaktadır. WSDL (Web Services Description Language) dokümanları ise kullanılacak olan web servisleriyle ilgili metotların isimleri, alacağı parametre adları ve veri tipleri hakkında bilgiler vermektedir.
Dış dünyaya kontrolsüz bir şekilde açılan web servisleri beraberinde güvenlik problemlerini de getirmektedir. Web servislerinin, uygulamaların API (Application Programming Interface)' lerine ve hedef uygulamalara erişim sağlaması, güvenlik açıklıklarına sebep olabilmektedir. Web servislerinin dağıtık ve uçtan-uca yapısı, tehdit ve güvenlik açıklıklarının bir uygulamadan başka uygulamalara atlamalarına neden olabilmektedir. Servisler üzerinde oluşabilecek güvenlik zafiyetleri web uygulamalarında görülen açıklıklardan çok farklı değildir. XSS (Cross site scripting), SQLI (Sql Injection), XPATH (Xml Path) enjeksiyonu gibi girdi denetimi eksikliğinden meydana gelen saldırı türleri, kimlik doğrulama ve yetersiz yetkilendirmeler, web servislerini güvensiz hale getirmektedir. Ayrıştırma (Parsing) saldırıları da sistemlerin yavaşlamasına ve durmasına neden olabilmektedir. Belirlenen bir eleman türünden çok sayıda iç içe aynı elemanın kopyaları gönderilerek XML ayıştırıcısının bellek ve CPU sınırları zorlanarak saldırılar gerçekleştirilebilmektedir.

Farklı veritabanlarına doğrudan erişim yerine web servisleri yazılarak birden çok veritabanının haberleştirilmesi ve birbirleriyle bütünleşik çalışması, web servislerini günümüzde vazgeçilmez kılmaktadır. Bu sebeple güvenlik seviyesinin en üst seviyede olması,

uygulama seviyesinde meydana gelebilecek hataların ve zafiyetlerin tespit edilmesi adına çok önemlidir.

Yue ve Tao (Yue ve Tao, 2012) çalışmalarında, farklı sistemler arasında kullanılan web servislerindeki güvenliğe dikkat çekmişlerdir. Çalışmada Microsoft .Net ve Axis platformları hakkında bilgi verildikten sonra bu 2 iki platformun birbirleriyle haberleşebildiği bir güvenlik modeli kurulmuştur. Antunes ve Vieira (Antunes ve Vieira, 2009) web servislerinin, sistemler arasında güvenliği en iyi seviyede tutması gerektiğini belirtmişlerdir. SQLI saldırıları web servislerinde veritabanlarına doğrudan müdahalede bulunarak zafiyet yaratmaktadır. Çalışma kapsamında SQLI saldırılarını tespit edebilmek için açık kaynak yazılımlar kullanılarak, statik kod analizi ve sızma testi sonuçları karşılaştırılmıştır. Sonuç olarak statik kod analizi çalışmalarının sızma testlerine göre daha fazla zafiyet tespit edebildikleri ortaya çıkarılmıştır. Masood ve Java çalışmasında (Masood ve Java, 2015) web servislerindeki güvenlik açıklarını tespit etmek için güvenlik araçlarını ve standartları, OWASP (Open Web Application Security Project) web servis rehberini kullanmıştır. Owasp' ın kabul ettiği 10 güvenlik açığını statik kod analiz teknikleriyle tespit etmeye çalışmışlardır.

Bu çalışmada web servislerinde meydana gelebilecek güvenlik açıklarından ve web servislerinin güvenliği konusunda kullanıcılara sunulan hibrit yaklaşımdan bahsedilmektedir. Çalışmanın materyal ve metot bölümünde güvenli yazılım geliştirme sürecinden, web servislerinden ve servis güvenliğini tehdit eden faktörlerden söz edilirken, web servisleri için geliştirilen hibrit model de bu bölümde açıklanmaktadır. Araştırma bulguları bölümünde ise web servislerinin geliştirme aşamasında yapılması gerekenler, kullanıcı adı ve şifre güvenliği, girdi denetimlerinin nasıl yapılması gerektiği, hafıza taşması gibi saldırılar, servis arayüzü erişim imkânı, DOS (Denial of Service) saldırılarına alınabilecek önlemler, sunucu bazında SSL (Secure Socket Layer) koruması açıklanmaktadır. Güvenlik denetimleriyle ilgili örnekler bu bölümde verilmiştir. Sonuç bölümünde ise geliştirilen model ile test web servisleri, açık kaynak yazılımlarda ve önerilen modelde test edilerek, modelin geçerliliği test edilmektedir. Ayrıca, modelin nerelerde kullanılabileceği ve nasıl uygulanabileceği de makale kapsamında açıklanmaktadır.

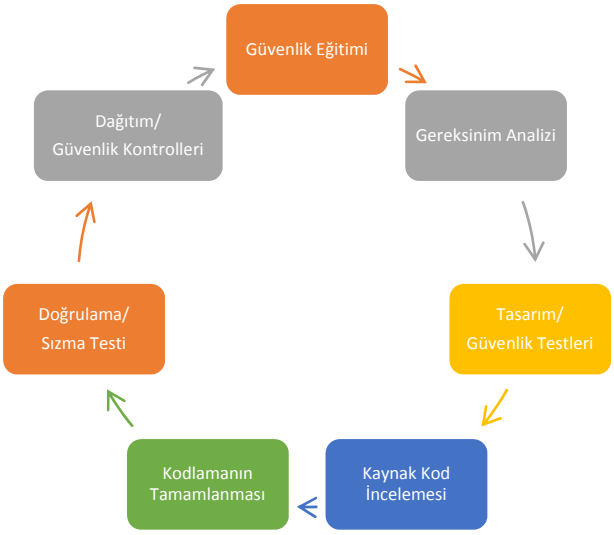
2. Materyal ve Metot

2.1. Güvenli Yazılım Geliştirme

Yazılımların, günümüz kritik altyapılarında, finansal sistemlerde, tıbbi bilgi sistemlerinde kullanımı gün geçtikçe artmaktadır. Bilgi sistemlerindeki güvenlik zafiyetleri de geliştirilen uygulamalar üzerinde yaygınlaşmaktadır. 2005 yılından itibaren kayda geçirilen bilişim güvenliği ihlalleri olaylarına göre, Privacy Rights Clearinghouse adlı dernek tarafından her yıl büyük bir bilgi hırsızlığı olayı yaşandığı tespit edilmiştir. Dernek tarafından 226 milyon bilişim güvenliği ihlali raporlanmıştır (Privacy Rights ClearingHouse, 2005). Yazılım ürünlerinde güvenliği arttırmayı düşünen kuruluşlar, yazılım güvenliğini keşfetmeye ihtiyaç duymaktadırlar. Bununla birlikte güvenli yazılım kolay kolay geliştirilememekte ve yatırımcılar yazılım geliştirme sürecinde yeterli önemin gösterilmesinden dolayı, yazılımataklarına veya mevcut güvenlik açıklarına karşı direnememektedirler. Yazılım projelerinin mimarisi ilk günkü gibi olmamakla birlikte işlevsellik gün geçtikçe artmakta ve beraberinde karmaşık ve detaylı projeler oluşmaktadır. Yazılım güvenliği, YGYD (Yazılım

Geliştirme Yaşam Döngüsünün) başlarında dikkate alınmamakta ve sadece projelerin son aşamalarında yazılıma entegre edilmeye çalışmaktadır. Bu sebeple, yazılım geliştirmenin çeşitli aşamalarında güvenlik açıklarının riskleri artmaktadır (McGraw, 2006).

Yazılımı oluşturan birçok bileşen ve dinamik yapı olduğu için, tam güvenli yazılım kavramından söz edilememektedir. Yazılımlardaki güvenlik zafiyetlerinin temel sebepleri; kodlama ve mimari hatalardır. Yazılım Güvenliği, yazılımların tersine mühendislik yöntemleri ve araçları ile algoritmaların ortaya çıkarılması veya değiştirilmesini engellemeyi amaçlayan yöntemler bütünü olarak tanımlanabilmektedir. Diğer bir tabirle, güvenli yazılım, güvenliği göz ardı etmeden tasarlanmış, güvenlik kontrolleri ile geliştirilmiş ve güvenli bir durumda kullanıma sunulmuş yazılım olarak tanımlanabilmektedir(Karayumak, 2013). Yazılım geliştirmenin ilk süreçlerinde farkına varılan yazılım açıklıklarının düzeltilmesinin daha ileri süreçlerde farkına varılan açıklıklara göre daha az maliyetli olacağı yazılım endüstrisince yaygın olarak kabul edilen bir ilkedir (Michael, 2005). Yazılım geliştirme yaşam döngüsünün temel ayağı olan 4 genel yaklaşım bulunmaktadır. YGYD metodolojisi şelale modeli, artırımlı model, spiral model, prototip modeli olarak sınıflandırılmaktadır. Tüm modellerde, güvenlik prensibini YGYD' ne dâhil etmek için yaşam döngüsünün her aşamasında güvenliği ele almak gerekmektedir. Şekil1' de güvenli yazılım geliştirme yaşam döngüsü verilmiştir.



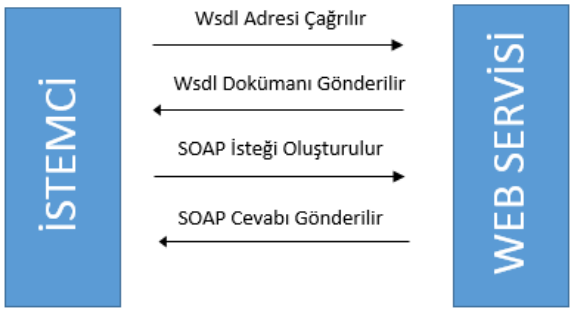
Şekil 1. Güvenli yazılım geliştirme yaşam döngüsü

GYGD' nin ilk aşaması olan güvenlik eğitiminde geliştiricilere güvenli tasarım, tehdit modelleme, kod analizi gibi temel kavramlar verilmektedir. Gereksinim analizi bölümünde ise amaç yazılımın ihtiyaçlarını belirlemektir. İhtiyaçlar belirlendikten sonra proje takımına güvenlik anketleri yapılarak güvenlik ile ilgili ön hazırlıklar tamamlanır. Tasarım aşamasında ilk olarak etkileşim noktaları tanımlanır. Bu noktalar yazılımın iş akışını anlamada yardımcı olur ve ortaya çıkabilecek güvenlik sorunları hakkında derinlemesine bir bilgi verir. Güvenlik tasarım uygulamaları bir yazılımın nasıl saldırıya maruz kalabileceğine, neyin saldırabileceğine, hangi alanlarda saldırı eğilimi olabileceğine, ne tür atakların uygulanabileceğine dair tam bir bilgi sağlar. Güvenli kod, tasarım aşamasında tanımlanan tehditler ve genel güvenlik kuralları dikkate alınarak yazılır. Tüm bu faktörler, güvenli kod yazmak için çok önemlidir ve herhangi birinin eksikliği güvenlik açıklarına sebep

olabilmektedir. Kodlama sırasındaki kaynak kod incelemeleri, güvenlik testleri süreçte projeyi sağlam bir temele oturtmaktadır. Doğrulama aşaması ürünün gizlilik, bütünlük ve erişilebilirlik yönüyle, tehdit modelleri göz önünde bulundurularak test edilmesi ile başlar. Bulunan zafiyetlerden ileride açıklık oluşturma ihtimali olanlar bu aşamada giderilir (Alparslan,2013). Yazılımın geliştirilmesi bittiğinde, son kullanıcıya dağıtıma başlanmadan hemen önce geliştirilen yazılımın güvenli olup olmadığının kontrol edilmesi ve her şeyinin güvenli olduğundan emin olunması gerekmektedir (Burak, 2015). Denetim sırasında güvenlik açığı tespit edilirse, yazılımda değişikliğe gidebilir. Denetim sonunda ise denetim raporu oluşturulur. Denetim raporuna göre, proje yönetimi yazılımın yayınlanma durumuna karar verir. Süreçlerin tümü tamamlandıktan sonra güvenli bir yazılım projesi ortaya çıkarılmış olunur.

2.2 Web Servis Güvenliği

Web servisleri, XML ve JSON standartlarında veri alışverişi yapan uygulamalardır. İnternet ortamında, platformlar arası haberleşmeyi sağlayan servisler, sahip oldukları standartlar sayesinde farklı programlama dillerinde çalışabilmektedirler. Web servisleri klasik web servisleri ve REST (Representational State Transfer) mimarisini olarak bilinen Web API olarak iki ana kategoride incelenmektedir. Klasik web servisleri XML standartını kullanan ve SOAP mimarisi ile haberleşen servislerdir. Web API ise istemci sunucu mimarisi içerisinde verinin basit bir şekilde transfer edilmesidir. REST, en temel anlamda, istemci ve sunucu arasında veri alış verişinin basit bir yoludur. REST mimarisinde HTTP metotlarından yararlanılmaktadır (Güvercin, 2015). Farklı uygulamalar arasındaki haberleşmeyi sağlayan web servis kullanımının kontrolsüz bir şekilde büyümesi beraberinde güvenlik problemlerini de getirmektedir. Web uygulamalarındaki güvenlik zafiyetleri aynı şekilde web servislerinde de görülebilmektedir. XML, verilerin saklanması ve taşınması işlemlerinde kullanılmaktadır. SOAP, web servisleri arasındaki haberleşmenin belirli standartlarda yapılmasını amaçlamıştır. WSDL ise kullanılacak olan web servisleriyle ilgili teknik detayları barındırmaktadır. Dokümantasyon sayesinde, kullanılan metotlar hakkında parametre ve fonksiyon detayları hakkında bilgilere erişilebilmektedir. (Demir, 2013). Şekil 2 de klasik web servislerinin çalışma yapısı gösterilmiştir.



Şekil 2. Klasik web servisleri çalışma yapısı

Web servis dağıtık uygulamalarda kullanılmak üzere oluşturulmuş, RPC (Remote Procedure Call) modelini kullanan istemci sunucu mantığıyla çalışan bir protokol yapısıdır(Güven, 2015).

Web servisleri konusunda kabul görmüş; OASIS (The Organization for the Advancement of Structured Information Standards), W3C (The World Wide Web Consortium), WS-I (The Web Services Interopability Organization) ve IETF (The Internet Engineering Task Force) gibi organizasyonlar ve IBM, Microsoft gibi firmalar sayesinde belirlenen standartlar ile web servis kullanımının güvenli bir şekilde gerçekleştirilmesi planlanmıştır (The Java Web Services Tutorial, 2015). Harici uygulamalar arası iletişim sağlayan servisler, dâhil oldukları uygulamalar üzerinde güvenlik açıkları oluşturabilmektedirler. Servislerin, dağıtık ve uçtan-uca yapısı nedeniyle tehdit ve güvenlik açıklıkları bir uygulamadan başka uygulamalara iletilebilmektedir. Sistem seviyesinde gerçekleştirilen bazı güvenlik önlemleri yeterli görülse de uygulama seviyesinde birçok güvenlik zafiyeti oluşabilmektedir. Çünkü servislere yapılan saldırılar son zamanlarda sistem seviyesinden daha çok uygulama seviyesinde meydana gelmektedir.

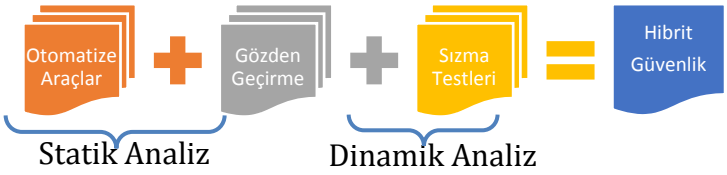
Web servislerinde meydana gelen güvenlik zafiyetleri web uygulamalarında meydana gelen zafiyetlerle birbirlerine benzemektedir. Servisler için WSDL taraması ile saldırgan, web servisi oluşturmak için kullanılan teknolojiyi tespit etmek ve ilgili güvenlik açıklarını bulmak için WSDL arayüzü taraması yapabilmektedir. Bu tür saldırılar, parametre kurcalama, zararlı içerik enjeksiyonu, komut enjeksiyonu gibi saldırılar gerçekleştirmek için yapılmaktadır. WSDL taraması veritabanından bilgi çekme, sistemleri hizmet dışı bırakma gibi zararlara yol açabilmektedir. DOS saldırıları ile web servisleri hizmet dışı bırakılabilmektedir. XML içerisindeki meta verilerin max boyutlarını aşarak ayrıştırıcı tarafından işlenememesiyle Jumbo tag zafiyetleri servislerde görülebilmektedir. Saldırganların XML içerisindeki cdata alanlarını kullanarak sistem komutları gönderebildikleri saldırı tipine coercive parsing saldırısı olarak bilinmektedir. XML dökümanları içerisine büyük boyutlu veriler girilmesiyle servisin çalışmasını engelleme saldırıları da web servisleri için önemli bir tehdittir (Alparslan,2013). Enjeksiyon saldırıları, web uygulamalarında olduğu gibi web servislerinde de sıkça görülebilmektedir. XML içeriğine sorgu göndermek suretiyle veritabanından veri elde etmeye yarayan saldırı tipi SQLI olarak bilinmektedir. XML enjeksiyonu saldırısı ise web servislerinde SQL cümlecikleri yerine XML dökümanı içerisindeki verileri sorgulayarak meydana gelmektedir. Bu saldırı türü XPATH (XML Path) enjeksiyonu olarak da bilinmektedir. Web servislerinin güvenliğini etkileyen diğer faktör ise yetkilendirme problemleridir. Yetkilendirmelerin önemsenmemesi web servislerinin yetkisiz kullanıcılar tarafından ele geçirilmesine ve önemli zafiyetlerin oluşmasına neden olmaktadır.

2.3 Hibrit Yaklaşım

Web uygulamalarının giderek yaygınlaşmasıyla birlikte farklı sistemlerin birbirleriyle veri alışverişi yapması daha kolay hale gelmiştir. Bu noktada, web servisleri aradaki bağlantıyı sağlayarak sistemleri birbirlerine bağlamaktadırlar. Farklı sistemlerin haberleşmesi yanında, iletişimin güvenli bir ortamda gerçekleşmesi sistemlerin risk altında kalmasını engellemektedir. Programlama dili seçiminin, proje kapsamının ihtiyaçlarını tam olarak gideremememesi, yetersiz güvenlik farkındalığı, geliştirme döngüsünün izlenmemesi ya da bu döngünün herhangi bir yerine güvenli yazılım geliştirme süreçlerinin dâhil edilmemesi ve güvenlik testlerinin yapılmaması veya yetersiz kalması nedeniyle web servislerinde birçok güvenlik sorunu ortaya çıkmaktadır (Yılmaz, 2010). Web uygulamalarındaki güvenlik önlemleri web servisleri içinde uygulanabilmektedir. Uygulamalardaki zafiyetleri tespit edebilmek için projeler güvenlik testlerinden geçirilmektedir. Güvenlik testleri dinamik ve

statik olmak üzere iki aşamada gerçekleştirilmektedir. Dinamik testler, çalışan uygulamalar üzerinde gerçekleştirilen zafiyet bulma testleridir. Aynı zamanda bu testler, sızma testleri olarak da bilinmektedirler. Sızma testleri, kara kutu ve beyaz kutu testleri olmak üzere ikiye ayrılmaktadır. Kara kutu testlerinde testi gerçekleştiren kişiler sistem hakkında hiçbir bilgiye sahip olmadan dışardan sisteme saldırıda bulunarak güvensiz noktaları tespit etmeye çalışırken, beyaz kutuda, test kullanıcıları sistem hakkında ip, kaynak kod gibi sistem hakkında ön bilgileri elde ederek sisteme saldırıda bulunurlar. Sızma testleri, sistemlerin güvensiz olma durumlarını tespit etmek için kullanılmaktadırlar. Yazılımların çalıştırılmasına gerek kalmadan uygulanabilen statik teknikler arasında kod gözden geçirmeleri, otomatikleştirilmiş statik kod analizi ile birim testleri yer almaktadır. Statik testler, uygulamalar henüz kodlama aşamasındayken, piyasaya sürülmeden önce gerçekleştirilen testler olduğu için güvenlik zafiyetlerini projelerin erken aşamalarında tespit edebilmektedirler. Statik analiz gözden geçirmeyle veya otomatik analiz araçlarıyla yapılabilmektedir. Statik analiz araçları daha çok atama ve denetim gibi programlama hatalarını tespit ederken, gözden geçirmelerle fonksiyonel hatalar ve otomatik araçlarla tespit edilmesi zor olan hatalar ortaya çıkarabilmektedir (Zheng vd., 2006).Dinamik ve statik testlerin kendilerine ait avantajları olsa da, tek başlarına yeterli olamamaktadırlar. Dinamik testler sırasında uygulamadaki hataları kara kutu testi yöntemiyle bulmaya çalışan tarayıcılar, sistemde yavaşlama, ortaya çıkabilecek hataların herkese yansıtılması gibi faktörlerden dolayı her durumda kullanılamamakta ve kod tarafındaki hataları tespit edememektedir. Statik kod analizinde ise otomatize araçlar her ne kadar genel kabul görmüş hataları tespit edebilseler de, yanlış negatif veya yanlış pozitif hataları ve tasarımsal hataları bulmakta zorlanmaktadırlar. Gözden geçirme yöntemi bu aşamada devreye girerek uygulamadaki fonksiyonel hatalar tespit edilebilmektedir.

Web servislerinin güvenlik testlerinde, yaşanan sıkıntılar kapsamında, hibrit bir yaklaşımın, servis güvenliğini en üst düzeyde tutacağı düşünülmüştür. Hibrit yaklaşım modelinde dinamik testlerle birlikte statik analiz de uygulanarak web servislerindeki zafiyet hem ürün piyasaya sürüldükten sonra kara kutu test tekniği ile test edilebilirken hem de kodlama sırasında oluşabilecek hataların statik analiz ile giderilmesi planlanmaktadır. Statik analizde, kod analiz araçlarının tespit edemediği zafiyetler için gözden geçirme tekniği hibrit modele dâhil edilmiştir. Web servislerinin güvenliğine ait hibrit yaklaşıma ise literatür çalışmalarında rastlanmamıştır. Şekil 3' de hibrit modele ait sistem tasarımı verilmiştir.



Şekil 3. Hibrit model tasarımı

3. Araştırma Bulguları

Web servislerinde yapılması gereken güvenlik testlerinin bazıları web uygulamaları ile paralellik gösterse de kimi test aşamaları web servis mimarisinin doğası gereği kendine özgü aşamalar gerektirmektedir. Servis güvenliğini en üst seviyeye taşımak için tüm güvenlik test tekniklerinin birleştirildiği çalışmada, web servislerine özgü zafiyetlerin gözden geçirme

tekniği ile tespit edilmesi planlanmıştır. Statik kaynak kod analiz araçları ve sızma testlerinin ise hazır saldırı tiplerini tespit edilebilen zafiyetler için kullanılması planlanmıştır.

Web servislerinde dikkat edilmesi gereken en önemli noktalardan birisi de kimlik denetiminin yapılmasıdır. Servislerde kimlik doğrulaması sunucu ve istemci taraflı yapılabilmektedir. Sunucu taraflı yapılan doğrulama, kullanıcıların güvenliğini sağlamak için SSL/TLS güvenlik protokollerini kullanmaktadırlar. İstemci tarafında yapılan kimlik doğrulaması ise geliştirilen uygulamalara bağlı olarak kullanılan uygulama çatılarına ve geliştiricilere göre farklılık gösterebilmektedir. SOAP başlığı kullanılarak gerçekleştirilen kimlik denetiminde belirlenen kullanıcı adı ve şifre ile kullanıcı girişi güvenli bir şekilde gerçekleştirilebilmektedir. SOAP paketindeki başlık bilgisi güvenli bir anahtar barındırmaktadır. Web çağrılarında iletilen içerik, şifrelenmiş olsa dahi başlık hangi şifreleme algoritmasını kullandığını karşı tarafa iletir ve anahtar bilgisiyle güvenli mesajlaşma ortamı oluşturulur(Charfi and Mezini, 2015).Asp.Net ile geliştirilen örnek uygulamada klasik web servislerinde SOAP başlığı kullanılarak geliştirilen kimlik denetimine ait örnek kod satırları Tablo 1' de verilmiştir.

Tablo 1.SOAPbaşlığı ile kimlik denetimi

```
public class ServisGuvenligi : System.Web.Services.WebService{
    public Yetki KimlikDenetleme; [WebMethod] [SoapHeader("KimlikDenetleme")]
    public string Denetim(){
      if (KimlikDenetleme != null){
        if (KimlikDenetleme.KullaniciAdi == "TestKullanicisi" && KimlikDenetleme.Sifre == "E03DF077-0A3A-4C81-94C3-
9993103D298D"){
          return "Giriş Başarılı";}
        else
          return "Servis Kullanıcı adı veya Şifre Yanlış.";}
      else
        return "Servis Yetkisi yoktur.";}}
  public class Yetki : SoapHeader{
    public string KullaniciAdi;
    public string Sifre;} }
```

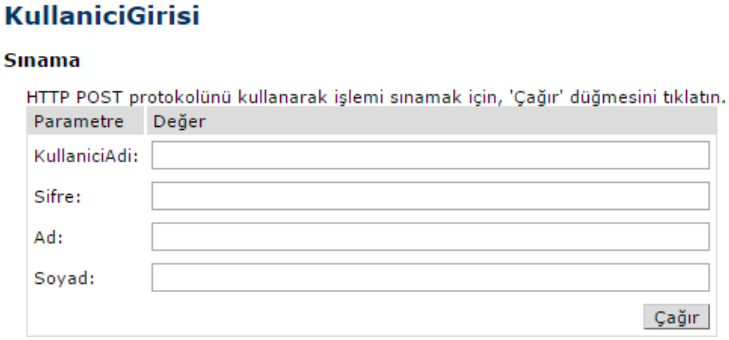
Web API uygulamalarında kimlik denetimi yapabilmek için jeton bazlı bir yetkilendirme yapılmaktadır. Uygulama çatılarının kendilerine özgü hazır güvenlik paketlerinin yanında, SOAP servislerinde de kullanılabilecek kullanıcı denetimi yapabilen fonksiyonlar geliştirilebilmektedir. Web servislerindeki kimlik denetimi kapsamında alınabilecek önlemler arasında özel güvenlik denetim metotları da bulunmaktadır. Çalışma kapsamında geliştirilen kontrol metodu Tablo 2' de verilmiştir.

Tablo 2. Özel metot ile kimlik denetimi

```
[WebMethod]
public string KullaniciDenetimi(string KullaniciAdi, string Sifre,string Ip,string ProjeAdi)
    {
      if(IpVarMi(Ip) && ProjeVarMi(ProjeAdi))
      {
        if (KullaniciAdi == "KullaniciAdi" && Sifre == "E03DF077-0A3A-4C81-94C3-9993103D298D*ETRY2962Qeer-
XV2A-58rT1-64G3-3454634ASadD564hD")
  return "Giriş Yapılmıştır";
else
return "Kullanıcı Adı veya Şifre Yanlış.";
      }
      else
        return "Ip veya Proje Adı Yetkilendirilmemiştir.";
    }
```

Kontrol fonksiyonunda Ip ve proje adı kontrolü yapılarak yetkisiz kişilerin servisi kullanması engellenmiştir. Kullanıcı adı ve Şifre ise Md5 özet fonksiyonu ile şifrelenip gönderilmekte ve dinlemelere karşı şifre çalma riski azaltılmaya çalışılmıştır.

Servislerde dikkat edilmesi gereken diğer bir nokta, uygulama dillerine bağlı olarak servislerin dışarıdan bilgi almamasıdır. Http-Post ve Http-Get protokolleri web servislerinde aktif edildiği durumlarda istenmeyen bilgi çağırmalarına maruz kalabilmektedir (Microsoft, 2015). Girilebilecek rastgele değerler ile zafiyet oluşturulabilmektedir. Örneğin Asp.Net web servislerinde, oluşturulan servisler eğer parametre alıyorsa, bu parametreler dışarıdan müdahaleye maruz kalabilmektedirler. Şekil 4' de örnek bir servis kullanımı verilmiştir.



Şekil 4. Web servisi parametre giriş ekranı

Parametre giriş ekranını kapatmak için Tablo 3' deki kod satırı web.config dosyasına eklenmelidir.

Tablo 3. Parametre alanlarını gizleme

```
<webServices>
<protocols>
<remove name="HttpPost" />
<remove name="HttpGet" />
</protocols>
</webServices>
```

WSDL taraması, sunuculardaki web servislerinin neler olduğunu, hangi fonksiyonları barındırdığını ve geri dönüş tiplerini tespit etmek için saldırganlar tarafından yaygın bir şekilde kullanılmaktadır. Tarama yöntemi, parametre kurcalama, zararlı içerik enjeksiyonu, komut enjeksiyonu gibi ciddi saldırıları gerçekleştirmek için kullanılmaktadır (Acar, 2012). Bu bilgilerin görüntülenmemesi için web.config dosyasına Tablo 4' deki kod satırları eklenmelidir.

Tablo 4. Dokümantasyon Kapatma

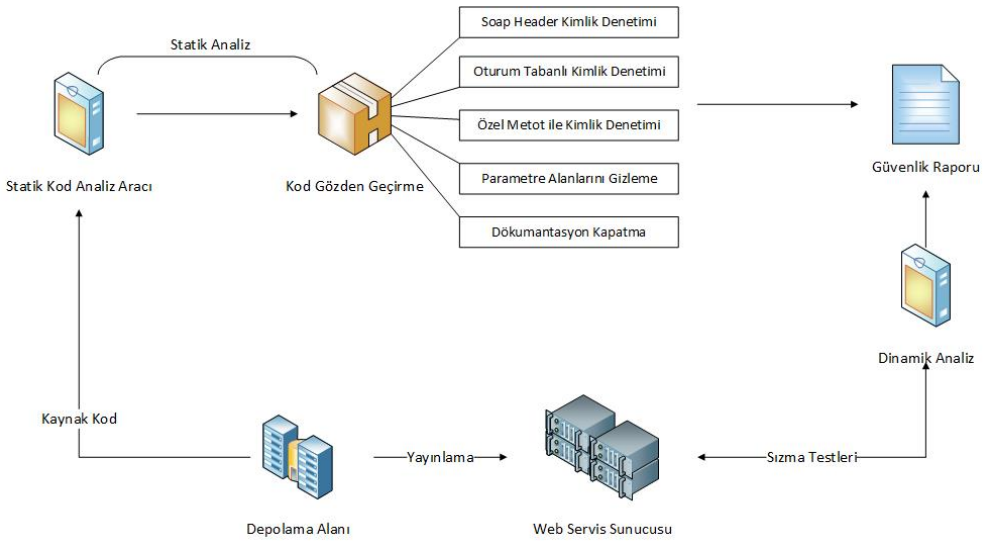
```
<webServices>
<protocols>
<remove name="Documentation"/>
</protocols>
</webServices>
```

Web servislerindeki yetkilendirme kurallarının yanında oturum kontrolü ile servis denetiminin yapılması, güvenlik önlemlerini bir kat daha artırmaktadır. Kullanıcı adı, şifre, oturum bilgisi ve api anahtarı adres satırında görülmediği için web servislerinin oturum tabanlı kimlik denetimini kullanmaları önerilmektedir (Oftedal and Stock, 2015). Web servisinde kullanıcı adı ve şifreyle giriş yapılarak oluşturulan oturum kodu ile metot kullanımı sağlanabilmektedir. Oturum kodu kullanılarak geliştirilen güvenli servis metoduna ait örnek kod satırları Tablo 5' de verilmiştir.

Tablo 5. Oturum tabanlı kimlik denetimi

```
[WebMethod]
        public string OturumAc(string KullaniciAdi, string Sifre)
        {
             if (KullaniciAdi == "KullaniciAdi" && Sifre == "E03DF077-0A3A-4C81-94C3-9993103D298D*ETRY2962Qeer-
XV2A-58rT1-64G3-3454634ASadD564hD"){
                 string kod = new Guid().ToString();
                 if(Session["OturumKodu"]!=null) Session["OturumKodu"] = kod;
                 else Session.Add("OturumKodu",kod);
                 return Session["OturumKodu"].ToString(); }
             Else return "Kullanıcı Adı veya Şifre Yanlış."; }
        [WebMethod]
        public string KullaniciGirisi(string OturumKodu)
        { if (Session["OturumKodu"]!=null){if(Session["OturumKodu"].ToString()==OturumKodu) { return "Giriş Yapılmıştır";
}Else  return "Kullanıcı Adı veya Şifre Yanlış.";
           } else return "Oturum Açılmamıştır.";}
```

Ayrıştırma saldırıları, servislere gelen isteklerdeki elemanları ve bunlara ait değerlerin kullanılmasıyla ortaya çıkmaktadır. Bu saldırı türünde belirlenen bir elemanın çok sayıda kopyası kullanılarak XML ayrıştırıcının çekirdek ve hafıza sınırları zorlanarak ayrıştırıcı zafiyeti oluşturulmaya çalışılır. Bu tür saldırılar için girdi denetimi yapılarak, geliştirme sırasında önlem alınabilmektedir. Girdi denetimleri ise statik kod analizi ile güvenlik zafiyetlerini tespit etmek için kullanılmaktadır. Girdi denetimiyle web servislerindeki satırlar zafiyetlerin türüne göre taranarak, SQL, Başlık, XPATH enjeksiyonları, XSS, CSRF (Cross-Site Request Forgery), Dosya Manipülasyonu gibi saldırılar tespit edilebilmektedir. Çalışma kapsamında, girdi denetimi eksikliğinden meydana gelen hatalar statik kod analiz araçlarıyla tespit edilerek raporlanmıştır. Web servislerinde, hibrit model uygulunarak analiz araçlarının tespit edemediği zafiyetler gözden geçirme yöntemiyle tespit edilerek yüksek seviyede güvenlik sağlanmaktadır. Şekil 5' de web servislerine ait hibrit modelin uygulama şeması gösterilmiştir.



Şekil 5. Hibrit model uygulama şeması

Web servislerindeki güvenlik zafiyetlerini tespit etmek ve daha güvenli duruma getirmek için geliştirilen hibrit modelin geçerliliğini test etmek adına Asp.Net ' de geliştirilen açık kaynak 10 web servis projesi kullanılmıştır. Servisleri test etmek için ise ücretsiz yazılımlar kullanılmıştır. Modeldeki gözden geçirme bölümü için de 10 projenin kaynak kodları incelenmiştir.

Gerçekleştirilen testlerde kullanılan projeler Github' ta kodları paylaşılmış ve piyasada kullanılan projelerdir. Güvenlik testlerinde statik kod analiz tekniği kullanılmıştır. Statik kod analiz araçlarıyla gerçekleştirilen testlerde Sql enjeksiyonu, XSS ve dosya manipülasyonu gibi zafiyetler tespit edilirken web servislerindeki güvenlik açıklarının ise sadece belirli bir bölümü giderilebilmiştir. Hibrit modelde açıklanan gözden geçirme yöntemiyle incelenen test projelerinin bazılarında kimlik denetimi yapılırken bazılarında sadece ayar dosyalarındaki düzenlemeler yapılmıştır. Tablo 6' da test sonucunda elde edilen değerler verilmiştir.

Tablo 6. Test verileri

Zafiyet Türü/Sayısı	AppCodeScan ve VCG Yazılımları	Gözden Geçirme Yöntemi	Toplam Zafiyet (MAX)
XSS	52	56	56
SQLI	0	10	10
Dosya Manipülasyonu	5	5	5
Girdi Doğrulama	4	2	4
İstek Sahteciliği	4	3	4
Kimlik Denetimi (SOAP Header)	1	1	2
Kimlik Denetimi (Oturum Bazlı)	0	2	2
Kimlik Denetimi (Özel Metot)	0	2	2
Parametre Alanları Gizleme	0	4	4
Dokümantasyon Gizleme	0	1	1

Tablo 6' daki sonuçlara göre otomatize araçların tespit edebildiği bazı zafiyetlerin gözden geçirmeyle tespit edilemediği ve gözden geçirmeyle tespit edilen zafiyetlerin bazılarınınaynı şekilde yazılımlarla tespit edilemediği gözlemlenmiştir. Test projelerinde gözden geçirme yönteminde incelenen zafiyetlerin projelerdeki tespit edilme sayıları Tablo 7' de verilmiştir.

Tablo 7. Gözden geçirme yoluyla tespit edilen web servis açıklıkları

Proje No	Soap Header Kimlik Denetimi	Oturum Bazlı Kimlik Denetimi	Özel Metot Kimlik Denetimi	Parametre Gizleme	Dokümantasyon Gizleme
1	1	0	0	1	0
2	0	0	1	0	1
3	0	0	0	1	0
4	0	1	0	0	0
5	0	0	0	0	0
6	0	0	1	1	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	1	0	0	0
10	0	0	0	1	0

4. Sonuç ve Öneriler

Test sonuçlarında gözden geçirme yönteminin otomatize araçlara göre bazı zafiyetleri tespit etmekte daha iyi olduğu görülmüştür. Web servislerinde olması gereken bazı güvenlik önlemleri ise otomatize araçlarla bulunamamıştır. Aynı şekilde otomatize araçların da tespit ettiği bazı zafiyetler gözden geçirme yönteminde tespit edilememiştir. Her iki yönteminde birbirlerinin bulduğu zafiyetleri tespit etmesinin yanında birbirlerinin bulamadığı zafiyetleri de tespit ettikleri gözlemlenmiştir. Geliştirilen hibrit yaklaşımın en önemli avantajı da bu noktada ortaya çıkmaktadır. Ayrı ayrı değerlendirildiğinde hem otomatize araçlar hem de gözden geçirme tek başına yeterli gibi görülse de birlikte kullanıldıklarında daha fazla zafiyetin ortaya çıkarıldığı tespit edilmiştir. Her iki yönteminde kendisine ait dezavantajları ve eksik olduğu yerler bulunmaktadır. Otomatize araçlarla gerçekleştirilen analizler zamandan tasarruf sağlarken diğer taraftan güncel zafiyetler ve uygulama çatılarına uyumluluk gibi yeni platformlardaki zafiyetlerin tespit edilmesinde yetersiz kalmaktadırlar. Gözden geçirme yöntemi ise birebir inceleme gerçekleştirip yanlış positif hataları göz ardı edebilirken yazılım geliştirme ve test aşamalarında çok fazla zaman kaybedilmesinden dolayı her zaman tercih edilmemektedir. Ayrıca testi gerçekleştiren kişinin bilgisine ve dikkatine göre test edildiği için eksik sonuçlarda çıkarabilmektedir. Hibrit yaklaşımla bu eksiklikler en aza indirgenmiştir. Web servislerinde sıkça tercih edilen kimlik denetim yöntemi SOAPbaşlığı, yazılımlarla kontrol edilebilirken diğer kimlik denetimleri geliştiriciye özel olduğu için otomatize araçlarla tespit edilememiştir. Hibrit modelde devreye giren gözden geçirme yöntemi ile oturum bazlı ve özel metotlarla gerçekleştirilen kimlik denetimleri test projelerinin bir kaçında tespit edilmiştir.

Tablo 6' daki sonuçlara göre gözden geçirme yöntemiyle tespit edilebilen zafiyetler sayıca otomatize araçlara göre daha fazla olmasına karşın her iki yönteminde ayrı ayrı tespit edebildiği ve tespit edemediği zafiyetler bulunmaktadır. Bu sebeple iki yöntemden birinin daha iyi olduğu söylenememektedir. Tablo 6 daki sonuçlara göre, birlikte kullanıldığında ise

en iyi sonucun elde edildiği tespit edilmiştir. Gözden geçirme yönteminde önerilen ve web servis güvenliğine özgü kimlik denetimi, parametre kontrolü, dokümantasyon kapatma gibi zafiyetler son zamanlardaki güvenlik raporlarında web servislerinde en sık karşılaşılan zafiyetlerlistesinde olup, günümüzde hazır ticari yazılımların birçoğu iletespit edilememektedir. Test edilen projeler dikkate alındığında, servislerdeki kimlik denetim uygulamalarının dikkate alınmadığı ve uygulamaların %60' ı ve % 90' ı arasında güvenlik denetimlerinin yapılmadığı Tablo 7' de görülmektedir. Tablo 7' de verilen sonuçlara göre web servis güvenliğinin genel olarak projelerde ön planda tutulmadığı, Tablo 6' da ise web servislerinde sık görülen zafiyetlerin uygulama çatılarına bağlı olması ve geliştiriciye özgü olmasından dolayı gözden geçirme yöntemiyle tespit edildiği, hazır yazılımlarla tespit edilemediği görülmektedir.

Web servislerinin güvenlik testleri için uygulanan hibrit model geliştiricilere yeni bir yaklaşım sunmuştur. Önceki çalışmalarda güvenlik testleri tek taraflı yöntemlerle yapılırken elde edilen sonuçlar daha çok bilinen açıklıkları tespit edebilmekteydi. Bu modelde ise yapılan testler bazı zafiyetlerin veya güvensiz kodların gözden geçirme yöntemiyle kesin olarak tespit edilebildiği gösterilmiştir. İlerleyen çalışmalarda ise java ve php gibi diğer web programlama dillerinde de servis zafiyetlerini tespit edilebilmesi hedeflenmektedir.

Teşekkürler

Bu çalışma Süleyman Demirel Üniversitesi Bilimsel Araştırma Projeleri Yönetim Birimi tarafından 3888-D1-14 No`lu projeyle desteklenmiştir.

Kaynaklar

Alparslan, E. (2009). Güvenli Yazılım Geliştirme Modelleri. http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/guvenli-yazilim-gelistirme-modelleri.html (2013.08.10).

Antunes, N., Vieira, M. (2009). Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services.15th IEEE Pacific Rim International Symposium on Dependable Computing, 16-18 Kasım,China, 301-306.

Acar, Ö. F. (2012). Web Servisi Güvenliği. http://www.bilgiguvenligi.gov.tr/web-guvenligi/web-servisi-guvenligi.html (2015.10.21).

Burak, E.(2009). Microsoft'un Güvenli Geliştirme Süreci (SDL). http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/microsoftun-guvenli-gelistirme-sureci-sdl.html (2015.10.15).

Charfi, A., Mezini, M. (2005). Using aspects for security engineering of web service compositions In Web Services, 2005 IEEE International Conference, 11-15 Temmuz, 59-66.

Demir, B.(2013). Yazılım Güvenliği Saldırı ve Savunma, Dikey Eksen Yayın Dağıtım, İstanbul,423s.

Güven, Ö. A. (2015). Web Service (namı diğer SOAP) nedir, ne işe yarar?http://programciyiz.biz/web-service-soap-nedir-ne-ise-yarar/ (2015.10.19).

Güvercin, E.(2015). SOAP ve REST Mimarilerine Genel Bakış. http://www.erenguvercin.com/2013/05/soap-ve-rest-mimarilerine-genel-baks.html (2015.12.12).

Karayumak, F. (2013). Yazılım Güvenliği Programı. http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/yazilim-guvenligi-programi.html (2015.08.01).

Masood, A., Java, J. (2015). Static Analysis for Web Service Security – Tools & Techniques for a Secure Development Life Cycle.Technologies for Homeland Security (HST), 14-16 Nisan, Waltham, 1-6.

McGraw, G.(2006). Software Security: Building Security In. Addison Wesley Professional, USA, 448s.

Michael, C, C. (2005). Risk-Based and Functional Security Testing. https://buildsecurityin.us-cert.gov/articles/best-practices/security-testing/risk-base-and-functional-security-testing (2015.09.16).

Microsoft, (2015). Configuration Options for XML Web Services Created Using ASP.NET.https://msdn.microsoft.com/library/b2c0ew36(v=vs.100).aspx. (2015.12.10).

Oftedal, E., Stock, A. 2015. REST Security Cheat Sheet. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet#Authentication_and_session_management. (Erişim Tarihi: 05.08.2015).

Privacy Rights ClearingHouse.(2005). Privacy Rights. http://www.privacyrights.org/ (2015.11.10).

The Java Web Services Tutorial.(2015). Web Services Security Initiatives and Organizations. https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/Security-WebSvcs6.html. (2015.09.14).

Yılmaz, O. (2010). Web Uygulama Güvenliğine Hibrid Yaklaşım. http://webguvenligi.org/dergi/WebUygulamaGuvenligineHibridYaklasim-Subat2010 OnurYilmaz.pdf. (2015.01.05).

Yue, H., Tao, X.(2012). Web Services Security Problem in Service-oriented Architecture.International Conference on Applied Physics and Industrial Engineering, 4-6 Haziran, Londra, 1635-1641.

Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J.P., Vouk,M.A.(2006). On the Value of Static Analysis for Fault Detection in Software. IEEE Transactionson Software Engineering, 32(4), 240-253.