Web Uygulamalarında Dolaşım ve Erişim Kontrolü Hatalarının Tespiti ve Yeniden Canlandırılması

Yudum Paçin[1] ve Aysu Betin Can[2]

ODTÜ - Enformatik Enstitüsü, Ankara, Türkiye[1,2]
yudum@metu.edu.tr[1], betincan@metu.edu.tr[2]

Özet. Bu çalışmada, dinamik web uygulamalarındaki dolaşım ve erişim hatalarına yol açan senaryoların tespiti ve otomatik olarak canlandırılması için bir araç sunulmaktadır. Sunulan yöntemde ilk olarak verilen web uygulamasının modeli, AJAX tabanlı web uygulamalarını analiz edebilen, web crawler araçları yardımıyla çıkartılmaktadır. Daha sonra, model, web crawler'a olan bağımlılığı azaltmak ve kullanıcılara çıkartılan modeller üzerinde değişiklik yapma imkanı vermek için oluşturulan ara modelleme diline çevrilmektedir. Ardından geliştirilen araç yardımıyla, model denetleme diline çevrilen modele denetlenmesi istenen özellikler kullanıcı tarafından tanımlanmaktadır. Son olarak, model denetleme aracının ürettiği karşı örnekler web tarayıcısı üzerinde canlandırılmaktadır. Bu özellikler ile model denetleme tekniğinin uygulanması ve çıktılarının herhangi bir geçmiş deneyim gerekmeden anlaşılabilmesi sağlanmaktadır. Aracın kullanılabilirliğini ölçmek için bir kullanıcı çalışması yürütülmüştür. Katılımcılar, hata tespiti ve hataların görselleştirilmesini yararlı bulduklarını bildirmişlerdir. Ayrıca, aracın dolaşım hatalarını tespitindeki etkililiği 14 web uygulamasıyla değerlendirilmiş ve 44 gerçek ulaşım hatası tespit edilmiştir. Araç yardımı ile erişim kontrolü hatalarının tespit edilebilirliği ise hata enjeksiyonu metoduyla değerlendirilmiştir. Ayrıca, web uygulamalarının modelleri, uygulamaların sahip olduğu karmaşık ve zengin yapıdan dolayı crawler tarafından eksik çıkartılabildiği için yanlış tespitlerin ortaya çıkabildiği gözlemlenmiştir.

Anahtar Kelimeler: Web uygulamaları, otomatik doğrulama, model denetleme, karşı örneklerin yeniden canlandırılması

1. Giriş

Web uygulamalarının kullanımı artık tüm alanlar için kaçınılmaz bir hale gelmektedir; ticaret, eğitim, sağlık, devlet gibi birçok alandaki hizmetler web uygulamaları aracılığıyla iletilmekte ve web teknolojilerinin hızla gelişmesiyle bu uygulamaların kullanımı giderek artmaktadır [22]. Web uygulamalarında var olan hataların tespiti için etkili bir yöntem ihtiyacı ise giderek önem kazanmaktadır. Yetersiz dolaşım özellikleri web uygulamalarındaki en yaygın hatalardan birini oluşturmaktadır. Gidecek bir sonraki sayfası olmayan, çıkmaz sayfalar (dead end) ve ulaşılamayan web sayfaları kullanılabilirliği azaltan dolaşım hatalardan bazılarıdır. De Alfaro [9] ve Di Sciascio v.d. [12] çıkmaz sayfaların bulunmamasının iyi tasarımın gereksinimlerinden biri olduğunu belirtmişlerdir. Ayrıca, web uygulamalarında

kullanıcılardan yetki bilgisini isteyen servislerin erişim kontrolündeki eksiklikler önemli bir başka hata kategorisini oluşturmaktadır. OWASP 2013 web uygulamaları güvenlik açıkları ilk 10 listesinde [23] 7. sırada yer alan "Eksik işlev seviyesindeki erişim kontrolü" ve CWE 2011 en tehlikeli 25 yazılım hataları listesinde [7] 6. sırada yer alan "Eksik yetkilendirme" maddeleri bu tip erişim hatalarını kapsamaktadır. Bu ve benzeri erişim kontrolü hatalarını ve dolaşım hatalarını önlemek veya tespit etmek için model denetleme kullanılan metotlardan biridir ([9], [11], [14], [15], [16]).

Formel sistem modellerinin, verilen zamansal mantık özellikleriyle otomatik doğrulanması için geliştirilen model denetleme metodu, web doğrulanması için kullanılması avantajlı bir metottur; model denetleme otomatik olarak, kullanıcı desteğine ihtiyaç duymadan, sistem modelinin uzayında kapsamlı arama ve denetleme yapabilmektedir. Ayrıca, model denetleme ihlal edilen her özellik için ürettiği karşı örneklerle hatanın bulunmasını kolaylaştırmaktadır. Fakat, bu getirilerine rağmen, model denetleme, formel metotlar ile ilgili deneyimsiz geliştiriciler için uygulaması zor bir metottur. Ek olarak, web uygulamasının formel modelinin çıkarılması, model denetleme uygulanmasını zorlaştıran başka bir nedendir. Çünkü karmaşık yapılarıyla, web uygulamaları, modellenmesi zor sistemlerdir. Tüm bu nedenler, model denetlemenin web uygulamalarının doğrulanması için tercih edilmemesine neden olmaktadır.

Bu çalışmada,

- model denetleme tekniğini deneyimsiz kullanıcılara arka plandaki formel metotlar olmaksızın uygulatmak,
- dolaşım ve erişim kontrolü hatalarının sembolik model denetleme ile tespit edilmesi amaçlanmıştır.

Geliştirilen araç, web uygulamasının Crawljax [21] ve WebMole'un [19] bir başka versiyonu olan, Micro-Crawler araçlarıyla elde edilmiş statik gösterimini model olarak kabul etmektedir. Model denetleme aracı olarak ise NuSMV[6], bir sembolik model denetleme aracı kullanılmaktadır. Sunulan yöntem web crawler araçlarının çıktısının web modele çevrilmesi, sembolik model denetleme ve karşı örneklerin canlandırılması için otomatik çalıştırılabilir betik üretme işlemlerini birleştirmektedir.

2. Literatür Taraması

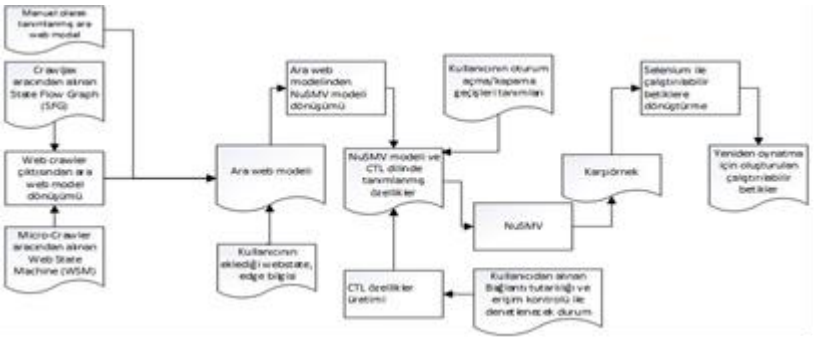
Model denetleme ile web uygulamalarının doğrulaması birçok kez denenmiş bir yöntemdir. Model denetleme araçları, web tasarımı doğrulaması, desteklenmesi ([5], [12], [14], [22]), hata tespiti [1] ve web akışını kontrol etmede [15] kullanılabilmektedir. Web uygulamalarının formel modellerini oluşturmak, amaçlar farklı olsa da, model denetlemeyi kullanan tüm çalışmalarda ortak bir süreçtir. Modelleme, model denetleme yönteminde kilit bir öneme sahiptir, çünkü model bazlı herhangi bir doğrulama işlemi sadece sistemin modeli kadar iyidir [2]. Web uygulamaları genelde dinamik betiklerle yazıldığı ve kullanıcı etkileşimleri uygulamanın akışını dinamik olarak oluşturduğu için, statik analizle yapılmış modelleme dolaşımsal hataların bulunmasındaki potansiyeli sınırlamaktadır [1]. Buna rağmen statik analiz ile, kaynak koddan, elde edilen model web uygulamalarının

kontrol akışını doğrulamakta kullanılabilmektedir [20]. Dinamik analizle modellemeye bir örnek olan Haydar v.d.'nin çalışmasında harici davranışları izlenerek, http istek/cevapları, web uygulamasının modeli yaratılmaktadır [17]. Bazı çalışmalarda ise web uygulaması modeli manuel olarak özel bir modelleme diliyle tanımlanmakta, ardından bu model, model denetleme aracının diline çevrilmektedir ([15], [24]). Ayrıca, var olan bir web uygulamasının modelini web crawler yardımıyla çıkarmak da mümkündür. Fakat, bu çok doğrudan bir süreç değildir [10], çünkü artan etkileşim ve dinamik içerikten dolayı web uygulamalarının bu şekilde modellenmesi zor bir süreç haline gelmektedir. Bu çalışmada, bahsedilen sebepten dolayı AJAX tabanlı web uygulamalarını crawl edebilme kabiliyetine sahip araçlar kullanılmaktadır.

Otomatik web doğrulama araçlarından AnWeb [12], web uygulamalarının tasarımlarını desteklemek için geliştirilmiştir. AnWeb'de model, kullanıcı form doldurma işlemleri için sunucu ve statik sayfaları belirlemek için istemci kodlarından oluşturulmaktadır. Model durumlarını web sayfaları, pencereler ve linkler oluşturmaktadır. WaVer [5] ise web uygulamalarının tasarımını denetlemek amacıyla UML diyagramlarını doğrulayan bir araçtır. NuSMV model denetleme aracının kullanıldığı bu çalışmada, çıktılar UML diyagramında gerekli düzenlemelerin yapılması için kullanılmaktadır. AnWeb ve WaVer'de formel modelleme otomatikleştirilmekte, model denetleme ise yazarlar tarafından hazırlanan özelliklerle uygulanmaktadır. Miao ve Zeng ise özelliklerin üretimini verilen tasarım diyagramı yardımıyla otomatikleştirmektedir [22]. Denetlenecek model ise web uygulamasının implementasyon modelini temsil etmekte ve manuel olarak tanımlanmaktadır. [22]'de tasarım modeli, özellikler için bir taban oluştursa da kullanıcı özellik tanımlama işleminin dışında kalmıştır. Denetlenecek özelliklerin kullanıcı tarafından tanımlandığı çalışmalar da mevcuttur ([4], [11], [17]). Bu yolla, deneyimsiz kullanıcılar için model denetleme aracının uygulaması daha anlaşılabilirleştirilse de, bu uygulamanın çıktısının anlaşılması için bir çeviri işlemi yapılmadığı gözlenmektedir. Hata çıktısının anlaşılabilirliğini gözeten bir çalışmaya örnek VeriWeb'dir [3]. VeriWeb, interaktif web uygulamalarındaki tüm yürütme yollarını keşfetmek ve analiz etmek için geliştirilen bir araçtır. Bu çalışmada VeriSoft'u kullanarak belirlenemezci bir algoritmayla arama ve keşfetme işlemi gerçekleştirilmekte, ayrı bir bileşenle yürütme yollarındaki hatalar tespit edilmekte ve WebVCR adlı kaydet/oynat aracıyla canlandırılabilmektedir. Fakat hata oynatma, otomatik olarak yapılmamakta bunun yerine kullanıcıya bir seçenek olarak sunulmaktadır.

3. Yöntem

Geliştirilen aracın ayrıntılı çalışma akış şeması Şekil 1'de gösterilmektedir. Süreç Crawljax veya Micro-Crawler'dan web modelin çıkartılması ile başlamaktadır. Bu iki web crawler web uygulamalarındaki durum değişiklerini URL değişimlerinden değil döküman nesne modelindeki (DOM) yapı farklılıklarından ayırt edebilmektedir. Crawljax'ın ürettiği web uygulaması modeli State Flow Graph (SFG), Micro-Crawler aracının ise Web State Machine (WSM) olarak adlandırılmaktadır.



Şekil 1. Yönetimin işleyiş diyagramı

Elde edilen model, geliştirdiğimiz ara web modelleme diline çevrilmektedir. Bu dil, WSM ve SFG'nin ortak elemanlarını kapsamaktadır. Bu dilde oluşturulan model (ara web modeli) basitçe XML tabanlı bir sonlu sistem modelidir. Bu model web uygulamasında oluşabilen her farklı DOM ağacını ifade eden webstate kümesi ve bu webstate'ler arasındaki tıklanabilir elemanlar ile yapılan geçişlerin kümesinden oluşmaktadır.

```
1       <? xml version='1.0' encoding='UTF-8' ?>
2       <webstates>
3       <webstate>
4       <name>
5       index
6       </name>
7       <url>
8       http://localhost/index.html
9       </url>
10      </webstate>
        ……
83      </webstates>
84
85      <edges>
86      <edge>
87      <from>index</from>
88      <to>state1</to>
89      <XPathOfElement>
90      /HTML[1]/BODY[1]/P[1]/A[1]
91      </XPathOfElement>
92      </edge>
        ……
        </edges>
```

Şekil 2. Ara web modelinin bir kısmı

Şekil 2'de örnek bir ara web modeli gösterilmektedir. İlk satır webstate tanımını ifade etmektedir. Bu formatta, bağlantı ilişkisi <edge> ve </edge> etiketleri arasında verilmiştir ve her biri, w_i, <from> etiketiyle tanımlanan bir webstate, w_j, <to> etiketiyle tanımlanan webstate ve w_i ve w_j arasındaki geçişleri tetiklemek için tıklanan elementin XPath ifadesini içeren etiketlerinden oluşturulmuştur. Şimdilik, geçiş sadece tıklama olayıyla tanımlanmıştır, geri tuşu ise dikkate alınmamıştır. Bu kısıtlamanın sebebi ise web crawler tarafından üretilen sonlu sistem modellerinden kaynaklanmaktadır.

Web crawler modelinin ara web modele çevrilmesinin ardından kullanıcılar eksik olduğunu düşündükleri webstate veya geçiş tanımlarını aracın arayüzü yardımıyla ekleyebilmektedir. Ayrıca, kullanıcı kendi web modelini elle tanımlayabilmektedir.

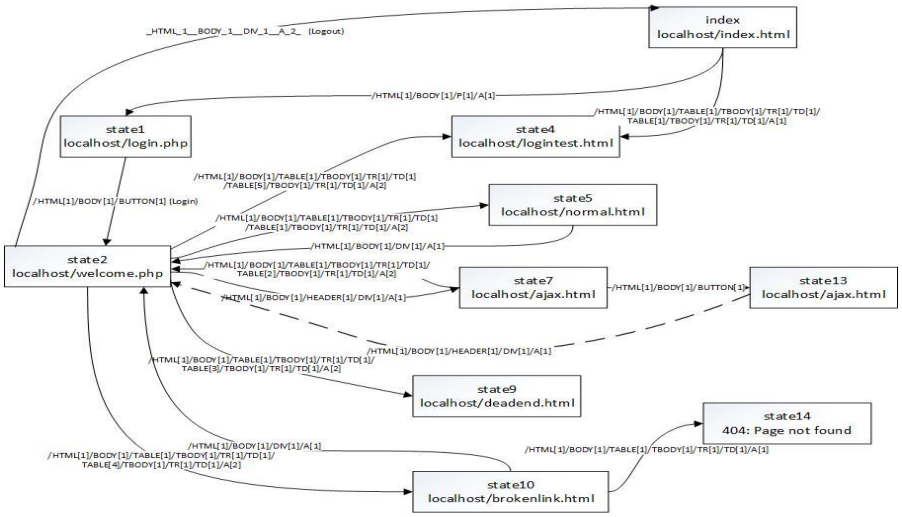
Web crawler aracından alınan web modelin ara web modeline dönüştürülmesinin ardından (Şekil 2) NuSMV modeli yaratılır. Aracımız, denetlenecek özellikleri CTL (Hesaplama Ağacı Mantığı, Computation Tree Logic) formülleri şeklinde tanımlar. CTL özellikleri hakkında detaylı açıklama bölüm 3.3'te, NuSMV modelinin yaratılması ise bölüm 3.2'de verilmektedir. En son aktivite olan karşı örneklerin Selenium Webdriver kütüphaneleri ile web tarayıcı üzerinde yürütülen çalıştırılabilir senaryolara çevrilmesidir. Bu işlem ise bölüm 3.4'te açıklanmaktadır.

3.1 Örnek Uygulama

Bu bölümün geri kalanında, aracın ara web modelinin oluşturulmasının ardından çalışma işleyişi örnek bir uygulama (Şekil 3) üzerinden anlatılmaktadır. Örnek web uygulaması 6'sı php dosyası olmak üzere 12 kaynak dosyasından oluşmaktadır. Uygulama özellikle geliştirilen araçla bulunabilen hataların gösterimi için tasarlanmıştır. Örnek uygulamada, AJAX tabanlı bir sayfa, bir oturum açma sayfası ve oturum açma gerektiren bir kaç sayfa, 2 çıkmaz sayfa ve 1 geçersiz bağlantı bulunmaktadır. Bu uygulamanın ara web modeli Şekil 2 ile gösterilmiştir. Bu ara web model, Crawljax aracından çıkan modelden dönüştürülmüştür, çünkü oturum açma için gereken form doldurma işlemi, kullanılan crawler araçlarından sadece Crawljax tarafından desteklenmektedir. Şekil 3'teki kesikli çizgiler Crawljax'ın yakalayamadığı geçişleri temsil etmektedir. Bu geçiş, aracımızın kullanıcı arayüzü yardımı ile ara web modeline eklenmiştir.

Şekil 3'te görüldüğü üzere, uygulamada toplam 10 webstate bulunmaktadır. State1 adlı webstate içinde oturum açma işlemi gerçekleşmektedir. Webstate'leri birbirine bağlayan kenarların üzerindeki XPath ifadesi, kenarın başlangıcındaki webstate üzerinde bulunan ve geçişe neden olan tıklanabilir elemanın DOM ağacı üzerindeki adresini vermektedir. Örnekte yalnızca index sayfası tüm kullanıcılara açıktır. State4 ise yetki gerektiren bir sayfa olması gerekirken, oturum açılmadan da erişilebilmektedir. Bu hatanın erişim kontrolü özelliği ile tespit edileceği beklenmektedir. State7 ve state13 aynı URL adresine sahiptir. Bu iki webstate arasındaki geçiş JavaScript olayı ile sağlanmaktadır. State10 geçersiz bağlantı içeren bir web sayfasını temsil etmektedir. State9 ve state4 ise hiçbir tıklanabilir eleman bulundurmayan çıkmaz sayfalardır.

Model çıkarım sırasında tüm webstate kümesi web crawler tarafından tanımlanmıştır, ancak geçişlerden, state13'ü state2'ye bağlayan geçiş tespit edilememiştir. State13 DOM ağacının asenkron değişmesini saylayan bir AJAX olayı ile ortaya çıkan bir durumdur. Bu gibi durumlarda, Crawljax sadece değişen kısımlarda yer alan farklı tıklanabilir elemanları tanımlayabilmektedir. Örnek web uygulamasında, state2'ye giden bağlantının olduğu kısım yenilenmediği için, bu bağlantı State13'teki haliyle aynı kalmış, bu sebeple, bağlantı ve dolayısıyla state2'ye giden geçiş Crawljax tarafından tanımlanamamıştır. Crawljax'ın tanımlayamadığı bu geçiş, sonradan eklenmeseydi, bu durum state13'ün çıkmaz sayfa olarak tespit edilmesine neden olacak ve yanlış alarma yol açabilecekti.



Şekil 3. Örnek web uygulaması ara web modeli gösterimi

3.2 NuSMV modelinin oluşturulması

Ara web modelinin NuSMV modeline dönüştürülmesi yalnızca sözdizimsel bir işlem değildir. NuSMV dilindeki web modelinde bir tuzak durumu bulunmaktadır, ayrıca NuSMV modelinde, ara web modelinden farklı olarak, durumların erişimlerini kısıtlayıcı, koruma koşulları bulunabilmektedir.

Ara web modeli yalnızca uygulama içindeki dolaşımı gösterebilir. Erişim kontrolü ve yetki hatalarını yakalamak için gerekli bilgiye sahip değildir. Bunun için kullanıcılardan oturum açma ve kapamanın gerçekleştiği durumları tanımlaması istenmektedir. Bu çalışmada oturum açma durumundan oturum kapamaya uğramadan ulaşılabilen tüm durumlara özel durum adı verilmiştir.

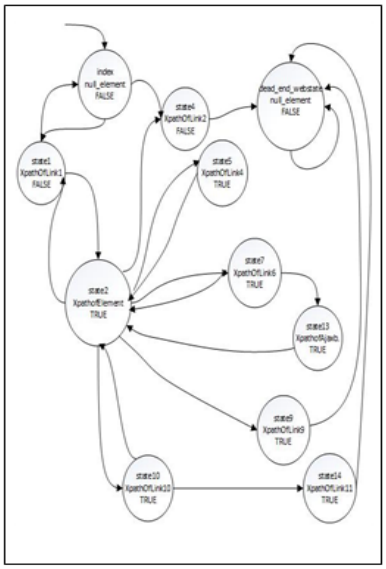
NuSMV web modelinde oturum açılıp açılmadığı bilgisini için boole değişkenler kullanılmıştır. Oturum açmak bir koruma koşulu olarak düşünülmektedir. Koruma koşulları rol bazlı yetkileri tanımlamak için de kullanılabilmektedir.

NuSMV web modelinde bulunan bir diğer bileşen ise tuzak durumudur. Model denetlemede kullanılan yapılarda her durumdan en az bir geçiş ilişkisi bulunmalıdır. Buna uyum sağlamak için dead_end_webstate durumu bir tuzak olarak webstate kümesine eklenmiştir. Bu sayede her webstate için gidecek bir sonraki webstate olması sağlanmıştır. Yukarıda belirtildiği üzere, tüm geçişler tıklanabilir bir sayfa elemanıyla tetiklenmektedir. Bu yüzden tuzak durumuna geçiş elemanı olarak null_element tanımlanmıştır.

Şekil 4 örnek web uygulamasının NuSMV web modelinin bir kısmını vermektedir. NuSMV girdi diliyle tanımlanan bu model, Şekil 2'deki ara web modelinden otomatik olarak üretilmiştir. NuSMV web modelinin grafiksel gösterimi Şekil 5'te verilmiştir.

```
1    MODULE main
2    VAR
3    login:boolean;
4    webstate:{index,state2,…,
dead_end_webstate};
5   element:{_HTML_1__BODY_1__P_1__A_1_,…,
null_element};
7    ASSIGN
8    init(login):=FALSE;
9    init(webstate):=index;
10   init(element):=null_element;
11   next(webstate):= case
…
13     webstate=state1 & next(login) = TRUE &
next(element)=_HTML_1__BODY_1__FORM_1__INPUT_3_:
state2;
26     TRUE: dead_end_webstate;
27   esac;
28   next(element):= case
29   webstate=state1  :
{_HTML_1__BODY_1__FORM_1__INPUT_3_};
…
35
webstate=state10:{_HTML_1__BODY_1__DIV_1__A_1_,_
HTML_1__BODY_1__TABLE_1__TBODY_1__TR_1__TD_1__A_
1_};
36
webstate=dead_end_webstate:null_element;
37     TRUE: element;
38    esac;
39   next(login):= case
40     webstate=state1 & next(element) =
_HTML_1__BODY_1__FORM_1__INPUT_3_ : TRUE;
41     webstate=state2 & next(element)=
_HTML_1__BODY_1__DIV_1__A_2_:FALSE;
42    TRUE: login;
43    esac;
```

Şekil 4. Örnek NuSMV modeli (bir kısmı)



Şekil 5. Örnek Kripke yapısı diyagramı

Şekil 4 ile görüldüğü üzere sistem bir ana modül ile tanımlanmıştır. Bu modül isimleri webstate, element ve login olan 3 değişken içermektedir. Webstate değişkeninin değer kümesi (satır 4) ara web modelde tanımlanmış webstate isim değerleridir. Element değişkeninin değer kümesi ara web model'de <edge> etiketinin içinde bulunan <XpathOfElement> etiketine ait XPath ifadelerinin NuSMV girdi diline uyarlanmış halidir. Aracımız XPath ifadesindeki köşeli parantez ve ters taksim işareti NuSMV diline uyum sağlamak için altçizgi işaretiyle değiştirmiştir.

ASSIGN bölümünde öncelikle değişkenlerin ilk değerleri init yapısı ile atanmıştır (satır 8-10). Element değişkenine, henüz hiçbir elemente tıklanmadığını belirtmek için, null_element değerini atanmıştır. Webstate değişkeninin ilk değeri uygulamanın açılış sayfası olan index'tir. Koruma koşulu değişkeni login, false değerini almıştır.

İlk değerlerin tanımlanmasının ardından, her bir değişkenin bir sonraki adımda alacağı değeri case yapısı içinde geçiş ilişkileri ile tanımlanmaktadır. Case yapısında her bir kısım bir koşul, noktalı virgül ve değişkenin alacağı değerden oluşmaktadır. TRUE kelimesiyle başlayan son case koşulu diğer hiçbir koşulu sağlamayan değişken değerlerinin aldığı değeri, (default value) belirtmektedir.

Şekil 4'te 11 ve 27. satırlar arasında tanımlanan webstate değerinin geçiş ilişkileri <edge>,</edge> etiketleri içinde tanımlanan bilgilere göre tanımlanmaktadır. Ara web model anlatımında değinildiği gibi geçişler <from>, <to> ve <XpathOfElement> alt öğelerinden oluşmaktadır. Webstate case koşulları <from> etiketi ve değiştirilmiş XPath ifadesiyle eşleşen elemanın tıklanma olayından oluşmaktadır. Buradaki next ifadesi eleman değişkeninin bir sonraki adımdaki değerini gösterir. Eleman değeri modeldeki bir webstate değerinden ötekine geçmek için gerekli olan tıklanabilir eleman bilgisini vermektedir. Bu yüzden, eleman değişkeni bir önceki webstate içindeki son tıklanan elemanı vermektedir. Son olarak, hiç bir geçiş sağlanamadığında tuzak durum olarak dead_end_webstate geçişini tanımlanmaktadır (Satır 26).

Eleman değişkeninin bir sonraki değeri o anki webstate değerinin, DOM ağacındaki tıklanabilir elemanlar arasından rastgele seçilmektedir (Satır 28-36). Eğer webstate dead_end_webstate değerini almışsa, sistemin tuzak durumunda kalmasını sağlamak için eleman değeri null_element olarak sabitlenmektedir (satır 36).

Örnek web uygulamasında bir oturum açma koşulu bulunmaktadır. Bu koşul modele kullanıcının aracımız yardımıyla oturum açma ve kapama geçiş ilişkilerini tanımlamasıyla eklenmiştir. Oturum açma Şekil 5 satır 40 ile verilmektedir. Bu bilgi ayrıca satır 13'ü de etkilemektedir. Satır 13 ile başarılı bir oturum açma işlemiyle state2'ye ulaşılabileceği bilgisi eklenmektedir. Oturum kapama ise satır 41'de tanımlanmaktadır.

3.3 Denetlenen Özellikler

Web modeli üzerinde sorgulanacak özelliklerinin CTL formülleri Tablo 1 ile verilmektedir. Bu formüllerin üretimi, kullanıcıların CTL yapısını anlamasına gerek duymaksızın otomatik veya kullanıcı kılavuzluğu ile yarı otomatik gerçekleşmektedir. Ulaşılabilirlik kategorisini denetleyen dolaşım ile ilgili CTL formülleri tamamen otomatik olarak üretilmektedir. Bağlantı tutarlılığı ve erişim kontrolü için formüller, kullanıcıdan ilgili webstate bilgileri istenerek yarı otomatik üretilmektedir.

Ulaşılabilirlik özellikleri anasayfaya ulaşım ve çıkmaz web sayfalarının oluşmamasını denetlemektedir. Sadece tarayıcının geri tuşuyla yönlendirilmesi mümkün olan, özel uzantılı (.jpg, .pdf gibi) veya sadece harici bağlantı bulunduran web sayfaları da çıkmaz sayfa olarak kabul edilmiştir. Bu özellikler ayrıca daha önce [9], [11], [13] ve [18] çalışmalarında da model denetlemede kullanılmıştır.

Tablo 1. Dolaşım ve erişim kontrolü özellikleri ve CTL formülleri

Kategori	CTL Formülü	Açıklama
Anasayfa (index) ulaşılabilirliği (otomatik)	Her bir webstate $state_i$ için , AG(webstate=$state_i$->EF (webstate= index))	Anasayfa ulaşılabilir diğer tüm durumlardan ulaşılabilirdir.
Çıkmaz sayfa yokluğu (otomatik)	Her bir webstate $state_i$ için , AG(webstate=$state_i$-> !EX(dead_end_webstate))	Tüm ulaşılabilir durumlardan başka bir duruma giden bir geçiş vardır.
Bağlantı Tutarlılığı (yarı otomatik)	AG(webstate=$state_1$-> EF(webstate=$state_2$))	$State_1$ durumdan $state_2$ duruma ulaşım her zaman mümkündür. (kullanıcı $state_1$ ve $state_2$ durumlarını seçer)
Erişim Kontrolü (yarı otomatik)	!EF(webstate=$state_1$& login=FALSE)	Özel durum $state_1$ oturum açılmadan ulaşılamaz. (kullanıcı $state_1$ durumunu seçer)

Tablo 1'de indexe ulaşım özelliği AG(EF(webstate=index)) olarak da tanımlanabilirdi. Fakat, bu durumda NuSMV sadece bir karşı örnek üretecekti. İhlal edilen her webstate için bir karşı örnek üretilmesini sağlama adına, her bir webstate için bir CTL özelliği tanımlanmıştır. Aynı konu çıkmaz sayfa özelliği için de geçerlidir.

Örnek web uygulamasının modelinin, belirtilen CTL formülleri ile model denetleme aracıyla yürütülmesi 7 karşı örneğin oluşmasına neden olmuştur. Index ulaşılabilirliği, state9, state4 ve state14 tarafından, çıkmaz sayfa yokluğu aynı webstate'ler tarafından ihlal edilmiştir. Erişim kontrolü özelliği ile ise aslında özel bir durum olması gereken state4'ün oturum açılmadan erişilebildiği tespit edilememiştir.

3.4 Karşı örneklerden çalıştırılabilir senaryo üretimi

NuSMV karşı örneklerinden JUnit kodu üretimi, hazırlanan şablon dosyadan yararlanılarak yapılmaktadır. Şablon dosyada Selenium ve JUnit paketlerinin içeri aktarım satırları, setup ve teardown metotlarının ortak kodları bulunmaktadır. Setup metodunda tarayıcı ilk kullanıma hazırlanmakta, teardown adlı metotta ise karşı örneği açıklayan diyalog kutusu gösterilmektedir. Karşı örneklerdeki ihlale neden olan adımlar test metodunun içine, şablona sonradan eklenerek tanımlanmaktadır.

Şekil 6, NuSMV tarafından AG(webstate=state4->EF(webstate=index)) özelliğinin modelde ihlal edilmesiyle üretilen karşı örnek vermektedir. Şekil 7, bu karşı örnek ile üretilen JUnit kodunu vermektedir. Ayrıca özellik bilgisi note değişkeniyle teardown metoduna diyalog kutusunu yaratmak için kullanılmıştır. Şekil 6'da hataya neden olan adımlar görülmektedir. Buna göre state4 index sayfasına ulaşamadığı için, state4'e giden yürütme adımları karşı örneği oluşturmaktadır. Bu yüzden Şekil 7'deki test metodunun içinde, index'ten state4'e giden adımlar bulunmaktadır. Test metodu web uygulamasının URL adresinin belirtilmesiyle başlamakta ve hataya yol açan

durumlara karşı örnekte yer alan XPath ifadeleriyle eşleşen tıklanabilir elemanlara tıklanmasıyla devam etmektedir. Karşı örnekte NuSMV koduna uyum sağlaması için değiştirilen XPath ifadesi, JUnit koduna çevrilirken orijinal haline dönüştürülmektir.

```
-- specification AG (webstate =
state4 -> EF webstate = index)
is false
--   as   demonstrated   by   the
following execu-tion sequence
Trace      Description:      CTL
Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  login = FALSE
  webstate = index
  element = null_element
-> State: 2.2 <-
  webstate = state4
  element =
_HTML_1__BODY_1__TABLE_1__TBODY_
1__TR_1__TD_1__TABLE_1__TBODY_1_
_TR_1__TD_1__A_1_
```

Şekil 6. NuSMV tarafından üretilen karşı örneklerden bir tanesi

```
   @Before
  public void setUp() throws
Exception {
driver = new FirefoxDriver();
driver.manage().timeouts().implic
itlyWait(30, TimeUnit.SECONDS);
baseUrl = "http://localhost/";  }
   @Test
  public void test() throws
Exception {
driver.get(baseUrl);
driver.findElement(By.XPath("
/HTML[1]/BODY[1]/TABLE[1]/TBODY[1
]/TR[1]/TD[1]/TABLE[1]/TBODY[1]/T
R[1]/TD[1]/A[1]")).click();   }
  @After
  public void tearDown() throws
Exception {
String note="violation of the AG
(webstate = state4 -> EF webstate
= index)  property, Index is
reachable from all states";
openDialog(note);
driver.quit();}
```

Şekil 7. Otomatik olarak, karşı örneğe göre üretilen JUnit kodu

4. Değerlendirme ve Deneyler

Geliştirilen aracın değerlendirilmesi 3 adımda gerçekleştirilmiştir. İlk olarak aracın kullanılabilirliğini ölçmek için bir anket uygulanmıştır. Ardından, 14 gerçek uygulama üzerinde aracın dolaşım hata tespit etme kapasitesi değerlendirilmiştir. Son olarak erişim kontrolü hatası bulma kapasitesini ölçmek için hata enjeksiyonu kullanılmıştır. Deneyler, Intel Core i5-3210 Windows 7 ve 4GB RAM bilgisayarda sürdürülmüştür.

4.1 Kullanılabilirlik

Kullanılabilirlik çalışmasının uygulandığı grup ODTÜ Enformatik Enstitüsü web admini, Drupal ile ileri derece web geliştirme deneyimi olan bir doktora öğrencisi, ikisi web geliştirme deneyimi olan, biri php-Mysql deneyimi olan yüksek lisans öğrencileri ve web geliştirme deneyimi olmayan 1 yüksek lisans ve 1 lisans öğrencisi

tarafından oluşturulmuştur. Katılımcıların hiçbirinin formel tekniklerle ilgili bilgisi bulunmamaktadır. Aracın, yaklaşık 10 dakika kullanıcılara tanıtılması ve nasıl kullanılacağının anlatılmasının ardından, doğrulama işleminin yapılması istenmiştir. İşlem web crawlerdan çıkan modelin araca yüklenmesi, otomatik olarak üretilen özelliklerle doğulama yapılması, bir bağlantı tutarlılığı özelliği eklenmesi ve bulunan hataların tarayıcı üzerinde oynatılmasından oluşmaktadır.

Hiçbir formel metot deneyimi olmayan kullanıcılar araç sayesinde CTL özelliği eklemeyi ve web uygulamasını model denetleme kullanarak doğrulamayı başarmışlardır. Bu deneyimin ardından katılımcılara 5 soruluk, likert ölçümüne göre hazırlanmış kullanılabilirlik anketinin tamamlanması istenmiştir. Anket, teknoloji kabul modeli (TAM) [8] faktörleri olan, algılanan kullanım kolaylığı ve algılanan yararlılık, gözetilerek hazırlanan 5 sorudan oluşturulmuştur. Sonuçlar (bkz. Ek, Tablo A-2) hata canlandırılmasının yararlı bir işlev olacağını öngörmektedir. Kullanım kolaylığı faktörünün düşük sonuçları ise geliştirilen aracın ticari bir yazılım olmadığı için kabul edilebilirdir.

4.2 Ulaşılabilirlik Hata Tespiti

Ulaşılabilirlik hatalarının tespiti performansının değerlendirilmesi için 14 web uygulaması kullanılmıştır (bkz. Ek, Tablo A-1). Bu uygulamalar php, asp.net ve java gibi farklı programlama dilleri ile geliştirilmiş, Apache veya IIS 6.0 web sunucularında çalışmaktadırlar. Kullanılan web teknolojileri arasında AJAX da bulunmaktadır.

Web uygulamaların modelleri Micro-Crawler aracıyla çıkarıldı. Aracın modeli doğrulaması ardından, toplamda 70 karşı örnek üretilmiştir. Her bir web uygulaması için tüm işlem minimum 203 ms (milisaniye), maksimum 2384 ms sürdü. Süre analizine göre medyan 864 ms, ortalama süre ise 533 ms varyansla 961 ms olmuştur. Karşı örneklerden 26 tanesi gerçek uygulamada hata olmayan yanlış gösterimler olmuştur. Yanlış alarm adını verdiğimiz bu karşı örneklerin web crawler aracının modelinin yetersizliğinden ortaya çıktığı gözlemlenmiştir. Örneğin W7 uygulamasında web crawler index ve bir durum arasındaki bağlantıyı rapor etmemiş, bu da iki tane yanlış karşı örneğe neden olmuştur. Bu durumu incelemek için, 3 tane web uygulaması (W7, W12 ve W14) seçilmiş ve model, tarayıcı üzerinden gösterilen hata senaryoları incelenerek, geliştirilen aracın arayüzüyle modeller rafine edilmiştir.

Tablo 3. Model değişimi sonrası ve öncesi karşı örnek ve yanlış alarm sonuçları

No	Web crawler aracından çıkan model				Değiştirilen model			
	Karşı örnek #	Yanlış alarm #	Durum #	Kenar #	Karşı örnek #	Yanlış alarm #	Durum #	Kenar #
W7	6	2	13	74	4	0	13	75
W12	12	4	16	78	8	0	16	80
W14	8	6	11	29	2	0	11	32

Tablo 3 hata senaryo sayıları, yanlış alarm senaryo sayıları, bulunan durum (state) ve kenar (edge) sayılarını model düzeltmeden önce ve sonraki değerlerini göstermektedir. İlk kısım Micro-Crawler aracının çıkardığı model, diğer kısım ise modelin yanlış alarmlara göre düzeltilmiş model için verilmiştir. 15-20 dakika süren karşı örneklerin incelenmesi ve yanlış alarmların hata senaryolarını gözlemleyerek düzeltilmesi süreci ardından yanlış alarmlar ortadan kaldırılmıştır.

4.3 Erişim Kontrolü Hata Tespiti

Erişim kontrolü hatalarının bulunmasının performansını değerlendirmek için, açık kaynaklı ve php ile geliştirilen Restaurant Script[1] adlı web uygulamasına hata enjekte edilmiştir. Bu uygulamada kullanıcılar web sitesine kayıt olup, kartları yardımıyla pizza siparişi verebilmektedir. Uygulama kaynak dosyalarından, "auth.php" oturum açma alanlarında doldurulan bilgilerin doğruluğunu kontrol etmektedir. Deney kapsamında, bu dosyanın diğer dosyalara eklendiği satırlar yorum haline getirilerek yetki kontrolünün bazı işlemler için yok sayılması sağlanmıştır. (Şekil 8). Bu yolla web uygulamasının 3 hatalı versiyonu yaratılmıştır. Bir versiyonda sadece üyelik işlemleriyle ilgili dosyalara hata enjekte edilmiştir. Diğer versiyonda kart işlemlerinin yetki alanları değiştirilmiştir. Son olarak hem kart hem de üyelik işlemlerine hata enjekte edilmiştir.

```
<?php
    //require_once('auth.php');
    require_once('admin/locale.php');
?>
```

Şekil 8. Erişim kontrolü hatalarına sebep olan kod parçası

Deney için Crawljax3.6 model çıkarımında kullanılmış ve CTL özellikleri tanımlanırken sadece yetki gerektiren sayfalar denetlenmiştir. Tablo 4'te hatalı 3 versiyon için de toplam durum, kenar sayısı ve üretilen karşı örnek sayısı verilmektedir.

Tablo 4. Kusur enjekte edilmiş versiyonlar ve bilgisi

Web Uygulaması	Durum sayısı	Kenar sayısı	Karşı örnek sayısı
Versiyon1	28	49	14
Versiyon2	50	63	4
Versiyon3	49	63	22

Versiyon 1,2 ve 3'te bulunan karşı örnekler yalnızca enjekte edilen hatanın bulunduğu dosyayla ilgilidir. Erişim kontrolü hataları, araç yardımıyla tespit edilebilmiştir.

[1] http://sourceforge.net/projects/restaurantmis/?source=directory

5. Sonuç

Bu çalışmada, formel metot bilgisi olmayan kullanıcılara, model denetleme ile web uygulamalarını doğrulama imkanı vermek için bir yaklaşım ve araç geliştirilmiş, bu kapsamda, modelleme işlemi, zamansal özelliklerin tanımlanması ve üretilen karşı örneklerin anlamlandırılması sağlanmıştır. Ayrıca web crawler'ların ürettiği web model formatları model denetlemede kullanılmış, kullanıcının sıfırdan bir model geliştirmesine gerek kalmamıştır. Geliştirilen aracın kullanılabilirliği formel metot bilgisi olmayan fakat web geliştirmeyle ilgili farklı seviyelerde bilgisi bulunan katılımcılarla değerlendirilmiş ve aracın hata bulma, görselleştirmesinde yararlı olduğu, ama kullanım kolaylığı açısından zayıf kaldığı gözlemlenmiştir. Hata tespit etme performansı ise ulaşılabilirlik ve erişim kontrolü için ayrı değerlendirilmiş, sonuçlara göre, 14 uygulamada 44 ulaşılabilirlik hatası ve 1 web uygulamasına kusur enjekte edilerek tüm erişim kontrolü hataları araç yardımıyla tespit edilmiştir.

Kaynaklar

1. Artzi, S., Kiezun, A., Dolby, J., Tip, F., Dig, D., Paradkar, A., & Ernst, M. D. (2010). Finding bugs in web applications using dynamic test generation and explicit-state model checking. Software Engineering, IEEE Transactions on,36(4), 474-494.
2. Baier C. , Joost-Pieter K. . Principles of model checking. Vol. 26202649. Cambridge: MIT press, 2008.
3. Benedikt, M., Freire, J., & Godefroid, P. (2002). VeriWeb: Automatically testing dynamic web sites. In In Proceedings of 11th International World Wide Web Conference (WW W'2002.
4. Brambilla, M., Cabot, J., & Moreno, N. (2007). Tool support for model checking of web application designs. In Web Engineering (pp. 533-538). Springer Berlin Heidelberg.
5. Castelluccia, D., Mongiello, M., Ruta, M., & Totaro, R. (2006). Waver: A model checking-based tool to verify web application design. Electronic notes in theoretical Computer Science, 157(1), 61-76.
6. Cimatti, A., Clarke, E., Giunchiglia, F., & Roveri, M. (1999, January). NuSMV: A new symbolic model verifier. In Computer Aided Verification (pp. 495-499). Springer Berlin Heidelberg.
7. Common Weakness Enumeration. (2011). Retrieved March 29, 2015, from http://cwe.mitre.org/top25/index.html#CWE-862
8. Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, 319-340.
9. De Alfaro, L. (2001, January). Model Checking the World Wide Web. InComputer Aided Verification (pp. 337-349). Springer Berlin Heidelberg.
10. Demarty, G., Maronnaud, F., Le Breton, G., & Hallé, S. (2013). SiteHopper: Abstracting navigation state machines for the efficient verification of web applications. In Web Services and Formal Methods (pp. 103-117). Springer Berlin Heidelberg.
11. Di Sciascio E, Donini FM, Mongiello M, Piscitelli G. Web applications design and maintenance using symbolic model checking. Proceedings of the European Conference on

Software Maintenance and Reengineering, Benevento, Italy. IEEE Computer Society Press: Silver Spring, MD, 2003; 63–72.
12. Di Sciascio, E., Donini, F. M., Mongiello, M., & Piscitelli, G. (2002, July). AnWeb: a system for automatic support to web application verification. InProceedings of the 14th international conference on Software engineering and knowledge engineering (pp. 609-616). ACM.
13. Di Sciascio, E., Donini, F. M., Mongiello, M., Totaro, R., & Castelluccia, D. (2005). Design verification of web applications using symbolic model checking. In Web Engineering (pp. 69-74). Springer Berlin Heidelberg.
14. Guerra, E., Sanz, D., Díaz, P., & Aedo, I. (2007). A transformation-driven approach to the verification of security policies in web designs. In Web Engineering (pp. 269-284). Springer Berlin Heidelberg.
15. Hallé, S., Ettema, T., Bunch, C., & Bultan, T. (2010, September). Eliminating navigation errors in web applications via model checking and runtime enforcement of navigation state machines. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 235-244). ACM.
16. Han, M., & Hofmeister, C. (2006, July). Modeling and verification of adaptive navigation in web applications. In Proceedings of the 6th international conference on Web engineering (pp. 329-336). ACM.
17. Haydar, M., Petrenko, A., & Sahraoui, H. (2004). Formal verification of web applications modeled by communicating automata. In Formal Techniques for Networked and Distributed Systems–FORTE 2004 (pp. 115-132). Springer Berlin Heidelberg.
18. Homma, K., Izumi, S., Abe, Y., Takahashi, K., & Togashi, A. (2010, July). Using the model checker spin for web application design. In Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on (pp. 137-140). IEEE.
19. Le Breton, G., Maronnaud, F., & Hallé, S. (2013, May). Automated exploration and analysis of ajax web applications with WebMole. In Proceedings of the 22nd international conference on World Wide Web companion (pp. 245-248). International World Wide Web Conferences Steering Committee.
20. Licata, D. R., & Krishnamurthi, S. (2004, September). Verifying interactive Web programs. In Automated Software Engineering, 2004. Proceedings. 19th International Conference on (pp. 164-173). IEEE.
21. Mesbah, A., van Deursen, A., & Lenselink, S. (2012). Crawling Ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web (TWEB), 6(1), 3.
22. Miao, H., & Zeng, H. (2007, July). Model checking-based verification of web application. In Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on (pp. 47-55). IEEE.
23. Top 10 2013-Top 10. (2013). Retrieved March 29, 2015, from https://www.owasp.org/index.php/Top_10_2013-Top_10
24. Torsel, A. M. (2013, March). A Testing tool for Web applications using a domain-specific modelling language and the NuSMV model checker. InSoftware Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on (pp. 383-390). IEEE.

Ek A

Tablo A-1. Değerlendirmede kullanılan web uygulamaları

No	URL	Durum Sayısı	Kenar Sayısı
W1	http://www.seleniumhq.org/	5	7
W2	http://www.brain.ii.metu.edu.tr/eng_dworld.html	19	217
W3	http://www.acarturk.net/	10	51
W4	http://www.uyms2015.yasar.edu.tr/2015/index.html	15	29
W5	http://www.metu.edu.tr/~sbilmis/	6	9
W6	http://www.ceng.metu.edu.tr/~manguoglu/	4	11
W7	http://www.ceng.metu.edu.tr/~polat/	13	74
W8	http://www.kktcbe.org.tr/www/tr/anasayfa.asp	20	79
W9	http://www.metu.edu.tr/~ymetin/	27	163
W10	http://www.everestyayinlari.com/tr/	29	132
W11	http://www.retina.cs.bilkent.edu.tr/	8	25
W12	http://www.cs.bilkent.edu.tr/~ozturk/index.html	16	78
W13	http://www.umram.bilkent.edu.tr/~ergin/	22	326
W14	http://www.atilkurttekin.com/	11	29

Tablo A- 2. Kullanılabilirlik anketi sonuçları

Sorular	**Medyan**	**Ortalama**	**Min.**	**Maks.**
Yararlı bir yazılım aracı olduğunu düşünüyorum	4.5	4.5	4	5
Web sitelerinin hatalarının bulunmasında kolaylık sağlayacak bir araçtır	5	4.75	4	5
Bulunan hataların canlandırılmasının yararlı olacağını düşünüyorum	4.5	4.5	4	5
Kullanımı kolay bir araçtır	3	3.5	2	5
Kullanım açısından anlaşılabilir bir araçtır	3.5	3.25	2	5