Uygulama Programlama Arayüzü(API) İstismarları

Fortify Taxonomy: Software Security Errors
C# / VB.NET / ASP.NET – API Abuse
(https://www.fortify.com/vulncat/en/vulncat/dotnet/APIA.html)

Semih Gelişli, sgelisli-at-gmail-dot-com, 17 Ağustos 2011

API, istemci ve sunucu arasında bir sözleşme gibidir. API istismarları genel olarak API istemcisinin bu sözleşmenin bittiğini düşünmesinden kaynaklanmaktadır. Örneğin, bir program chroot() fonksiyonunu çağırdıktan sonra chdir() fonksiyonunu çağırma işlemi başarısız olursa, aktif olan klasörün güvenli modda nasıl değiştirileceği belirtilen sözleşmeyi ihlal etmiş olur. Kütüphanelerin, istemci tarafından güvenilir DNS bilgisi dönmesini beklemeleri de başka bir güzel örnek olarak verilebilir. Bu durumda, istemci istenilen API'nın davranışı hakkında belirli varsayımda bulunarak API kuralını ihlal eder. Aynı ihlali API tarafından da görebiliriz. Örneğin, yazılımcı SecureRandom() alt sınıfını kullanıyorsa ve bu alt sınıf değer olarak rastgele olmayan değerler dönüyorsa sözleşme ihlal edilmiş olur.

a) Kod Doğruluğu: GC.Collect() çağırma

Özet

Çöp Toplayıcı(Garbage Collector) için açık istekler olası performans sorunlarını gösterir.

Açıklama

Her .Net geliştiricisinin kariyerinin bir noktasında hata ayıklama sırasında alternatifi olmayan gizemli, aşılmaz ve ayıklaması zor olan durumlarla karşılaşılmıştır ve bu durumlar yüzünden Çöp Toplayıcı suçlanmıştır. Özellikle hata zaman ve durum ile ilgili ise ampirik kanıtlar bu teoriyi desteklemek için bir ipucu olabilir: Bazen GC.Collect() fonksiyonunu çağırmak sorunu giderebilir.

Karşılaşılan çoğu durumda görebiliriz ki GC.Collect() fonksiyonunu çağırmış olmak bir hatadır. GC.Collect() fonksiyonu gereğinden fazla çağırılırsa performans sorunlarına yol açabilir.

Referanslar

[1] StandardsMapping - OWASP Top 10 2004 - (OWASP 2004) *A9 Application Denial of Service*
[2] StandardsMapping - Security Technical Implementation Guide Version 3 - (STIG 3) *APP6080 CAT II*
[3] StandardsMapping - CommonWeaknessEnumeration - (CWE) *CWE ID 730*
[4] RicoMariani *PerformanceTidbits*
[5] StandardsMapping - PaymentCardIndustry Data Security StandardVersion1.1 -
(PCI 1.1) *Requirement 6.5.9*
[6] Scott Holden *Theperils of GC.Collect()*

b) Kod Doğruluğu: Equal() Tanımlı Olmayan Sınıf

Özet

Equals() fonksiyonu tanımlı olmayan objede equals() fonksiyonunu çağırma

Açıklama

Geliştiriciler objeleri karşılaştırırken genellikle objelerin özelliklerini karşılaştırmak isterler. Ancak bir sınıftan Equals() metodunu çağırmak Equals() metodunun açıkça tanımlamış olmaz ve çağırıldığında metot özellikleri System.Object sınıfından miras olarak gelir. Obje üye alanlarının ya da diğer özelliklerinin karşılaştırılmasının yerine Object.Equals() metodu kullanılarak iki objenin aynı olup olmadığını karşılaştırır. Object.Equals() metodunun normal kullanımına izin verilmiş olmasına rağmen genellikle kodun hatalı olduğu izlenimi verir.

Referanslar

[1] StandardsMapping - CommonWeaknessEnumeration - (CWE) *CWE ID 398*

c) Kod Doğruluğu: Dizide Array() kullanmak

Özet

Dizi tanımında Array() fonksiyonu çağırmak

Açıklama

Çoğu durumda dizi tanımında toString() çağırmak dizinin içerdiği bilgiyi string olarak getirir. Ancak array üstünde direkt toString() çağırmak dizinin türünü string olarak getirir.

Örnek:

Kod örneği çıktı olarak System.String[] verecek

```
String[] stringArray = { "element 1", "element 2", "element 3", "element 4" };
System.Diagnostics.Debug.WriteLine(stringArray.ToString());
```

Referanslar

[1] *Class Arrays* Microsoft

[2] StandardsMapping - CommonWeaknessEnumeration - (CWE) *CWE ID 398*

d) Eksik Null Kontrolü Yapmak

Özet

Program bir nullpointer'e referans olarak verilmiş olabilir çünkü fonksiyondan dönen değerin null olup olmadığı kontrolü yapılmamıştır.

Açıklama

Yazılım sistemlerine yapılan her ciddi saldırı yazılımcının düşük seviyede güvenlik varsayımlarından kaynaklanır. Saldırıdan sonra yazılımcının varsayımları zayıf ve çürük olarak görünür ancak saldırıdan önce her yazılımcı kendi varsayımlarını sonuna kadar savunabilir.

İki varsayım vardır ki bunlar kod varsayımları için sabitlenmiştir ve onlar bu "Bu fonksiyonun çağırımında kesinlikle sorun olmaz" ve "Bu fonksiyonun çağırımında sorun olursa önemli bir şey olmaz " düşünceleridir. Eğer yazılımcı bir fonksiyondan dönen değeri kontrol etmez ise bu iki düşüncenin arkasına sığınmış olur.

Örnek:

Aşağıdaki kod; Item() özelliğinden dönen değerin null olup olmadığını Equals() fonksiyonunu çağırmadan önce kontrol etmektedir ve null dereference sorununa sebep olmaktadır.

```
string itemName = request.Item(ITEM_NAME);
if (itemName.Equals(IMPORTANT_ITEM))
{
        ...
}
        ...
```

Bu kodun geleneksel savunması ise:

"Metoda gönderilen değer sürekli var olacak çünkü… Eğer o değer olmazsa program istenen davranışı sağlayamaz ve bu yüzden ortaya çıkabilecek hatayı göz önüne alıp almamak önemli değildir."

Özellikle istisnalar karmaşık ise saldırganlar kullandıkları programlar aracılığı ile beklenmedik açıklıklar bulabilirler.

Referanslar

[1] StandardsMapping - OWASP Top 10 2004 - (OWASP 2004) A9 Application Denial of Service

[2] StandardsMapping - Security Technical Implementation Guide Version 3 - (STIG3) APP3120 CAT II, APP6080 CAT II

[3] StandardsMapping - CommonWeaknessEnumeration - (CWE) CWE ID 253, CWE ID 690

[4] StandardsMapping - PaymentCardIndustry Data Security StandardVersion1.1 - (PCI1.1) Requirement 6.5.9

e) Nesne Modeli İhlali: Sadece Equals() yada GetHashCode() fonksiyonlarından biri tanımlı

Özet

Bu sınıf sadece Equals() ya da GetHashCode() fonksiyonunu geçersiz kılar.

Açıklama

.NET objelerinin eşitlikle ilgili bazı değişmezlere uyması beklenir. Bu değişmezlerden biri ise eşit sınıfların eşit hash kodlara sahip olmasıdır. Diğer bir deyişle eğer

a.equals(b)== true then a.GetGashCode()== b.GetHashCode()

Sınıfın objeleri yığında tutulduğunda bu değişmezliği desteklemek başarısız olursa soruna yol açabilir. Soru içinde olan sınıfın objeleri hash tablosunda anahtar olarak tutulmuş ise veya kütüphaneye veri olarak girilmişse eşit objelerin eşit hash kodlara sahip olması kritik bir duruma gelir.

Örnek:

Aşağıdaki sınıf kodu Equals() fonksiyonunu geçersiz kılmaktadır.

```
public class Halfway()
{
        public override boolean Equals(object obj)
        {
        ...
        }
}
```

Referanslar

[1] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 581

[2] MSDN Library: Equals Method (Object) Microsoft Corporation

[3] MSDN Library: GetHashCode Method (Object) Microsoft Corporation

f) Genel Yanlış Kullanım: Kimlik Doğrulama

Özet

Saldırganlar DNS girdilerini kendi DNS'leri gibi gösterebilirler. DNS isimlerini güvenlik açısından güvenmeyiniz.

Açıklama

Çoğu DNS sunucusu saldırılara karşı savunmasız durumdadır. Bundan dolayı geliştirdiğiniz yazılımın güvensiz bir DNS sunucusu ile çalışabileceğini göz önünde bulundurmalısınız. Eğer saldırganlar DNS sunucusundaki girdileri istedikleri şekilde güncellerler ise sizin ağ trafiğinizi kendi sistemlerine yönlendirebilirler veya kendi IP'lerini sizin sisteminizin bir parçası olarak gösterebilirler. Sistem güvenliğinizi DNS isimlerine göre düzenlemeyiniz.

Örnek:

Aşağıdaki kod DNS lookup fonksiyonunu kullanırken DNS sunucusundan dönen cevabın güvenilir olup olmadığını sorgulamamaktadır. Eğer saldırgan DNS önbelleklerini ele geçirirse sistem tarafından güvenilir bir pozisyona gelmiş olurlar.

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com"))
{
        trusted = true;
}
```

IP adresler DNS isimlerine göre daha güvenilir olmasına rağmen IP adresleri de ele geçirilebilir. Saldırganlar gönderdikleri paketlerin kaynak IP'sini kolaylıkla düzenleyebilirler fakat gönderilen paketlere dönen cevaplar verilen kaynak adreslerine geri gönderilirler. Saldırgan cevap olarak gönderilen paketleri görmek için istemci ve sunucu arasındaki trafiği izinsiz olarak kayıt altına almaya çalışırlar. İşlemi başarı ile bitirmek için kendilerini istemci bilgisayarın olduğu ağa yerleştirirler. Saldırganlar kaynak yönlendirme kullanarak kendilerini ağa yerleştirme işlemine ihtiyaç duymayabilirler ama kaynak yönlendirme Internet'in çoğunda devre dışı duruma getirilmiştir. Özetle IP adres doğrulama işlemi "Kimlik Doğrulama" işleminin bir parçası olarak kullanışlı olabilir ama "Kimlik Doğrulama" işlemi içinde tek faktör olmamalıdır.

Referanslar

[1] Standards Mapping - OWASP Top 10 2004 - (OWASP 2004) A3 Broken Authentication and Session Management

[2] Standards Mapping - OWASP Top 10 2010 - (OWASP 2010) A3 Broken Authentication and Session Management

[3] Standards Mapping - OWASP Top 10 2007 - (OWASP 2007) A7 Broken Authentication and Session Management

[4] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG3) APP3460 CAT I

[5] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 247, CWE ID 292, CWE ID 558, CWE 807
[6] Standards Mapping - FIPS200 - (FISMA) IA
[7] Standards Mapping - Web Application Security Consortium 24 + 2 - (WASC 24 +2) Insufficient Authentication
[8] Standards Mapping - SANS Top 25 2010 - (SANS 2010) Porus Defense - CWE ID 807
[9] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1 - (PCI 1.1) Requirement 6.5.3
[10] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2 - (PCI 1.2) Requirement 6.5.7

g) Denetlenmeyen Dönüş Değeri

Özet

Bir metodun yada fonksiyonun geri dönüş değerinin göz ardı edilmesi programı beklenilmeyen durumlara ve koşullara sokabilir.

Açıklama

Programcılar için Read() ve System.IO sınıflarının bir çok fonksiyonunu yanlış anlamak alışılmadık bir şey değildir. Çoğu hata ve olağan dışı durumlar .NET'te exception hatalarının atılmasında ortaya çıkar.(Bu durum .NET'in C dili gibi diğer yazılım dillerine göre avantajıdır: Exception'lar programcıların sorunun nerede olduğunu anlamalarını kolaylaştırır.) Ama küçük bir miktar verilerin kullanılabilir hale gelmesi durumunda "Stream" ve "Reader" sınıflarını olağandışı düşünmüyoruz. Bu sınıflar küçük miktaradaki verileri arabelleğe alarak geri dönüş yapar ve geri dönüş değeri olarak bayt yada karakter sayısını verir.

Bu davranış Read() ve diğer IO metotlarından dönen değeri anlamak için programcılar için önemlidir ve bekledikleri veriyi almalarını sağlar.

Örnek:

Aşağıdaki kod bir kullanıcı dizisine girerek her kullanıcı için özel bir dosya okumaktadır. Programcı bütün dosyaların her zaman 1kb olacağını varsaymıştır ve Read() metotundan dönen değeri kontrol etmemiştir. Eğer saldırgan daha ufak bir dosya oluşturabilirse 1kb veri doldurabilmek için bir önceki kullanıcının verilerine ulaşacaktır ve verinin saldırgana ait olacağını düşünecektir.

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current())
{
        string userName = (string) i.Current();
        string pFileName = PFILE_ROOT + "/" + userName;
        StreamReader sr = new StreamReader(pFileName);
        sr.Read(byteArray,0,1024); //dosya büyüklüğü her zaman 1kb
        sr.Close();
        processPFile(userName, byteArray);
}
```

Referanslar

[1] Standards Mapping - OWASP Top 10 2007 - (OWASP 2007) A6 Information Leakage and Improper Error Handling

[2] Standards Mapping - OWASP Top 10 2004 - (OWASP 2004) A7 Improper Error Handling

[3] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3) APP3120 CAT II

[4] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 252, CWE ID 754

[5] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2 - (PCI 1.2) Requirement 6.3.1.2, Requirement 6.5.6

[6] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1 - (PCI 1.1) Requirement 6.5.7

[7] Standards Mapping - SANS Top 25 2010 - (SANS 2010) Risky Resource Management - CWE ID 754