İlk olarak kodumun kısımlarına ait konuların açıklaması

sonra kodun açıklaması

KONU: machine learning malware detection

Açıklama: makine öğrenimi ile oluşturulmuş bir dedektörün malware ve zararı olmayanlardan string ve iat tablolarından elde etmiş olduğu bilgiler ile analiz yapıp test edilen dosyanın zararlı olup olmadığına karar veriyor.

Githuba yükle: Yazıların çeviri hali

özet hali

resimler

kodlar

Mediuma özetlerden bir şeyler yükle

Raporu hocaya yolla:

Öncellikle projem boyunca malwaredatascience adlı kitaptan yararlanıp onların sağlamış olduğu işletim sistemindeki kaynaklardan yararlandım. Bu kitapta olan machine learning malware detector kodundaki stringe göre ayıklama yapan sisteme öğretmiş oldukları iat tablolarını da çıkarıp analize katmayı ekleyip ilerlettim.

Şimdi konu konu size anlatıp en son kodların üzerinden geçelim.

Statik analiz:Bir programın  dosyasının disassembled kodunun , grafik görüntülerinin , yazdırlabilir dizelerin strings ve diğer disk üzerindeki kaynakların analiz edilmesiyle gerçekleşir. Programı çalıştırmadan reverse engineering tersine mühendisliği ifade eder.  Statik analizin eksikleri olsa da bize bir zararlıyı anlamızda yardımcı bilgiler verir.Statik analiz teknikleri çoğu windows programı tarafından kullanılan portable executable(PE) dosya formatından bilgi çıkarma, içe aktarma analizi describe techniques import analysis, grafik görüntü analizi , graphical image analysis ve dizi analizi strings analysis tekniklerinden bahsedebiliriz.

Benim projemde analiz yapmış olduğum  içe aktarma analizi describe techniques import analysis ve stringsleri biraz da açalım.

\*The Microsoft Windows Portable Executable Format: Mikrosoft Windows Taşınabilir Yürütülebilir Biçim:

Statik malware analizi için , .exe,.dllve .sys dosyaları gibi modern Windows program dosyalarının yapısını tanımlayan ve verileri sakalam yöntemlerini tanımlayan Windows PE biçimine bakalım.PE dosyaları x86 talimatlarını , görüntüler ve metin gibi verileri ve bir programın çalışması için ihtiyaç duyduğu meta verileri içerir.

Pe dosya biçimi windows!a bir programın belleğe nasıl yüklenmesi gerektiğini ve nereye yükleneceğini açıklar. Ayrıca windows'un program kodunun neresinde bir programın yürütülmesini , başlatmaısnı ve hangi dinamik olarak bağlantılı kod kitaplıklarının belleğe yüklenmesi gerektiğini de söyler. ÇAlışan bir programın yürütülmesi sırasında kullanabileceği kaynaklarıü gui iletişim kutularını veya konsol çıktısında bulunanlar gibi karakter dizilerinin yanı sıra görüntüleri veya videoları içerebilir.

Windows kodun güvenilir bir jaynaktan geldiğinden emin olmak için dijital kod imzaları gibi güvenlik verilerini kullanır.

kaynak:https://ring0.info/posts/pe-dosya-formatina-dalis

8-> .reloc section (memory translations )

7-> .rsrc section (strings, images, ..)

6->  .idata section (imported libraries)

5-> .text section (program code)

4-> Section headers Bölüm başlıkları

3-> Optional headers İsteğe bağlı başlıklar

2-> PE header

1-> DOS header

8'e doğru offaset değeri artıyor.

PE biçimi , işletim sistemine programın belleğe nasıl yükleneceğini anlatan bir dizi başlık içerir.  Ayrıca, gerçek program verilerini içeren bir dizi bölüm içerir. Windows bölümleri belleğe, bellek uzaklıkları diskte göründükleri yere karşılık gelecek şekilde yükler.1980'lerden kalma Microsoft DOS işletim sisteminin bir kalıntısı olan ve yalnızca uyumluluk nedenleriyle sunulan DOS başlığıyla ilgili tartışmayı atlayacağız.

\*The PE Header:

Bir programın binary kodu, görüntüleri, sıkıştırılmış  veriler ve diğer program öznitelikleri gibi genel özniteliklerini tanımlayan PE başlığıdır. Ayrıca bize bir programın 32 bitlik mi 64 bitlik sistemler için mi tasarlandığını söyler. Anlayacağınız PE bize temel ama yararlı bilgiler verir. Örneğin üstbilgi header, malwarein ne zaman dosyayı derlediğini içeren zaman damgası alanı içerir. Bunu tabi sahte bir değerle de değiştirebilir yazan kişi.🙂

\*The Optional Header:

Günümüzün PE çalıştırılabilir programlarında aslında her yerde bulunur. Programın yüklendikten sonra çalıştırdığı ilk talimatı ifade eden PE dosyasındaki programın giriş noktasının konumunu tanımlar.ayrıca PE dosyasını windows alt sistemini , program hedefleirni (Windows GUI veya Windows komut satırı gibi) ve programla ilgili diğer üst düzey ayrıntıları yüklerken Windows'un vellğe yüklediği verileirn boyutunu da tanımlar. Bu başlıktaki bilgiler tersine mühendislik için programın başlangıç noktasını verdiğinden dolayı değerlidir.

\*Section Header:

Bir PE dosyası içinde bulunan veri bölümlerini açıkalr. Programın belleğe nasıl yüklenmesi gerektiği ile ilgili talimatları içeren veri yığınıdır. Bu kısım ayrıca windowsa bölümlere hangi izinleri vermesi gerektiğini örneğin program yürütülürken program tarafından okunabilir ,yazılabilir veya çalıştırılabilir olması gerektiğini söyler.

\*The .text Section:

her pe programı bölüm başlığında yürütülebilir olarak işaretlenmiş en az bir x86 kodu bölümü içerir. bu bölümler hemen hemen her zaman .text olarak adlandırılır.

\*the .idata Section:

içe aktarma import olarak adlandırılan idata bölümü dinamik olarak bağlantılı kitaplıkları ve işlevlerini listeleyen içe aktarma tablosu (IAT) içerir. IAT , analiz için bir PE ikili programına ilk bakılırken incelenecek en önemli PE yapıları arasındadır. Çünkü bir programın yaptığı kitaplık çağrılarını ortaya çıkarır.

\*The Data Sections:

PE dosyasındaki data bölümleri , fare ilecinin görüntüleri , ses ve bir program tarafından kullanılan diğer ortamlar gibi öğeleri depolayan .rsrc , .data ve .rdata gibi bölümleri içerebilir. .rsrc kısmı bir programın texti string olarak işlemek için kullandığı yazdırılabilir strings içerir. Buradan yazdırılabilir karakter dizelerini ,grafik görüntüleri ve diğer varlıkları inceleyerek dosyanın işlevselliği hakkında önemli iğuçları elde edilebilir. malwarelerin kaynaklar bölümlerinden graik görüntülerini icotuils ile çıkarırız.  .reloc kısmı : bir PE ikili kodu konumundan bağımsız değildir yani amaçlanan bellek konumundan yeni bir bellek konumuna taşınırsa doğru şekilde çalışmayacaktır. reloc kosun kırılmadan taşınmasına izin vererek bunun üstesinden gelir. windows işletim sistemine kod taşınmışsa kodun hala düzgün çalışması için PE doasyaısnın kodundaki bellek adreslerini çevirmesini söyler.

Bir programın bölümlerini ayrıştırmak için pefile kullanmaya ek olarak, bir ikilinin yükleyeceği DLL'leri ve bu DLL'ler içinde isteyeceği işlev çağrılarını listelemek için de kullanabiliriz. Bunu, bir PE dosyasının IAT'sini boşaltarak yapabiliriz. Şimdi ircbot.exe nin IAT'yi dökmek için pefile'ın nasıl kullanıcağına bakalım:

Bu, kötü amaçlı yazılım hakkında oldukça temel bilgiler olsa da, kötü amaçlı yazılımın davranışını daha ayrıntılı olarak anlamak için bir başlangıçtır.

\*Şimdi malware hakkında bilgi çıkarma işlemlerinden bir diğeri olan string inceleme işlemlerine bakalım.Dizeler Strings , bir program ikili dosyası içindeki yazdırılabilir karakter dizileridir.Bu dizeler genellikle web sayfalarını ve dosyaları, IP adreslerini ve kötü amaçlı yazılımın hangi adreslere bağlandığını size söyleyen ana bilgisayar adlarını ve benzerlerini indiren HTTP ve FTP komutları gibi şeyler içerir.Bazen stringlerde ki kullanılan  dil bile bize ülke hakkında bilgi verir. . Dizeler ayrıca bir ikili dosya hakkında daha fazla teknik bilgi ortaya çıkarabilir. Örneğin, onu oluşturmak için kullanılan derleyici, ikilinin yazıldığı programlama dili, gömülü komut dosyaları veya HTML vb. Hakkında bilgi bulabilirsiniz. . Kötü amaçlı yazılım yazarları tüm bu izleri gizleyebilse, şifreleyebilse ve sıkıştırabilse de, gelişmiş kötü amaçlı yazılım yazarları bile çoğu zaman en azından bazı izleri açığa çıkararak kötü amaçlı yazılımları analiz ederken dize dökümlerini incelemeyi özellikle önemli hale getirir.

strings komutu minimum 4 bayt uzunluğa sahip tüm yazdırılabilir dizeleri bulur, ancak command page sayfasında listelendiği gibi farklı bir minimum uzunluk ayarlayabilir ve çeşitli diğer parametreleri değiştirebilirsiniz.

Bu satırlar, ircbot.exe'nin bir saldırgan tarafından belirtilen dosyaları hedef makineye indirmeyi deneyeceğini gösterir.  Başka birini analiz etmeyi deneyelim. Liste 1-8'de gösterilen dize dökümü, ircbot.exe'nin saldırgandan gelen bağlantılar için hedef makinede dinleyen bir web sunucusu görevi görebileceğini belirtir.

Teknikleri anlattığımıza göre biz bunları kendi malware dedektörümüze nasıl uygulamamız gerekiyor ona bakalım.

malware detection sistemler için kullanılan scifit learn açık kaynak makine öğrenimi paketinden yararlandık.İlk olarak scikit-learn kullanmadan önce bilmemiz gerekn terminoloji ve kavramlara bakalım.Birde karar ağacı yapılarının açıklamasını yapalım.Biz burada rastgele orman dedektörü ilemakine öğrenmimi sistemlerini değerlendirmek için scikit-learn yapsını kullanıyoruz.

açık kaynak kitaplık olan sckit-learn kısaca sklearn , hem güçlü  hem de kullanımı kolay olduğu için makine öğrenimi topluluğunda popüler hale geldi.Birçok makine öğrenimi çerçevesi gibi sklearn de vektör biçiminde eğitim verileri gerektirir. Vektörler, dizideki her diznin eğitim örnek yazılım ikili programının tek bir özelliğine karşılık geldiği sayı dizileridir.Örneğin makine öğrenimi algılayıcımızın kullandığı yazılım ikili dosyalarının iki özelliği olan sıkıştırılmışsa ve şifrelenmiş veriler içeriyorsa eğitim örneği ikili dosyası için özellik vektörümüz [0,1] olabilir. Burada vektördeki ilk dizin ikilinin sıkıştırılıp sıkıştırılmadığını temsil eder, sıfır "hayır"ı gösterir ve ikinci dizin ikili dosyanın şifrelenmiş veri içerip içermediğini gösterir ve bir "evet"i gösterir.Her bir dizinin hangi özelliğe eşlendiğini hatırlamanız gerektiğinden, vektörlerle çalışmak garip olabilir.

Neyse ki sklearn, diğer veri temsillerini vektör biçimine çeviren yardımcı kod sağlar. Örneğin, eğitim verilerinizin sözlük temsillerini (örneğin, {"sıkıştırılmış": 1, "şifrelenmiş veriler içerir": 0}) sklearn'ın üzerinde çalıştığı vektör gösterimine dönüştürmek için sklearn'ın DictVectorizer sınıfını kullanabilirsiniz, örneğin [0 , 1]. Daha sonra, vektörün indisleri ile orijinal özellik adları arasındaki eşlemeyi kurtarmak için DictVectorizer'ı kullanabilirsiniz.

Sklearn tabanlı bir detektörü eğitmek için, sklearn için iki ayrı nesneyi aktarmanız gerekir: özellik vektörleri feature vectors(daha önce açıklandığı gibi) ve bir etiket vektörü a label vector.Bir etiket vektörü, eğitim örneği başına bir sayı içerir ve bizim durumumuzda, örneğin kötü amaçlı yazılım veya zararlı yazılım olup olmadığına karşılık gelir.Örneğin, sklearn için üç eğitim örneğini ve ardından etiket vektörünü [0,1,0] geçirirsek, sklearn'a ilk örneğin iyi huylu yazılım, ikinci örneğin kötü amaçlı yazılım ve üçüncüsünün iyi huylu yazılım olduğunu söyleriz.Geleneksel olarak, makine öğrenimi mühendisleri eğitim verilerini temsil etmek için bir büyük harf X değişkeni ve etiketleri temsil etmek için küçük bir y değişkeni kullanır..Durumdaki fark, matrisleri temsil eden değişkenlerin büyük harfle yazılması (vektör dizileri olarak düşünebiliriz) ve ayrı vektörleri temsil eden küçük harfli değişkenlerin matematiğindeki geleneği yansıtır.

Makine öğrenimi tabanlı dedektörlere "dedektörler detectors" demek yerine, sklearn bunları "sınıflandırcı classifiers" olarak adlandırıyor. Bu bağlamda, sınıflandırıcı terimi basitçe, nesneleri iki veya daha fazla kategoriye ayıran bir makine öğrenimi sistemi anlamına gelir.Bu nedenle, bir algılayıcı (bu kitapta kullandığım terim), nesneleri kötü amaçlı yazılım ve zararlı yazılım malware and benignware gibi iki kategoriye ayıran özel bir sınıflandırıcı türüdür. Ayrıca, eğitim terimini kullanmak yerine sklearn dokümantasyonu ve API genellikle fit terimini kullanır.Son olarak, sınıflandırıcılar bağlamında algılama terimini kullanmak yerine, sklearn tahmin terimini kullanır.

.dot olanları resimle gösterebilirim

-String Features:Dize Özellikleri:

Dizeleri çıkarıp en az karakteri belirleyip eleyip fazla dizeleri alıyoruz. String özelliklerinin sklearnın anlayabileceği bir biçime dönüştürmek için onları bir Python sözlüğüne koymamız lazım.  Bunu, gerçek dizeleri sözlük anahtarları olarak kullanarak ve ardından söz konusu ikilinin bu dizeyi içerdiğini belirtmek için değerlerini 1 olarak ayarlayarak yaparız. Örneğin, önceki örnek ikili, {"PE yürütülebilir": 1, "Kötü amaçlı yük": 1} özellik vektörünü alacaktır. Elbette, çoğu yazılım ikili dosyasında yalnızca iki değil, yüzlerce yazdırılabilir dize bulunur ve bu dizeler, bir programın ne yaptığı hakkında zengin bilgiler içerebilir.

-Import Address Table (IAT) Features:

Bölüm 1'de öğrendiğiniz İçe Aktarma Adres Tablosu (IAT) da makine öğrenimi özelliklerinin önemli bir kaynağıdır. IAT, bir yazılım ikili dosyasının harici DLL dosyalarından içe aktardığı işlevlerin ve kitaplıkların bir listesini içerir. Bu nedenle, IAT, önceki bölümde açıklanan PE başlık özelliklerini tamamlamak için kullanabileceğiniz program davranışı hakkında önemli bilgiler içerir.

  IAT'yi makine öğrenimi özelliklerinin kaynağı olarak kullanmak için, her dosyayı bir özellikler sözlüğü olarak temsil etmeniz gerekir; burada içe aktarılan kitaplığın ve işlevin adı anahtar, anahtar ise dosyanın söz konusu belirli içe aktarmayı içerir (örneğin, "KERNEL32.DLL: LoadLibraryA" anahtarı, burada KERNEL32.DLL DLL ve LoadLibraryA işlev çağrısıdır). Bir örnek için IAT özelliklerinin bu şekilde hesaplanmasından elde edilen özellik sözlüğü, bir ikili dosyada gözlemlenen herhangi bir tuşa 1 atayacağımız {KERNEL32.DLL: LoadLibraryA: 1, ...} gibi görünür.Tek başına yeterli değil Bununla birlikte, IAT özelliklerini PE başlık özellikleri ve dizi özellikleri gibi diğer özelliklerle birlikte kullandığınızda, sistem doğruluğunu artırabilirler.

tüm özellikleri kullanınca bellek ve zamanda sıkıntı çıkabilir.

->Using the Hashing Trick to Compress Features:Özellikleri Sıkıştırmak için Karma İşlemi Hilesini Kullanma:

Çok fazla özelliğe sahip olma sorununu aşmak için, özellik karması olarak da bilinen hashing hilesi adı verilen popüler ve basit bir çözümü kullanabilirsiniz.elindeki veriyi gücünün yetdiği veri boyutuna hashing ile sıkıştırıyorsun. Hafif bozulma olur ama olsun.

-Implementing the Hashing Trick:Hashing Hilesini Uygulama

nasıl çalıştığını göstermek için bu kodu gösteriyorum; daha sonra, sklearn’ın bu işlevin uygulamasını kullanacağız. Örnek kodumuz bir işlev bildirimiyle başlar:

def apply\_hashing\_trick(feature\_dict, vector\_size=2000):

İki parametre alır. orijinal özellik sözlüğü ve hash hilesini uyguladıktan sonra daha küçük özellik vektörünü sakladığımız vektörün boyutu.

Ardından, aşağıdaki kodu kullanarak yeni özellik dizisini oluşturuyoruz

new\_features = [0 for x in range(vector\_size)]

rastegele diizlere atıyor.New\_features dizisi, karma hile uygulandıktan sonra özellik bilgilerini depolar. Ardından, listeleme 8-8'deki gibi bir for döngüsü içinde hashing hilesinin anahtar işlemlerini gerçekleştiriyoruz.

for key in (1)feature\_dict:

array\_index = (2)hash(key) % vector\_size

new\_features[array\_index] += (3)feature\_dict[key]

veriyi sıkıştırıyor boyutu kadar.

Burada, özellik sözlüğündeki (1) her özelliği yinelemek için bir for döngüsü kullanıyoruz. Bunu yapmak için, önce sözlüğün anahtarlarına hash işlemi uygularız (dizge özellikleri durumunda, bunlar yazılımın ikili dizgelerine karşılık gelir) modulo vector\_size, hash değerleri sıfır ve vector\_size - 1 (2) arasında sınırlandırılır. Bu işlemin sonucunu array\_index değişkeninde saklarız.

Yine de for döngüsü içinde, array\_index dizinindeki new\_feature dizisi girişinin değerini, orijinal özellik dizimizdeki (3) değer kadar artırıyoruz. Dize özellikleri durumunda, özellik değerlerimizin yazılım ikilisinin bu belirli dizeye sahip olduğunu belirtmek için 1 olarak ayarlanması durumunda, bu girişi bir artıracağız. Özelliklerin bir değer aralığına sahip olduğu PE başlık özellikleri durumunda (örneğin, bir PE bölümünün alacağı bellek miktarına karşılık gelir), özelliğin değerini girişe ekleriz.

Son olarak, for döngüsünün dışında, basitçe new\_features sözlüğünü şu şekilde döndürürüz:  return new\_features

Bu noktada sklearn, milyonlarca benzersiz özellik yerine yalnızca binlercesini kullanarak new\_features üzerinde çalışabilir.

-Using sklearn’s FeatureHasher:Kendi karma çözümünüzü uygulamak yerine sklearn’in yerleşik uygulamasını kullanmak için önce sklearn’ın FeatureHasher sınıfını şu şekilde içe aktarmanız gerekir:

from sklearn.feature\_extraction import FeatureHasher

Ardından, FeatureHasher sınıfını somutlaştırın:

hasher = FeatureHasher(n\_features=2000)

Bunu yapmak için, karma hilesinin uygulanmasından kaynaklanan yeni dizinin boyutu olarak n\_features'ı bildirirsiniz.

Ardından, karma hilesini bazı özellik vektörlerine uygulamak için, bunları FeatureHasher sınıfının dönüştürme yöntemiyle çalıştırmanız yeterlidir:

features = [{'how': 1, 'now': 2, 'brown': 4},{'cow': 2, '.': 5}]

hashed\_features = hasher.transform(features)

Sonuç, Listeleme 8-9'da gösterilen özellik hashing hilesinin özel uygulamamızla etkili bir şekilde aynıdır. Aradaki fark, iyi korunmuş bir makine öğrenimi kitaplığını kullanmak kendi kodumuzdan daha kolay olduğu için burada sadece sklearn uygulamasını kullanıyor olmamızdır. Örnek kodun tamamı Liste 8-10'da gösterilmektedir.

->Building an Industrial-Strength Detector:Endüstriyel Mukavemet Detektörü Oluşturma

gerçek dünya dedektörümüzün üç şey yapması gerekecektir: eğitim ve tespitte kullanmak için yazılım ikili dosyalarından özellikleri çıkarmak, eğitim verilerini kullanarak kötü amaçlı yazılımları tespit etmek için kendini eğitmek ve yeni yazılım ikili dosyalarında gerçekten algılama gerçekleştirmek.

Bu bölümde kullandığım koda, bu kitaba eşlik eden koddaki malware\_data\_science / ch8 / code / complete\_detector.py adresinden veya bu kitapla sağlanan sanal makinede aynı konumdan erişebilirsiniz. Tek satırlık bir kabuk komut dosyası olan malware\_data\_science / ch8 / code / run\_complete\_detector.sh, dedektörün kabuktan nasıl çalıştırılacağını gösterir.

\*Extracting Features:Unsurları Çıkarma:

Dedektörümüzü oluşturmak için uyguladığımız ilk şey, eğitim ikili dosyalarından özellikleri çıkarmak için koddur (burada standart kodu atlıyorum ve programın temel işlevlerine odaklanıyorum). Özelliklerin ayıklanması, eğitim ikili dosyalarından ilgili verilerin çıkarılmasını, bu özelliklerin bir Python sözlüğünde saklanmasını ve daha sonra, benzersiz özelliklerimizin sayısının engelleyici bir şekilde artacağını düşünürsek, sklearn'ın karma hile uygulamasını kullanarak bunları dönüştürmeyi içerir.Basitlik uğruna, yalnızca dizgi özelliklerini kullanıyoruz ve hashing numarasını kullanmayı seçiyoruz.

get string \_features:

,Burada, hedef ikili (1) yolunu ve argümanları olarak sklearn'ın özellik karma sınıfının (2) bir örneğini alan get\_string\_features adında tek bir işlev bildiriyoruz. Ardından, hedef dosyanın dizelerini, minimum uzunluktaki tüm yazdırılabilir dizeleri ayrıştıran bir normal ifade kullanarak çıkarıyoruz. Daha sonra, her dizenin değerini sözlükte 1 olarak ayarlayarak daha fazla işlem için özellikleri bir Python sözlüğünde (3) saklıyoruz, sadece bu özelliğin ikili dosyada mevcut olduğunu belirtir.

Ardından, sklearn’in hashing hile uygulamasını kullanarak hasher'ı çağırarak özellikleri karma hale getiriyoruz. String\_features sözlüğünü bir Python listesine sardığımıza dikkat edin, çünkü sklearn tek bir sözlük yerine dönüştürülecek bir sözlükler listesini iletmemizi gerektirir.

Özellik sözlüğümüze bir sözlük listesi olarak geçtiğimiz için, özellikler diziler listesi olarak döndürülür. Ek olarak, bu kitapta tartışmayacağımız büyük matrisleri işlemek için yararlı olabilecek sıkıştırılmış bir temsil olan seyrek biçimde döndürülürler. Verilerimizi normal bir uyuşuk vektöre geri döndürmemiz gerekiyor.

Verileri normal biçime geri döndürmek için todense () ve asarray () çağırıyoruz ve ardından son özellik vektörümüzü kurtarmak için hasher sonuçları listesindeki ilk diziyi seçiyoruz. Fonksiyondaki son adım basitçe özellik vektörü hashed\_features (5) arayana geri döndürmektir.

\*Training the Detector:Dedektörü Eğitme

Sklearn, makine öğrenimi sistemlerini eğitmenin zor işlerinin çoğunu yaptığından, hedef ikili dosyalarımızdan makine öğrenimi özelliklerini çıkardıktan sonra, bir dedektörü eğitmek yalnızca küçük bir miktar kod gerektirir. Bir dedektörü eğitmek için önce eğitim örneklerimizden özellikleri çıkarmamız ve ardından kullanmak istediğimiz özellik hasherini ve sklearn makine öğrenimi algılayıcısını somutlaştırmamız gerekir (bu durumda rastgele bir orman sınıflandırıcı kullanıyoruz). Daha sonra, örneklerin ikili dosyaları üzerinde eğitmek için detektörde sklearn'ın fit yöntemini çağırmamız gerekir. Son olarak, detektörü ve özellik hasher'ını diske kaydederiz, böylece ileride dosyaları taramak istediğimizde onları kullanabiliriz.

def (1)get\_training\_data(benign\_path,malicious\_path,hasher):

def (2)get\_training\_paths(directory):

targets = []

for path in os.listdir(directory):

targets.append(os.path.join(directory,path))

return targets

(3) malicious\_paths = get\_training\_paths(malicious\_path)

(4)benign\_paths = get\_training\_paths(benign\_path)

(5) X = [get\_string\_features(path,hasher)

for path in malicious\_paths + benign\_paths]

y = [1 for i in range(len(malicious\_paths))]

+ [0 for i in range(len(benign\_paths))]

return X, y

def (6)train\_detector(X,y,hasher):

classifier = tree.RandomForestClassifier()

(7)classifier.fit(X,y)

(8) pickle.dump((classifier,hasher),open("saved\_detector.pkl","w+"))

Sağladığımız eğitim örneklerinden özellikleri çıkaran get\_training\_data () işlevini (1) bildirerek başlayalım. İşlevin üç bağımsız değişkeni vardır: iyi huylu ikili programların örneklerini içeren bir dizine giden bir yol (benign\_path), kötü niyetli ikili programların örneklerini içeren bir dizine giden yol (kötü niyetli\_yol) ve özellik karması yapmak için kullanılan sklearn FeatureHasher sınıfının bir örneği ( hasher). Daha sonra, bize belirli bir dizinde meydana gelen dosyalar için mutlak dosya yollarının listesini veren yerel bir yardımcı işlev olan get\_training\_paths () (2) 'yi beyan ederiz. Sonraki iki satırda, bize kötü niyetli (3) ve iyi huylu (4) eğitim örnek dizinlerinde oluşan yolların listelerini almak için get\_training\_paths kullanıyoruz.

Son olarak, özelliklerimizi çıkarıyoruz ve etiket vektörümüzü oluşturuyoruz. Bunu, her eğitim örneği dosya yolunda (5) Liste 8-11'de açıklanan get\_string\_features işlevini çağırarak yaparız. Etiket vektörünün her kötü niyetli yol için bir 1'e ve her iyi huylu yol için bir 0'a sahip olduğuna dikkat edin, öyle ki etiket vektöründeki indekslerdeki sayılar, X dizisindeki aynı indekslerdeki özellik vektörlerinin etiketine karşılık gelir. Bu, sklearn'ın özellik ve etiket verilerini beklediği biçimdir ve kitaplığa her özellik vektörünün etiketini söylememizi sağlar.

Artık özellikleri çıkarmayı bitirdiğimize ve özellik vektörümüzü X ve etiket vektörümüz y'yi oluşturduğumuza göre, sklearn'a özellik vektörlerini ve etiket vektörünü kullanarak dedektörümüzü eğitmesini söylemeye hazırız.

Bunu, üç bağımsız değişken alan train\_detector () işlevini (6) kullanarak yapıyoruz: eğitim örneği özellik vektörleri (X), etiket vektörü (y) ve sklearn'ın özellik karma (hasher) örneği. İşlev gövdesinde sklearn detektörü olan tree.RandomForestClassifier'ı başlatıyoruz. Ardından, onu eğitmek için dedektörün yerleştirme yöntemine X ve y'yi geçiririz (7) ve ardından dedektörü ve hasher'ı ileride kullanmak üzere kaydetmek için Python turşu picklemodülünü (8) kullanırız.

\*Running the Detector on New Binaries:Dedektörü Yeni İkililerde Çalıştırma

Şimdi, yeni program ikili dosyalarında kötü amaçlı yazılımları tespit etmek için yeni eğittiğimiz kaydedilmiş algılayıcıyı nasıl kullanacağımızı gözden geçirelim. 8-13 listesi, bunu yapmak için scan\_file () işlevinin nasıl yazılacağını gösterir.

def scan\_file(path):

if not os.path.exists("saved\_detector.pkl"):

print "Train a detector before scanning files."

sys.exit(1)

(1) with open("saved\_detector.pkl") as saved\_detector:

classifier, hasher = pickle.load(saved\_detector)

features = (2)get\_string\_features(path,hasher)

result\_proba = (3)classifier.predict\_proba(features)[1]

# if the user specifies malware\_paths and

# benignware\_paths, train a detector

(4) if result\_proba > 0.5:

print "It appears this file is malicious!",`result\_proba`

else:

print "It appears this file is benign.",`result\_proba`

Burada, bir dosyanın kötü niyetli mi yoksa zararsız mı olduğunu belirlemek için scan\_file () işlevini açıklıyoruz. Tek argümanı, tarayacağımız ikili dosyaya giden yoldur. Fonksiyonun ilk işi, kaydedilen detektörü ve hasher'ı kaydedildikleri turşu dosyasından yüklemektir (1).

Ardından, Liste 8-11'de tanımladığımız get\_string\_features (2) işlevini kullanarak hedef dosyadan özellikleri çıkarıyoruz.

Son olarak, çıkarılan özellikler göz önüne alındığında, söz konusu dosyanın kötü amaçlı olup olmadığına karar vermek için algılayıcının tahmin yöntemini çağırıyoruz. Bunu, sınıflandırıcı örneğinin predikt\_proba yöntemini (3) kullanarak ve döndürdüğü dizinin ikinci öğesini seçerek yaparız, bu da dosyanın kötü amaçlı olma olasılığına karşılık gelir. Bu olasılık 0,5 veya yüzde 50'nin (4) üzerindeyse dosyanın kötü amaçlı olduğunu söyleriz; aksi takdirde, kullanıcıya zararsız olduğunu söyleriz. Yanlış pozitifleri en aza indirmek için bu karar eşiğini çok daha yüksek bir değere değiştirebiliriz.

\*What We’ve Implemented So Far:Şimdiye Kadar Uyguladıklarımız

sonra koddan devam et