Kanonikalizasyon (C14N), Yerel and Evrensel Kodlama

Amaç

Kodlama, uluslararasılaştırma ve evrensel kodlama girdilerinin etkisi altında kaldığında, uygulamanın sağlam olduğundan emin olmak.

Etkilenen Platformlar

Bütün İllgili COBIT Başlıkları

DS11.9 – Veri işleme bütünlüğü

Açıklama

Uygulamalar evrensel kodlama kullanımları için nadiren teste tabi tutulmuştur. Buna rağmen uygulamaların çoğu HTTP Istek Kaçırma'ya neden olan aynı tür sorunlardan dolayı, tehlikeye açıktır. Her browser, her web sunucu, her uygulama güvenlik duvarı veya HTTP saptama elemanı ve diğeri, kullanıcı bölgesi kullanımına farklı (ve genelde kafa karıştırıcı) bir biçimde davranır.

Kanonikalizasyon, verileri bir formdan bir başka forma çeviren sistemlerle ilgilenir. Kanonikal, herhangi bir şeyin en basit ve en standart formu anlamına gelir. Kanonikalizasyon, bir şeyi mevcut halinden en basit haline çeviren prosese verilen addır.

Web uygulamaları, IP adres çevriminden tutun da URL kodlamasına kadar bir çok kanonikalizasyon meselesi ile uğraşır. Güvenlik kararları mükemmellikten uzak olarak kanonikalize edilmiş verilere dayanarak verildiğinde, öncelikle uygulamanın kendisi beklenmedik girdi ile güvenli bir biçimde başa çıkabilecek durumda olmalıdır.

Önemli Not : Kanonikalizasyon saldırılarına karşı güvenlikte olmak, her uygulamanın uluslararasılaştırılması gerektiği anlamına gelmez, ancak evrensel ve kusurlu simgeler giriş yaptığında bütün uygulamalar güvende olmalıdır.

Evrensel Kod

Evrensel kodlama karakterleri çoklu bitler halinde toplamak için kullanılan bir yöntemdir. Girdi verilerine izin verildiği anda, kötü amaçlı kodu saklamak için Evrensel Kod'u kullanarak veri girişi yapılabilir ve bu yolla bir dizi saldırıya izin verilebilir. RFC 2279, metnin kodlanabilmesi için bir çok yöntem önermektedir.

Evrensel kod, dünya çapında hemen bütün yazılı sistemleri içine alan bir Evrensel Karakter Seti (EKS-UCS) oluşturmak amacıyla geliştirilmiştir. Ancak çoklu-sekizli karakterler, şu anda mevcut bulunan uygulama ve protokollerin çoğu ile uyumlu değildir ve bu durum değişken karakterler içeren yeni bir kaç EKS dönüşüm formatının (UTF) geliştirilmesine yol açmıştır. UTF-8, bütün US-ASCII kapsam alanını saklama/düzenleme özelliğine sahiptir. Dosya sistemleriyle, ayrıştırıcılarla-parser- ve US-ASCII değerleriyle çalışan ama diğer değerlere de açık olan yazılımlarla uyumludur.

UFT-8 simgelerinin önemi, web sunucularının ve web uygulamalarının bu formatın girdisinde bir kaç aşamalı bir işleve sahip olması gerçeğinden kaynaklanır. Bu aşamaların sırası bazen uygulamanın güvenliği açısından kritik önem arzeder. Bu aşamalar esas olarak, 'UTF-8' kod çözümü'nü takiben 'URL kod çözümü' ve bunlarla birbirine karışan ve diğer bir işlem aşaması olan çeşitli güvenlik kontrolleridir. Örneğin, güvenlik kontrollerinden biri ".." yi arıyorsa ve bu UTF-8 kodlaması gerçekleşmeden yapılırsa, ".."'yi uzunlamasına ifade edilmiş bir UTF-8 formatına yerleştirmek mümkün olmayacaktır. Güvenlik kontrolleri, nokta işareti için, kanonik olmayan formatın bir kısmını tanımlayabilse dahi, bütün formatlar aynı şekilde 'bilinen' konumunda olmayabilir. ASCII karakteri "."'yı (nokta) ele alalım. Kanonik simgesi bir noktadır (ASCII 2E). Ancak ikinci UTF-8 alanında (2 bayt) bir karakter olarak düşünecek olursak, uzunlamasına ifade simgesini C0 AE olarak görürüz. Ayni şekilde başka uzunlamasına ıfade sımgeleri de vardır : E0 80 AE, F0 80 80 AE, F8 80 80 80 AE and FC 80 80 80 80 AE gibi.

UCS-4 Aralığı	UTF-8 Kodlaması
0x00000000-0x0000007F	0xxxxxxx
0x00000080 - 0x000007FF	110xxxxx 10xxxxxx

0x00000800-0x0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
0x00010000-0x001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0x00200000-0x03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0x04000000-0x7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Bir "."'nın C0 AE sunumunu düşünün. Tıpkı UTF-8 kodlamasının gerektirdiği gibi, en belirgin iki bit için, ikinci sekizli '10' a sahiptir. Şimdi, olası 2 bit kombinasyondan artakalanı sıralayarak bunun için 3 değişken belirlemek mümkündür ('00', '01' ve '11'). Kimi UTF-8 kod çözücüler bu değişkenleri orijinal simgeye eş gibi görecektir (En belirgin 2 biti dikkate almadan en az 6 bit kullanırlar). Sonuç olarak, C0 2E, C0 5E ve C0 FE mevcut 3 değişkendir.

Bu nedenden dolayı da yasal olmayan UTF-8 kod çözümleyici oluşturmak iki türlü mümkündür :

Mevcut bir simge için bir UTF-8 serisi, simgeyi temsil edenden daha uzun olabilir.

- UTF-8 serisi yanlış formatta oktetleri içerebilir (yani yukarıda belirtilen 6 formattan hiçbirine uymaz).

'Karmaşık' şeylere yardımcı olması amacıyla, her sunum değişik yollarla HTTP ile yollanabilir: Kaba : hiç URL kod çözümlemesi uygulamadan. Bu genelde, sorgu veya ana yapı, yolda, ASCII olmayan oktetleri yollarken, HTTP standartlarına uymayarak yapılır. HTTP sunucularının çoğu ASCII olmayan karakterlerle sorunsuz çalışır.

Geçerli URL kod çözümleme : ASCII olmayan her karakter (daha açık soylemek gerekirse, URL kod çözümlemesi gerektiren bütün karakterleri-ASCII olmayan karakterlerden oluşan bir set), URL kod çözümlemelidir. Bu yollama sırasında diyelim şöyle bir sonuç verir : , %C0%AE.

Geçersiz URL kod çözümlemesi : bu, onaltılık rakamların onaltılık olmayan rakamlarla yer değiştirdiği, buna rağmen sonucun özgün sonuçla aynı olarak yorumlandığı yerlerde geçerli bir URL kod çözümlemesi değişkenidir. Örneğin, %C0, ('C'-'A'+10)*16+('0'-'0') = 192 olarak yorumlanır. Aynı algoritmin %M0'a uygulanması ise ('M'-'A'+10)*16+('0'-'0') = 448 gibi bir sonuç verir ki tek bir bite indirgenirse (8 en belirgin bit) sonuç baştaki gibi, 192 olur.Yani, algoritm onaltılı olmayan rakamları kabul ettiğinde (M gibi), %C0'ın %M0 ve %BG gibi değişkenler vermesi mümkündür. Bu

tekniklerin kod çözümlemesiyle direkt ilişkisi olmadığı ve kod çözümlemesi içermeyen saldırılarda kullanılabileceği unutulmamalıdır.

http://www.example.com/cgi-bin/bad.cgi?foo=../../bin/ls%20-al

Örnek saldırının URL kod çözümlemesi:

http://www.example.com/cgi-bin/bad.cgi?foo=..%2F../bin/ls%20-al

Örnek saldırının URL kod çözümlemesi:

http://www.example.com/cgi-bin/bad.cgi?foo=..%c0%af../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%9c../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%pc../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c0%9v../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c0%qf../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%8s../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%1c../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%9c../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%c1%af../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%e0%80%af../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%f0%80%80%af../bin/ls%20-al

http://www.example.com/cgi-bin/bad.cgi?foo=..%f8%80%80%80%af../bin/ls%20-al

Nasıl Korunmalı

Uygun bir kanonikal form seçilmelidir ve bütün kullanıcı girdileri herhangi bir yetkilendirme söz konusu olmadan önce kanonikalize edilmelidir. Güvenlik kontrolleri UTF-8 kod çözümlemesi tamamlandıktan sonra yapılmalıdır. Daha da ötesi, temsil ettiği simge için UTF-8 kod çözümlemesinin geçerli olup olmadığının kontrol edilmesi önerilir. http://www.ietf.org/rfc/rfc2279.txt?number=2279

Girdi formatları

Web uygulamaları genelde içsel olarak, ASCII, ISO 8859-1, veya Unicode (Java programları, UTF-16 ya örnektir) olarak çalışır. Kullanıcılarınız farklı bir alan kullanıyor olabilir ve saldırı sahipleri kendi alanlarını seçebilir.

Tehlikeye açık olup olmadığınızı saptama

Web uygulamasının, içerideki kod sayfasından, alandan veya kültürden gelip gelmediğini saptayın.

Eğer varsayılan karakter seti, alan saptanmadıysa, aşağıdakilerden biri olacaktır :

HTTP tayinleri. İlginç bir ayrıntı : HTTP tayinleri ISO 8859-1 gerektirir ki bu da iki bit karakter setlerinin çoğunda verilerin kaybolmasına neden olur. Uygulamanızı, destekleyen bir browser ile test ederek tam olarak kodlanmış çift bit karakterlerin güvenlik içinde aktarılıp aktarılmadığını saptamanız gerekir.

HTTP alımları. Daha önceden iade edilen sayfa ve beher browser uygulamalarına bağlıdır ama URL kodlaması iki bir karakter setleri için tanımlanmamıştır. IE, bütün başvurularını sunucuda uygun biçimde kanonikalize edilen UTF-8 cinsinden yapması için zorlanabilir.

.NET: evrensel kod (küçük endian)

- JSP uygulamaları, örneğin Tomcat: UTF-8 – web.xml'de "java kodlaması" bölümüne bakınız.
- Java: evrensel kodlaması (UTF-16, büyük endian, ***veya*** JVM çalıştırılırkenki İS'ne bağlı)
- PHP: php.ini'deki ayarlar, ISO 8859-1.
  Önemli Not**: PHP fonksiyonlarının çoğu karakter setiyle ilgili (geçersiz) varsayımlarda bulunur make (invalid) ve başka bir karakter setine geçildiğinde doğru dürüst çalışmayabilir. Yeni karakter setiyle uygulamanızı mutlaka test edin !**

Kendinizi koruma

- Uygulamanızın gerekliliklerini belirleyin ve dil alanı ve karakter setini birbirine uygun olacak biçimde ayarlayın.

Alan onayı

Web sunucu mutlaka bir alan ve tercihen "en_US", "fr_FR", "zh_CN" gibi bir ülke kodu onaylamalıdır.

Tehlikeye açık olup olmadığınızı saptama

Web sunucunuz için ya HTTP başlık bulucu ya da telnet kullanın:

```
HEAD / HTTP1.0
```

Sonuç şöyle olmalı:

```
HTTP/1.1 200 OK
Date: Sun, 24 Jul 2005 08:13:17 GMT
Server: Apache/1.3.29
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

Kendinizi koruma

Aşağıdaki klavuz bilgilerini gözden geçirin ve uygulayın:

http://www.w3.org/International/technique-index

En azından doğru çıktı alanını ve karakter setini seçin.

çift (veya n-) kodlama

web uygulamalarının çoğu girdinin dopru evrensel kodlara de-encode edilip edilmediğini belirlemek içinsadece bir kez kontrol yapar. Ancak, saldırgan saldırı cümleciğini iki kez kodlayabilir..

tehlikeye açık olup olmadığınızı saptama

- XSS Cheat Sheet çifte kodlayıcı elemanın kullanarak XXS cümleciğini iki kez kodlayın.
  http://ha.ckers.org/xss.html
- sonuçta meydana gelen injection başarılı bir XSS çıktısı ise, uygulamanız tehlikeye açık demektir.
- Bu saldırı aşağıdaki elemanlara karşı da kullanılablir:
  1. dosya adları
  2. rapor türleri ve dil seçiciler gibi belirgin olmayan kalemler
  3. tema isimleri

Kendinizi koruma

- uygulamanız için doğru alan ve karakter seti saptayın.
- Evrensel kodlamanın farklı browser, sunucu ve uygulama bileşimlerince yanlış biçimde kullanılmasını önlemek amacıyla HTML elemanları, URL kodlaması ve diğerlerini kullanın.
- Kodunuzu ve bütün çözümü yaygın biçimde test edin.

HTTP istek kaçakçılığı

HTTP istek kaçakçılığı (HTK)referanslar kısmında Klein, Linhart, Heled, ve Orrin tarafından ayrıntılı biçimde incelenerek resmi rapor halinde yazılmış bir meseledir. HTTP istek kaçakçılığının temelinde, bir çok çözümün, bir web uygulaması sağlamak için pek çok bileşen kullanması vardır. Güvenlik duvar, web uyglaması duvarı, yükleme dengeleyicileri, SSL hızlandırıcılar, ters proksiler ve web sunucular arasındaki fark, özellikle usta bir saldırının kullanıcı tarafındaki bütün kontrolleri aşmasına ve doğrudan web sunucuya saldırmasına izin vermesidir.

Tanımladıkları saldırı türleri:

- Web ön bellek zehirlenmesi
- Güvenlik duvarı/IDS/IPS ele geçirme
- ileri ve geri HTK teknikleri
- istek kaçırma
- istek güven kaçırma

resmi rapor yazıldığından bu yana, gerçek hayatta HİK'ya örnek olacak bir kaç saldırı saptanmıştır.

tehlikeye açık olup olmadığınızı saptama

- resmi raporu gözden geçirin.
- Tehlikeye açık bileşenler için altyapınızı gözden geçirin.

Kendinizi koruma

- Gelen HTTP isteklerini yorumlayabilecek olan toplam bileşen sayısını en aza indirin
- Yama programları ile altyapınızı güncel tutun.

Ileri okuma

- Evrensel kodlama kullanarak IDS Ele geçirme
  http://online.securityfocus.com/print/infocus/1232
- W3C uluslararasılaştırma ana sayfa
  http://www.w3.org/International/
- HTTP istek kaçakçılığı
  http://www.watchfire.com/resources/HTTP-Request-Smuggling.pdf
- XSS Cheat Sheet
  http://ha.ckers.org/xss.html

Dökümanın Aslı : OWASP GUIDE 2.0.1

Çeviri : Billur C. Yılmazyiğit - info@biledge.com