1.1.1 Yetersiz Yığın Denetimi (Heap Inspection) (CWE-244)

Açıklık Önem Derecesi: Düşük

Açıklığın Etkisi: Hassas Bilgilere Yetkisiz Erişim, Gizlilik İhlali

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar tarafından bellekte şifrelenmeden depolanan tüm değişkenler/nesneler yetkisiz kullanıcılarca yetkili erişim sağlanarak elde edilebilir. Örneğin "yetkili" bir saldırgan çalışan process'e (işleme) bir debugger (hata ayıklayıcı) bağlayabilir, ya da swapfile dosyası veya crash dump dosyasından process'in (işlemin) bellek dökümünü elde edebilir. Saldırgan bu yollarla bellekteki kullanıcı parolalarını bulabilir ve bu bilgileri kolaylıkla bir kullanıcı profiline girmek için sistemde kullanabilir.

String değişkenleri / nesneleri immutable'dırlar (değişmezdirler). Diğer bir deyişle; bir string değişkeni/nesnesi tanımlandı mı bu değişkenin/nesnenin değeri değiştirilemez veya silinemezdir. Bu nedenle bu string'ler garbage collector (çöp toplayıcı) tarafından silinene kadar belirsiz bir süre boyunca bellekte - muhtemel birden fazla konumda - yer alırlar. Bu string'ler ile tanımlanan hassas veriler (örn; parolalar, şifreleme anahtarları, api anahtarları, …) yaşam döngüleri boyunca kontrolsüz bir şekilde bellekte plaintext (açık metin/şifresiz metin) olarak sergilenir kalırlar. Bu ise belleğe erişim hakkı kazanmış saldırganların bellekteki şifresiz hassas verileri görüntüleyerek bu hassas verileri ele geçirmelerine sebep olabilir. Uygulamalar hassas verileri bellekte şifresiz halde konumlandırdıklarında heap inspection (yığın denetimi) açıklığı vardır denir.

Heap Inspection açıklığını somutlaştırmak adına çeşitli dillerde heap inspection zafiyetli kod blokları ve güvenli halleri verilmiştir:

C++ Güvensiz Kod Bloğu:

```
char* password = (char*) malloc(256);                    // GÜVENSİZ
int i=0;
char ch;
ssize_t k;
while(k = read(STDIN_FILENO, &ch, 1) > 0)
{
    if ((ch == '\n') || (i >= 255))
    {
        password[i]='\0';
        break;
    }
    else {
        password[i++]=ch;
    }
}

// ..
// Verify password
// ..

free(password);
```

C++ Güvenli Kod Bloğu:

```
volatile char* password = (char*) malloc(256);          // GÜVENLİ
int i=0;
char ch;
ssize_t k;
while(k = read(STDIN_FILENO, &ch, 1) > 0)
{
  if ((ch == '\n') || (i >= 255))
  {
    password[i]='\0';
    break;
  }
  else {
    password[i++]=ch;
    fflush(stdin); // Zero-out input stream
  }
}

i=0; // Zero-out password length
// ..
// Verify password
// ..

while (i <= 255) {
  password[i++] = 0; // Zero-out password, clearing it from the heap
}
free((void*)password);
```

C++ örneğinde görüldüğü gibi hassas bir veri (bu örnekte parola verisi) immutable (değişmez) bir nesne olarak tanımlanmıştır. Hassas veriler mutable (değişebilir) tanımlanmalıdır. Bunun için C++ uygulamalarda hassas verileri tutacak nesneler volatile şeklinde tanımlanabilir. Bu şekilde açıklık kapanacaktır.

C# Güvensiz Kod Bloğu:

```
class Heap_Inspection
{
    private static string password                // GÜVENSİZ
    {
        get;
        set;
    }

    public void setPassword(string newPassword)
    {
        password = newPassword;
    }

    public string getPassword()
    {
        return password;
    }
}
```

C# Güvenli Kod Bloğu:

```
class Heap_Inspection_Fixed
{
    private static SecureString password
    {
        get;
        set;
    }

    public void setPassword(SecureString newPassword)
    {
        password = newPassword;
    }

    public SecureString getPassword()
    {
        return password;
    }
}
```

C# örneğinde görüldüğü gibi hassas bir veri (bu örnekte parola verisi) immutable (değişmez) bir nesne olarak tanımlanmıştır. Hassas veriler mutable (değişebilir) tanımlanmalıdır. Bunun için C# uygulamalarda hassas verileri tutacak nesneler SecureString şeklinde tanımlanabilir. Bu şekilde açıklık kapanacaktır.

Java Güvensiz Kod Bloğu:

```
class Heap_Inspection
{
    private String password;                              // GÜVENSİZ

    public void setPassword(String password)
    {
        this.password = password;
    }
}
```

Java Güvenli Kod Bloğu:

```
class Heap_Inspection_Fixed
{
    private SealedObject password;                        // GÜVENLİ

    public void setPassword(Character[] input)
    {
        Key key = getKeyFromConfiguration();
        Cipher c = Cipher.getInstance(CIPHER_NAME);
        c.init(Cipher.ENCRYPT_MODE, key);
        List<Character> characterList = Arrays.asList(input);
        password = new SealedObject((Serializable) characterList, c);

        // input'a sıfır yazdırılır. Bu işlem
        // ayrıca characterList nesnesinin de-
        // ğerleri üzerine de reference ile
        // sıfır yazdırmayı sağlar.
        Arrays.fill(input, '\0');
    }
}
```

Java örneğinde görüldüğü gibi hassas bir veri (bu örnekte parola verisi) immutable (değişmez) bir nesne olarak tanımlanmıştır. Hassas veriler mutable (değişebilir) tanımlanmalıdır. Bunun için Java uygulamalarda hassas verileri tutacak nesneler SealedObject şeklinde tanımlanabilir. Bu şekilde açıklık kapanacaktır.

Örneğin güncel hayattan bu açıklığın minimize edilmesine bir masaüstü uygulaması olan Kaspersky Password Manager örnek olarak verilebilir. Bu uygulama bir parola saklama uygulamasıdır. Ne zaman kullanıcı bir parolasına uygulamada bakmaya çalıştığında parola sabit diskten belleğe aktarılır, parolayı kopyaladıktan yaklaşık 1 dakika içerisinde ise parola

panodan (bellekten) silindi mesajı ufak balon halinde ekrana yansır. Yani parola bellekte bilgisayar kapanana kadar kalmaz.

Kurum uygulamada heap inspection açıklığı tespit edilmiştir:

::::::: BULGU :::::::

Açıklığın Önlemi:

Uygulanması önerilen güvenlik önlemleri genel manada şu şekildedir:

- Kısa bir süreliğine dahi olsa plaintext (açık metin / şifresiz) bir şekilde bellekte hassas veri (örn; parolalar, şifreleme anahtarları, api anahtarları, v.b.) depolamayın.
- Belleğe şifrelenmiş veri depolayan özelleştirilmiş sınıfları kullanmayı tercih edin ki bellekten bulup çıkarılamasın.
- Hassas verileri ham biçimde kullanılması gerektiğinde bellekten okunurluğunu azaltmak için geçici olarak mutable (değişebilir) veri türünde depolayın (örn; byte dizileri halinde). Ardından hemen akabinde bu verilerin bellekteyken sergilenme süresini azaltmak için bellekteki konumlarına sıfır yazdırın.
- Yukarıdaki önlemler uygulansa bile bellek dökümlerinin güvenilmez taraflarla (untrusted parties'le) değiş tokuş edilmediğinden emin olunmalıdır. Çünkü şifreli değişkenlere / nesnelere tersine mühendislik yapmak veya bellekten hassas veri byte'larını çıkarmak ve tekrar derlemek halen mümkün olabilir.

Heap Inspection önlemi olarak immutable (değişmez) yerine temizlenen mutable (değişebilir) bir veri türü kullanılsa bile veya bir şifre çözme anahtarı getirip bellekten hassas veriyi şifresini çözerek getirsek bile heap inspection saldırıları neticesinde bellekten veriyi getirtmek halen mümkündür, fakat bahsedildiği şekilde bir hassas veri katmanlama önlemi uygulanması saldırganların yapması icap eden eforların miktarını oldukça arttıracaktır. Bellekten hassas verilerin getirtilişini, bellekteki hassas verilerin sergilenişini ve miktarını azaltmak için güvenlik çıtasını yukarıda tutarak saldırganların değerli veri elde etmesi yolundaki başarısı önemli ölçüde düşürülebilir.

Heap Inspection açıklığından kaçınırken şunu vurgulamak önemlidir: Bir uygulamanın bellek dökümüne veya belleğine herhangi bir erişim verildiğinde saldırganlara hassas bir veri ifşa etmek her daim mümkündür. Bahsedilen güvenlik önlemleri belleğe okuma erişimi başarılı şekilde sağlanmış durumlarda hassas verilerin korunması noktasında defense-in-depth (derinlikli defans) prensibinin bir parçası olarak yer alır. Bu öneriler bellekteki hassas verilerin sergilenmesi ve yaşam ömründe önemli bir azaltma sağlar. Ancak şu bir gerçektir ki yeterli zaman, çaba ve belleğe sınırsız erişim verildiğinde uygulama tarafından kullanılan hassas

verileri korumada yalnızca bir yere kadar gidilebilir. Heap Inspection açıklığının üstesinden gelmenin tek yolu hassas verilerin bellekte sergilenmesini azaltmak ve minimize etmek ve mümkün olan belleğin her yerinde bu hassas verileri karartmak, yani saldırganların harcaması gereken eforu arttırmaktır.

Referanslar:

1. https://cwe.mitre.org/data/definitions/244.html