Web Güvenlik Tarayıcılarının Evrimi

Ferruh Mavituna, Aralık 2009, WGT E-Dergi 3. Sayı

Web uygulaması güvenliğinin hava civa olduğu yıllarda, 2004'te Istanbul`da (*R'87 - Web Application Security Solutions*) nacizane bir firmanın kurucularından biriydim. Plan basitti sadece web uygulaması penetration testi yapmak ve bu uzmanlığı korumak. Sanırım şu an yazıyı okuyan ve sektörün 5-6 sene önceki halini bilen ya da hayal edebilen herkes vücudunun hangi nadide kısmı ile güleceğini bilememiştir.

O zamanlar bu işe başladık ve beklenmedik bir sürpriz olmadı bir sene kadar sonra battık ama anlatacağım hikaye bu başarısızlık öyküsü değil, hikaye o zamanlar Türkiye'de WebInspect 4.0 ü satmaya çalışmamızla ilgili. Firmayı kurduktan sonra ilk işlerimden biri WebInspect'in Türkiye reseller'lığını almak olmuştu.

WebInspect hatırı sayılır derecede kaliteli ve eski bir tarayıcı o zamanlar özellikle çok ciddi rakipleri de yoktu ve otomatik testlerde büyük bir yardımcıydı. Onun harici başka yazılımlar da bu kervana katıldı ve artık en azından beş adet kaliteli otomatik web uygulaması tarayıcısı sayabilir olduk. AppScan, WebInspect, Acunetix, HailStorm ve bir kaç başka orta - düşük kalite yazılımlar ile SaaS çözümleri.

Lakin bu tarayıcılar 6 sene önceki hallerinden ileriye giderken bir şekilde bir yerlerde tökezlediler. Teknik ilerleme göstermemeninin nedeninden tam emin değilim, bir çok nedeni olabilir ama ilerlemedikleri ya da gerektiği kadar ilerleyemedikleri belliydi. Hatta henüz Netsparker'ın ilk dönemlerinde (o zamanlar ismi Dilemma idi) WhiteHat Security "Attribute based XSS" i tespit edebilen ilk tarayıcı olduğunu duyurunca ben de şöyle bir yazı yazmıştım:

- *If current web application scanners can't find an issue which has been around for 5 years now, aren't they f*** useless?*

Kibar bir Türkçe ile yazarsak:

- *Eğer piyasadaki web uygulaması tarayıcıları 5 senedir bilinen bir açığı dahi bulamıyorlarsa, ne işe yararlar?*

Neler kötüydü ya da Hala Kötü

- Tarayıcılar çetrefilli güvenlik açıklarını görmezden geldi.

- Blind SQL Injection'lar
- Çok genel olmayan XSS açıkları
- Basit blacklisting yapan filtrelemeler
- Hafif komplike SQL Injection' lar (mesela gruplama içeren SQL cümleciklerindeki injectionlar)

- False-Positive hayatımızın bir parçası oldu

Eğer bir siteyi otomatik olarak test ediyorsanız test sırasında program "SQL Injection" ya da "Remote Code Injection" dediğinde artık sevinmiyorsunuz bile, çünkü biliyorsunuz ki %50 false-positive. Manual olarak açığı onaylamadıkça o açığın gerçek olduğunu bilemiyorsunuz, bu da web uygulaması güvenlik tarayıcılarının sonuç üreten araçlardan çok yarım-akıllı fuzzer' lar gibi algılanmasına neden oldu.

False-positive' ler güvenlik uzmanları için kabul edilir hale geldi ve bir çoğu bunları manual kontrol etmeyi işinin doğal bir parçası haline getirdi. Her ne kadar bu güvenlikçiler için büyük bir sorun olmasa da güvenlik konusunda uzman olmayan geliştiriciler bu raporlara güvenmez bir hale geldi.

Piyasadaki hiç bir yazılımın bu konuda bir şey yapmaması, bunu bu yazılımlar arasında komik bir tekele dönüştürdü. False-positive olmadan raporlama yapmak imkansız kabul edildi ve diğerlerine göre daha az false-positive raporlayan yazılımlar iyi kabul edilir oldu.

- Exploitation ve Tarama bir birinden ayrı tutuldu

Sanırım buffer overflow exploitation ile SQL Injection exploitation arasındaki riskleri analiz edemeyen kişiler tarafından insanlar bir şekilde web uygulamalarında exploitation gereksiz ya da tehlikeli olduğu inandırıldı. Bunun dolaylı bir etkisi olarak da bir çok araç bu tip bir özelliği eklemedi.

Daha sonra bu açığı farkedince de entegre çözümler yerine eski yazılımlarının yanına bir yazılım yapıştırdılar ki bu yazılımlar genelde sadece "Biz de bu da var" kıvamında kaldı ve sektörde de pek kabul edilmediler. Çözüm entegre olmayacaksa ve kullanıcı HTTP isteğini kopyala yapıştır, yapıp tespit edilen aynı injection ' ı tekrar konfigüre edecekse bu ek yazılımlar yerine daha stabil ve iyi olan sqlmap gibi open source yazılımları tercih etmeye başladı.

Bu sırada web uygulaması güvenlik tarayıcısı kullanıcıları artık beklentilerini o kadar düşürdü ki kimse "*Bulduğun SQL Injection'ı gene kendi yazılımın ile gelen SQL Injection aracı ile exploit edememek nasıl bir şeydir?*" diye sesli bir şekilde sormadılar bile, çünkü bu kayıp bir savaştı.

Özellikle son senelerini sadece web uygulaması testine harcayan bir penetration tester olarak bu araçların yetersizliğinden dolayı oluşan vakit kaybını hiç hazmedemedim.

Benim kişisel motto'm basitti "**Eğer otomatize edilebiliyorsa, otomatize edilmeli. Nokta.**"

Netsparker

İşte Netsparker'ı bu tuhaf çekişmesiz sektöre bu tip yenilikler getirmek için geliştirdim. Gelişiminin ileriki dönemlerinde takımımız büyüdü, 3 senenin sonunda yapılamaz denilen ya da kimsenin üşenip yapmadığı şeyleri yaptık.

Teknik Konular, False Positive

İlginçtir insanlara ilk olarak Netsparker'ı anlattığımda ve False-Positive raporlamıyor dediğimde ilk üç dakika içerisinde bana kimse inanmıyor, ben de diyalogu şu şekilde devam ettiriyorum:

- *Eğer bir açığı exploit edebiliyorsan o false-positive olabilir mi?*

Bu sorunun cevabı gelmeden zaten koca sarı bir lamba çakıyor ve kendinizi "*Tabii ki ya, ben niye bunu düşünmemiştim*" tadında oynaşan iki göz bebeğine bakarken buluyorsunuz.

Özetle Netsparker bir güvenlik açığının gerçek olduğunu anlamak için onu exploit ediyor, eğer exploit edebilirse açığı onaylıyor eğer edemezse size burada bir sorun olduğunu ama bu sorunu onaylayamadığını raporluyor.

Özetle eğer SQL Injection açığı bulunursa Netsparker SQL Injection üzerinden database versiyonunu alıyor, eğer XSS bulunursa yapılan saldırıyı gerçek bir tarayıcada çalıştırıp enjekte edilen kodun Javascript olarak çalıştırılıp çalıştırılmadığını analiz ediyor.

Teorik olarak bu fikir çok basit gözükse de pratikte bulunan bir SQL Injeciton' ın, XSS' in, LFI' ın otomatik olarak expliot için doğru sintaks' ını tespit edip exploit edilmesi o kadar da basit bir olay değil. Her bir onay sistemi başlı başına teknik bir makale konusu, belki sonradan derginin ilerleyen sayılarından birinde onlara değinme şansı buluruz.

Bütün bu exploit desteği entegre manuel exploitation ve post-exploitation kontrollerini de izin veriyor. Mesela Netsparker bir SQL Injection bulduğunda otomatik olarak veritabanı kullanıcısının haklarını inceliyor ve eğer database kullanıcısı yönetici haklarına sahipse yeni bir güvenlik açığı raporluyor "**Application connected to the Database as an Admin User**". Bu da gerçek bir güvenlikçinin yapacağı bir adet az iş demek ve güvenlikçi olmayan birinin belki de hiç bir zaman farkedemeyeceği bir şeyi farkedebilmesi demek.

Netsparker'ın bu noktadaki vizyonu o kadar güçlü kü bir sonraki versiyonlarda veritabanının versiyonunu alıp o versiyonda raporlanmış ilgili açıkları listeleyecek ve eğer varsa manual olarak privilige-escelation'a izin verecek.

Teknik Konular, Çetrefilli Açıklar

Her açık aynı yazılmamıştır. Çılgın bir dünyada, çılgın geliştiricilerin çılgınca geliştirdiği yazılımlarda çılgınca açıklar görmek gayet doğal, ama bu çılgınlığın harici gayet beklenen güvenlik açıkları da görmek mümkün.

Mesela şu Blind SQL Injection' ı kaç tane tarayıcı yakalar sizce?

- *SELECT id FROM Users WHERE ((approved=1 OR builtin_user= 1) AND active=1) AND (username=' $username' AND pass='$pass')*

Muhtemelen hiç biri, çünkü SQL Cümleciklerinde iki tane grup olması çok anormal bir olay(!).

Ben bunun doğru olmadığını düşündüm ve Netsparker' ı test ederken tek bir güvenlik açığı varyasyonu için 100' den fazla test case' i oluşturduk. Tüm bu testler her versiyondan önce Netsparker ile test ediliyor ve biz de her daim neyi ne kadar bulabildiğimizi biliyoruz.

Eğer false-positive'den kötü bir şey varsa o da false-negative yani bir güvenlik açığını tamamen kaçırmaktır.

Not düşmek ve gerçekçi olmak lazım ki bu testleri çok uçmadan oluşturduk, çünkü acı bir gerçektir, istek sayısı ile kapsama alanınız arasında bazı ayarlamalar yapmanız gerekiyor. Maalesef bir tarama 1 hafta sürerse de pek bir sonuç getirmeyecektir. Bunu da bir gün daha iyi çözeceğiz ama henüz değil...

Teknik Konular, Exploitation

Belirttiğim gibi Netsparker zaten hali hazırda bulduğu açıkları nasıl exploit edeceğini biliyor. Bunun anlamı kullanıcılara en basit şekilde exploit imkanı vermek ve ileri seviye güvenlik bilgisi olmayan kullanıcıların bile basitçe güvenlik açığının etkisini görebilmesini sağlamaktı. Şurada eski versiyonların birinde kaydedilen bir reverse-shell exploitation videosu var.

Eğer bir geliştiriciye SQL Injection'ın ne kadar kötü bir şey olduğunu göstermek istiyorsanız, bu işinizi görecektir.

Mesela Netsparker bir "Arbitrary File Reading" / "LFI" açığı bulduğunda, sizin bu açığı exploit etmeniz için yapmanız gereken tek şey içeriğini görmek istediğiniz dosyanın adını yazmak. Netsparker zaten o dosyanın kaç klasör yukarıda olduğunu ve filtreleri geçmek için ne yapması gerektiğini biliyor.

Dolayısıyla mesela siz ".htaccess" yazdığınızda o şu isteği gönderiyor:

- *../../../../.htaccess%00.php*

Evet siz de bunu yapabilirdiniz ama sizin yapacak daha önemli işleriniz var, mesela mantıksal açıkları bulmak ya da geliştiricilerin enselerine hala "parameterised query" kullanmadıkları için şaplak atmak gibi...

Kapanış

Otomatik web uygulaması tarayıcıları dünyasında daha yapılacak çok iş var ve hepsi de bir gün yapılacak.

Bugün Netsparker resmi olarak satışa duyuruldu ve bundan bir sene sonra eminim ki diğer tüm diğer tarayıcılar da Netsparker' ın açtığı bu yola girmiş ve Netsparker' ı bu yolda takip ediyor olacaklar.

Bundan bir sene sonra False-Positive' ler artık herkese kabul edilemez gelecek, URL Rewrite için kuralları kullanıcıya soran tarayıcılar taşa tutulacak, Javascript analiz edemeyen tarayıcılar cadılarla birlikte samanlıklarda yakılacak, exploit desteği olmayan tarayıcılara şaka

gözü ile bakılacak, bir LFI açığı RFI' a çevrilebiliyor mu diye bakmayan tarayıcılar Taksim meydanında sallandırılacak...

Bu da benim 2010 yılı için yaptığım tahminler olarak Google kayıtlarında tutulsun...