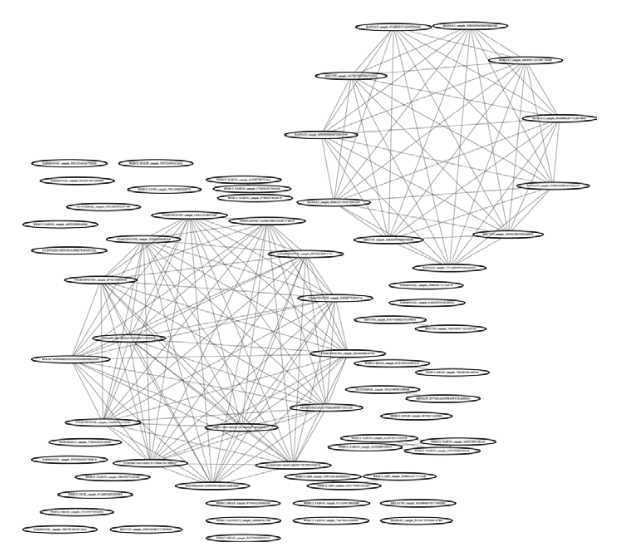
**4.Identifying Attack Campaigns Using Malware Networks:Kötü Amaçlı Yazılım Ağlarını Kullanarak Saldırı Kampanyalarını Tanımlama**

Kötü amaçlı yazılım ağ analizi malware network , kötü amaçlı yazılım veri kümelerini değerli tehdit istihbaratına dönüştürebilir, düşman saldırı kampanyalarını , yaygın kötü amaçlı yazılım  taktiklerini ve kötü amaçlı yazılım örnekleri kaynaklarını ortaya çıkarabilir. Bu yaklaşım, kötü amaçlı yazılım örnekleri gruplarının, gömülü IP adresleri, ana bilgisayar adları, yazdırılabilir karakter dizileri, grafikler veya benzeri olup olmadıklarına bakılmaksızın, paylaşılan özniteliklerine bağlı olma yollarını analiz etmekten oluşur.

Örneğin, Şekil 4-1, bu bölümde öğreneceğiniz tekniklerle oluşturulması yalnızca saniyeler süren bir grafikte kötü amaçlı yazılım ağ analizinin gücünün bir örneğini gösterir.



nation-state malwares social network connections revealed via shared attribute analysis Şekil 4-1: Ulus devlet kötü amaçlı yazılımın sosyal ağ bağlantıları, paylaşılan öznitelik analizi yoluyla ortaya çıkar.Şekil, bir grup ulus devlet düzeyinde kötü amaçlı yazılım örneğini (oval şekilli düğümler olarak temsil edilir) ve bunların "sosyal" ara bağlantılarını (düğümleri birbirine bağlayan hatlar) gösterir. Bağlantılar, bu örneklerin aynı ana bilgisayar adlarına ve IP adreslerine "geri döndükleri" ve aynı saldırganlar tarafından konuşlandırıldıklarını gösterdikleri gerçeğine dayanmaktadır. Bu bölümde öğreneceğiniz gibi, bu bağlantıları, kuruluşunuza yönelik koordineli bir saldırı ile suçla motive olmuş farklı bir dizi saldırgan arasında ayrım yapmaya yardımcı olması için kullanabilirsiniz.

Bölümün sonunda öğreneceklerim:

\*Kötü amaçlı yazılımlardan tehdit istihbaratını çıkarmakla ilgili olduğu için ağ analizi teorisinin temelleri The fundamentals of network analysis theory as it relates to extracting threat intelligence from malware

• Kötü amaçlı yazılım örnekleri arasındaki ilişkileri belirlemek için görselleştirmeleri kullanmanın yolları

• Veri analizi ve görselleştirme için Python ve çeşitli açık kaynak araç takımları kullanılarak kötü amaçlı yazılım ağlarından istihbarat nasıl oluşturulur, görselleştirilir ve çıkarılır

• Gerçek dünyadaki kötü amaçlı yazılım veri kümelerindeki saldırı kampanyalarını ortaya çıkarmak ve analiz etmek için tüm bu bilgiler nasıl bir araya getirilir

-->Nodes and Edges:

Düğümler ve Kenarlar

Kötü amaçlı yazılım üzerinde paylaşılan öznitelik analizi gerçekleştirmeden önce, ağlar hakkında bazı temel bilgileri anlamanız gerekir. Ağlar, bağlı nesnelerin koleksiyonlarıdır (düğüm adı verilir). Bu düğümler arasındaki bağlantılara kenarlar denir. Soyut matematiksel nesneler olarak, bir ağdaki düğümler, kenarları gibi hemen hemen her şeyi temsil edebilir. Amaçlarımız için önemsediğimiz şey, bu düğümler ve kenarlar arasındaki ara bağlantıların yapısıdır, çünkü bu, kötü amaçlı yazılımlar hakkında ayrıntılı bilgi verebilir.

Kötü amaçlı yazılımları analiz etmek için ağları kullanırken, her bir kötü amaçlı yazılım dosyasını bir düğümün tanımı olarak ele alabiliriz ve ilgili ilişkileri (paylaşılan kod veya ağ davranışı gibi) bir uç tanımı olarak ele alabiliriz. Benzer kötü amaçlı yazılım dosyaları kenarları paylaşır ve bu nedenle zorla yönlendirilmiş ağları uyguladığımızda birlikte kümelenir (bunun tam olarak nasıl çalıştığını daha sonra göreceksiniz). Alternatif olarak, hem kötü amaçlı yazılım örneklerini hem de özniteliklerini kendilerine ait düğümler olarak ele alabiliriz. Örneğin, geri arama IP adreslerinin düğümleri vardır ve kötü amaçlı yazılım örnekleri de öyle. Kötü amaçlı yazılım örnekleri belirli bir IP adresini geri çağırdığında, bu IP adresi düğümüne bağlanırlar.

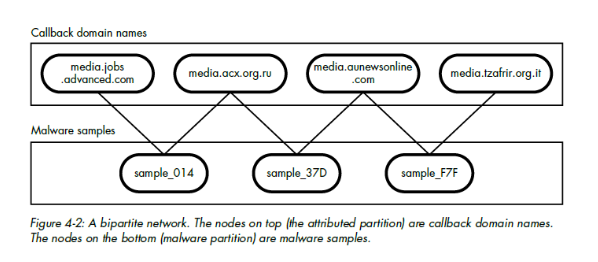
Kötü amaçlı yazılım ağları, bir dizi düğüm ve uçtan daha karmaşık olabilir. Spesifik olarak, iki bağlı örneğin paylaştığı kod yüzdesi gibi düğümlere veya kenarlara eklenmiş özniteliklere sahip olabilirler. Yaygın bir kenar özelliği, numuneler arasındaki daha güçlü bağlantıları gösteren daha büyük ağırlıklara sahip bir ağırlıktır. Düğümlerin, temsil ettikleri kötü amaçlı yazılım örneklerinin dosya boyutu gibi kendi öznitelikleri olabilir, ancak bunlar genellikle yalnızca öznitelikler attributes olarak adlandırılır.

-->Bipartite Networks:İkili Ağlar

İki taraflı ağ, düğümleri iki bölüme (gruplara) bölünebilen ağdır ve hiçbir bölüm dahili bağlantılar içermemektedir. Bu tür ağlar, kötü amaçlı yazılım örnekleri arasında paylaşılan öznitelikleri göstermek için kullanılabilir.

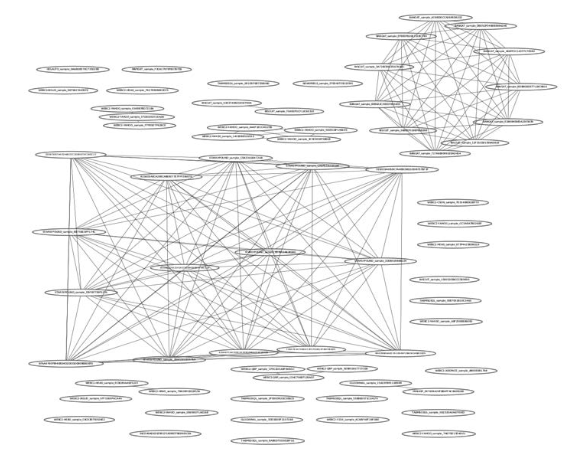
Şekil 4-2, kötü amaçlı yazılım örnek düğümlerinin alt bölüme gittiği ve diğer bölüme (saldırganla iletişim kurmak için) örneklerin "geri çağırdığı" alan adlarının bulunduğu iki taraflı bir ağ örneğini göstermektedir. Geri aramaların hiçbir zaman doğrudan diğer geri aramalara bağlanmadığını ve kötü amaçlı yazılım örneklerinin, iki taraflı bir ağın özelliği olduğu gibi hiçbir zaman doğrudan diğer kötü amaçlı yazılım örneklerine bağlanmadığını unutmayın.

Gördüğünüz gibi, bu kadar basit bir görselleştirme bile önemli bir zeka parçasını ortaya çıkarır: kötü amaçlı yazılım örneklerinin paylaşılan geri arama sunucularına dayanarak, sample\_014'ün muhtemelen sample\_37D ile aynı saldırgan tarafından konuşlandırıldığını tahmin edebiliriz. Ayrıca, sample\_37D ve sample\_F7F'nin muhtemelen aynı saldırganı paylaştığını ve bu sample\_014 ve sample\_F7F'nin muhtemelen aynı saldırganı paylaştığını tahmin edebiliriz, çünkü sample\_37D ile bağlantılılar (ve aslında, Şekil 4-2'de gösterilen örneklerin hepsi aynı "APT1" Çinli saldırgan grubu).



Ağımızdaki düğümlerin ve bağlantıların sayısı çok büyüdükçe, tüm öznitelik bağlantılarını yakından incelemeye gerek kalmadan kötü amaçlı yazılım örneklerinin nasıl ilişkili olduğunu görmek isteyebiliriz. Ortak olarak diğer bölümde (öznitelik bölümünde) düğümleri varsa, ağın bir bölümündeki düğümleri bağladığımız iki bölümlü bir ağın daha basit bir sürümü olan iki taraflı bir ağ projeksiyonu oluşturarak kötü amaçlı yazılım örnek benzerliğini inceleyebiliriz. Örneğin, Şekil 4-1'de gösterilen kötü amaçlı yazılım örnekleri söz konusu olduğunda, geri arama alan adlarını paylaşıyorlarsa kötü amaçlı yazılım örneklerinin bağlantılı olduğu bir ağ oluşturuyoruz.

  Şekil 4-3, daha önce atıfta bulunulan tüm Çin APT1 veri kümesinin paylaşılan geri arama sunucularının öngörülen ağını göstermektedir. shared-callback servers



Şekil 4-3: APT1 veri kümesinden gelen kötü amaçlı yazılım örneklerinin bir projeksiyonu, yalnızca en az bir sunucuyu paylaşıyorlarsa kötü amaçlı yazılım örnekleri arasındaki bağlantıları gösterir.

İki büyük küme clusters , iki farklı saldırı kampanyasında campaigns kullanıldı.

cluster:

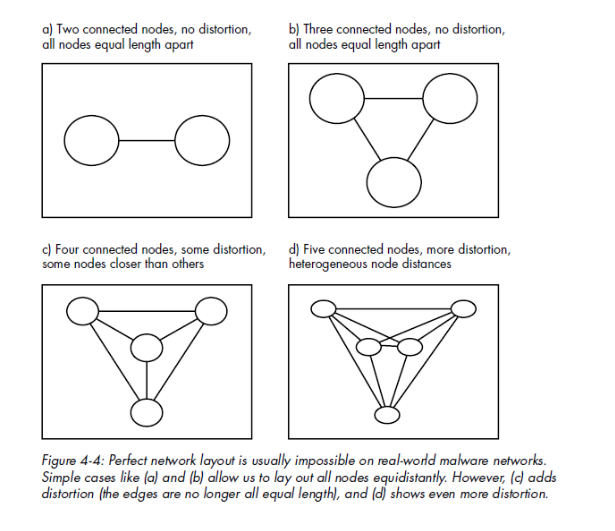
Buradaki düğümler kötü amaçlı yazılım örnekleridir ve en az bir geri arama sunucusu paylaşıyorlarsa bağlanırlar. Kötü amaçlı yazılım örnekleri arasındaki bağlantıları yalnızca geri arama sunucularını paylaşıyorlarsa göstererek, bu kötü amaçlı yazılım örneklerinin genel "sosyal ağını" görmeye başlayabiliriz. Şekil 4-3'te görebileceğiniz gibi, iki büyük gruplama mevcuttur (sol orta alandaki büyük kare küme ve sağ üst alandaki dairesel küme), bunlar daha fazla incelemenin ardından yürütülen iki farklı kampanyaya karşılık gelir. APT1 grubunun 10 yıllık geçmişi hakkında.

-->Visualizing Malware Networks:Kötü Amaçlı Yazılım Ağlarını Görselleştirme

Ağları kullanarak kötü amaçlı yazılımın paylaşılan öznitelik analizini gerçekleştirirken, şimdiye kadar gösterilenler gibi ağlar oluşturmak için büyük ölçüde ağ görselleştirme yazılımına güvendiğinizi göreceksiniz. Bu bölüm, bu ağ görselleştirmelerinin algoritmik bir bakış açısıyla nasıl oluşturulabileceğini açıklamaktadır. En önemlisi, ağ görselleştirmesindeki en büyük zorluk, görselleştirmenizin iki veya üç boyutlu bir koordinat alanı içinde bir ağdaki her bir düğümün nerede oluşturulacağına karar verme süreci olan ağ düzenidir. 3 boyutlu. Düğümleri bir ağa yerleştirirken ideal yol, onları birbirlerinden görsel uzaklıkları ağdaki en kısa yol mesafesi ile orantılı olacak şekilde koordinat boşluğuna yerleştirmektir. Başka bir deyişle, birbirinden iki atlama uzaklıkta olan düğümler, birbirinden yaklaşık iki inç uzakta olabilir ve üç atlama uzaklığındaki düğümler yaklaşık üç inç uzakta olabilir. Bunu yapmak, benzer düğüm kümelerini gerçek ilişkilerine göre doğru bir şekilde görselleştirmemize olanak tanır. Ancak bir sonraki bölümde göreceğiniz gibi, özellikle üçten fazla düğümle çalışırken bunu başarmak genellikle zordur.

\*The Distortion Problem:Bozulma Sorunu

Görünüşe göre, bu ağ düzeni sorununu mükemmel bir şekilde çözmek çoğu zaman imkansızdır. Şekil 4-4 bu zorluğu göstermektedir. Bu basit ağlarda görebileceğiniz gibi, tüm düğümler diğer tüm düğümlere eşit ağırlıktaki kenarlarla bağlıdır. Bu bağlantılar için ideal düzen, tüm düğümleri sayfaya eşit uzaklıkta yerleştirir. Ancak görebileceğiniz gibi, (c) ve (d) 'de olduğu gibi dört ve daha sonra beş düğümden oluşan ağlar oluştururken, eşit olmayan uzunluktaki kenarlar nedeniyle giderek daha fazla distorsiyon getirmeye başlıyoruz. Ne yazık ki, bu distorsiyonu sadece minimize edebiliriz, ortadan kaldırmayabiliriz ve bu minimizasyon, ağ görselleştirme algoritmalarının ana hedeflerinden biri haline gelir



gerçek dünyadaki kötü amaçlı yazılım ağlarında mükemmel ağ düzeni genellikle imkansızdır. distorsiyon (kenarların hepsi artık eşit uzunlukta değildir) ve (d) daha da fazla distorsiyon gösterir.

\*Force-Directed Algorithms:Kuvvet Yönlendirmeli Algoritmalar:

Yerleşim bozulmasını en iyi şekilde en aza indirmek için, bilgisayar bilimcileri genellikle kuvvet yönlendirmeli düzen algoritmaları kullanır. Kuvvet yönelimli algoritmalar, yay benzeri kuvvetlerin ve manyetizmanın fiziksel simülasyonlarına dayanır. Ağ kenarlarının fiziksel yaylar olarak simüle edilmesi, genellikle iyi düğüm konumlandırmasına yol açar, çünkü simüle edilmiş yaylar, düğümler ve kenarlar arasında eşit uzunluk elde etmeye çalışmak için itip çeker. Bu kavramı daha iyi görselleştirmek için, bir yayın nasıl çalıştığını düşünün: Yayı sıkıştırdığınızda veya gerdiğinizde, dengede uzunluğuna geri dönmeye "çalışır". Bu özellikler, ağımızın tüm kenarlarının eşit uzunlukta olması arzumuzla yakından ilişkilidir. Kuvvet yönlendirmeli algoritmalar bu bölümde odaklandığımız şeylerdir.

-->Building Networks with NetworkX:

Artık kötü amaçlı yazılım ağları hakkında temel bir anlayışa sahip olduğunuza göre, açık kaynaklı NetworkX Python ağ analizi kitaplığını ve GraphViz açık kaynak ağ görselleştirme araç setini kullanarak kötü amaçlı yazılım ilişkileri ağlarını nasıl oluşturacağınızı öğrenmeye hazırsınız. Kötü amaçlı yazılımla ilgili verileri programlı olarak nasıl çıkaracağınızı ve ardından bu verileri kötü amaçlı yazılım veri kümelerini temsil edecek ağları oluşturmak, görselleştirmek ve analiz etmek için nasıl kullanacağınızı gösteriyorum. Los Alamos Ulusal Laboratuvarı ve Python'un fiili ağ işleme kitaplığı merkezli bir ekip tarafından sürdürülen açık kaynaklı bir proje olan NetworkX ile başlayalım (bu bölümdeki kütüphane bağımlılıklarını NetworkX dahil olmak üzere bu bölümün kodunu girerek kurabileceğinizi hatırlayın. ve veri dizini ve pip install -r requirements.txt komutu). Python'u biliyorsanız, NetworkX'i şaşırtıcı derecede kolay bulmalısınız. NetworkX'i içe aktarmak ve bir ağ örneği oluşturmak için Liste 4-1'deki kodu kullanın.

#!/usr/bin/python

import networkx

# instantiate a network with no nodes and no edges.

network = networkx.Graph()

Bu kod, NetworkX'te bir ağ oluşturmak için NetworkX Graph constructor yapısına yalnızca bir işlev çağrısı kullanır.

Not: NetworkX kitaplığı bazen ağ yerine grafik graph  terimini kullanır, çünkü iki terim bilgisayar biliminde eşanlamlıdır - her ikisi de kenarlarla bağlanmış bir düğüm kümesini belirtir.

-->Adding Nodes and Edges:Düğümler ve Kenarlar Ekleme:

Artık bir ağ oluşturduğumuza göre, birkaç düğüm ekleyelim. NetworkX ağındaki bir düğüm herhangi bir Python nesnesi olabilir. Burada size ağımıza çeşitli türlerdeki düğümleri nasıl ekleyeceğinizi göstereceğim:

nodes = ["hello","world",1,2,3]

for node in nodes:

network.add\_node(node)

Gösterildiği gibi, ağımıza beş düğüm ekledik: "merhaba", "dünya", 1, 2 ve 3.

Ardından, kenar edges  eklemek için, aşağıda gösterildiği gibi add\_edge () adını veriyoruz:

network.add\_edge("hello","world")

network.add\_edge(1,2)

network.add\_edge(1,3)

Burada, bu beş düğümden bazılarını kenarlar aracılığıyla birbirine bağlıyoruz. Örneğin, kodun ilk satırı (1), aralarında bir kenar oluşturarak "merhaba" ve "dünya" düğümlerini birbirine bağlar.

\*Adding Attributes: Öznitelik Ekleme:

NetworkX, özellikleri hem düğümlere hem de kenarlara kolayca eklememizi sağlar. Bir düğüme bir öznitelik eklemek (ve bu özniteliğe daha sonra erişmek için), düğümü ağa eklediğinizde özniteliği bir anahtar sözcük bağımsız değişkeni olarak ekleyebilirsiniz, örneğin:

network.add\_node(1,myattribute="foo")

Daha sonra bir öznitelik eklemek için, aşağıdaki sözdizimini kullanarak ağın düğüm sözlüğüne erişin:

network.node[1]["myattribute"] = "foo"

Ardından düğüme erişmek için düğüm sözlüğüne erişin:

print network.node[1]["myattribute"] # prints "foo"

Düğümlerde nodes  olduğu gibi, burada gösterildiği gibi, başlangıçta kenarları eklediğinizde anahtar sözcük bağımsız değişkenlerini kullanarak kenarlara nitelikler ekleyebilirsiniz:

network.add\_edge("node1","node2",myattribute="attribute of an edge")

Benzer şekilde, burada gösterildiği gibi, uç edge  sözlüğünü kullanarak bir ağa eklendikten sonra kenarlara öznitelikler ekleyebilirsiniz:

network.edge["node1"]["node2"]["myattribute"] = "attribute of an edge"

Kenar sözlüğü, Liste 4-2'de gösterildiği gibi, ilk olarak hangi düğüme başvurduğunuz konusunda endişelenmenize gerek kalmadan, düğüm niteliklerine tam tersi şekilde erişmenizi sağlaması açısından büyülüdür.

network.edge["node1"]["node2"]["myattribute"] = 321

print network.edge["node2"]["node1"]["myattribute"] # prints 321

Gördüğünüz gibi, ilk satır "özniteliğimi" düğüm1 ve düğüm2'yi (1) bağlayan bir uçta ayarlar ve ikinci satır, düğüm1 ve düğüm2 referanslarının çevrilmesine rağmen "özniteliğime" erişir (2)anladığım yani hem atama işlemini yapabilir hemde atadığını tekrar yazdırma işlemini yapabilirsin.

->Saving Networks to Disk: Ağları Diske Kaydetme

Ağlarımızı görselleştirmek için, ağ analizi dünyasında yaygın olarak kullanılan ve birçok ağ görselleştirme araç setine aktarılabilen bir format olan .dot biçiminde NetworkX'ten diske kaydetmemiz gerekir. Bir ağı .dot biçiminde kaydetmek için, Liste 4-3'te gösterildiği gibi NetworkX write\_dot () işlevini çağırmanız yeterlidir.

#!/usr/bin/python

import networkx

from networkx.drawing.nx\_agraph import write\_dot

# instantiate a network, add some nodes, and connect them

nodes = ["hello","world",1,2,3]

network = networkx.Graph()

for node in nodes:

network.add\_node(node)

network.add\_edge("hello","world")

write\_dot((1)network,(2)"network.dot")

Gördüğünüz gibi, kodun sonunda, kaydetmek istediğimiz ağı (1) ve onu kaydetmek istediğimiz yolu veya dosya adını (2) belirtmek için write\_dot () işlevini kullanıyoruz.

->Network Visualization with GraphViz: GraphViz ile Ağ Görselleştirme:

Write\_dot () NetworkX işlevini kullanarak bir ağı diske yazdıktan sonra, ortaya çıkan dosyayı GraphViz kullanarak görselleştirebiliriz. GraphViz, ağlarınızı görselleştirmek için mevcut en iyi komut satırı paketidir. AT & T'deki araştırmacılar tarafından destekleniyor ve veri analistleri tarafından kullanılan ağ analizi araç kutusunun standart bir parçası haline geldi. Ağları hem düzenlemek hem de oluşturmak için kullanılabilen bir dizi komut satırı ağ düzeni aracı içerir. GraphViz, bu kitapla birlikte sağlanan sanal makineye önceden yüklenmiş olarak gelir ve ayrıca

https://graphviz.gitlab.io/ download / adresinden indirilebilir. Her GraphViz komut satırı aracı .dot biçiminde ifade edilen ağları besler ve bir ağı .png dosyası olarak işlemek için aşağıdaki sözdizimi kullanılarak çağrılabilir:

$ <toolname> <dotfile> -T png –o <outputfile.png>

Fdp kuvvet yönelimli force-directed grafik oluşturucu, bir GraphViz ağ görselleştirme aracıdır. Burada gösterildiği gibi, diğer tüm GraphViz araçlarıyla aynı temel komut satırı arayüzünü kullanır:

$ fdp apt1callback.dot –T png –o apt1callback.png

Burada fdp aracını kullanmak istediğimizi ve bu kitaba eşlik eden verilerin ~ / ch3 / dizininde bulunan apt1callback.dot olan düzenlemek istediğimiz ağ .dot dosyasını adlandırmak istediğimizi belirtiyoruz. Kullanmak istediğimiz formatı (PNG) belirtmek için –T png belirtiyoruz. Son olarak, -o apt1callback.png kullanarak çıktı dosyasının nereye kaydedilmesini istediğimizi belirtiyoruz.

\*Using Parameters to Adjust Networks: Ağları Ayarlamak için Parametreleri Kullanma:

GraphViz araçları, ağlarınızın çizilme şeklini ayarlamak için kullanabileceğiniz birçok parametre içerir. Bu parametrelerin çoğu, aşağıdaki formatta –G komut satırı bayrağı kullanılarak ayarlanır:

G<parametername>=<parametervalue>

Özellikle kullanışlı olan iki parametre örtüşme ve eğrilerdir. GraphViz'e herhangi bir düğümün birbiriyle çakışmasına izin vermemesini söylemek için örtüşmeyi false olarak ayarlayın. Ağlarınızdaki kenarları takip etmeyi kolaylaştırmak için GraphViz'e düz çizgiler yerine eğri çizgiler çizmesini söylemek için splines parametresini kullanın. Aşağıda, GraphViz'de örtüşme ve eğri parametrelerini ayarlamanın bazı yolları verilmiştir. Düğümlerin çakışmasını önlemek için aşağıdakileri kullanın:

$ <toolname> <dotfile> -Goverlap=false -T png -o <outputfile.png>

Ağ okunabilirliğini iyileştirmek için kenarları eğri çizgiler (spline) olarak çizin:

$ <toolname> <dotfile> -Gsplines=true -T png -o <outputfile.png>

Ağın okunabilirliğini artırmak için kenarları eğri çizgiler (spline'lar) olarak çizin ve düğümlerin görsel olarak örtüşmesine izin vermeyin:

$ <toolname> <dotfile> -Gsplines=true –Goverlap=false -T png -o <outputfile.png>

Sadece bir parametreyi birbiri ardına listelediğimizi unutmayın:

Gsplines = true –Goverlap = false (sıralama önemli değildir), ardından -T png -o <outputfile.png> gelir. Bir sonraki bölümde, en kullanışlı GraphViz araçlarından bazılarının (fdp'ye ek olarak) üzerinden geçeceğim.

->The GraphViz Command Line Tools:

İşte en yararlı bulduğum mevcut GraphViz araçlarından bazıları ve her bir aracı kullanmanın ne zaman uygun olduğuna dair bir fikir.

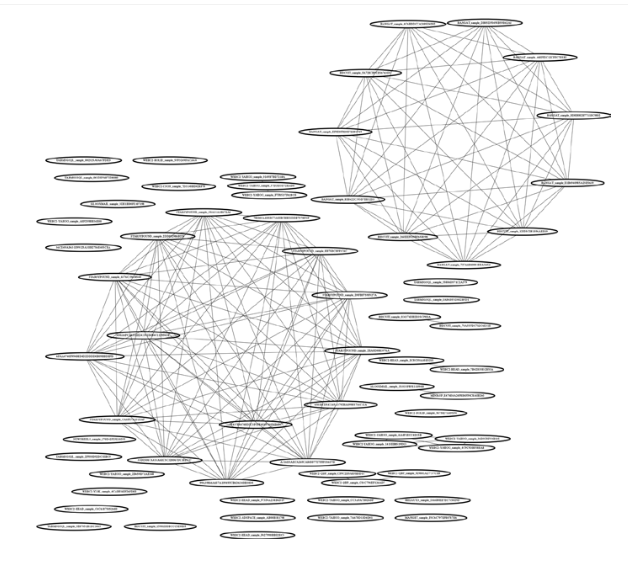
\*fdp:

Önceki örnekteki fdp mizanpaj layout  aracını, 40. sayfadaki "Kuvvet Yönlendirmeli Algoritmalar" bölümünde açıklandığı gibi zorla yönlendirilmiş bir düzen oluşturmak için kullandık. 500'den az düğüme sahip kötü amaçlı yazılım ağları oluştururken, fdp yapar makul bir süre içinde ağ yapısını ortaya çıkarmak için iyi bir iş. Ancak 500'den fazla düğümle çalışırken ve özellikle düğümler arasındaki bağlantı karmaşık olduğunda, fdp'nin oldukça hızlı bir şekilde yavaşladığını göreceksiniz.

Şekil 4-3'te gösterilen APT1 paylaşılan geri arama sunucusu ağında fdp'yi denemek için, bu kitaba eşlik eden verilerin ch4 dizininden aşağıdakileri girin (GraphViz'in kurulu olması gerekir):

$ fdp callback\_servers\_malware\_projection.dot -T png -o fdp\_servers.png –Goverlap=false

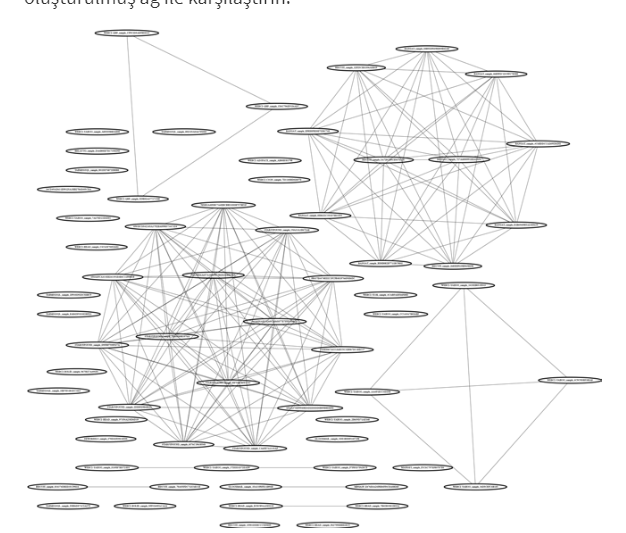
Bu komut, Şekil 4-5'te gösterilene benzer bir ağı gösteren bir .png dosyası (fdp\_servers.png) oluşturacaktır.



Fdp düzeni, şekilde bir dizi temayı görünür kılar. İlk olarak, şeklin sağ üst ve sol alt alanlarında açıkça görüldüğü gibi, iki büyük örnek kümesi birbiriyle oldukça ilişkilidir. İkinci olarak, sağ altta görülebilen birkaç örnek çifti birbiriyle ilişkilidir. Son olarak, birçok örnek birbiriyle belirgin bir ilişki göstermez ve başka herhangi bir düğümle bağlantılı değildir. Bu görselleştirmenin, düğümler arasındaki paylaşılan geri arama sunucusu ilişkilerine dayandığını hatırlamak önemlidir. Bağlantısız örneklerin, Bölüm 5'te inceleyeceğimiz paylaşılan kod ilişkileri-ilişkileri relationships gibi başka tür ilişkiler yoluyla şekildeki diğer örneklerle ilişkili olması mümkündür.

\*sfdp:

Sfdp aracı, mizanpaj layout  için fdp ile hemen hemen aynı yaklaşımı kullanır, ancak daha iyi ölçeklenir, çünkü düğümlerin yakınlıklarına göre süper düğümlerle birleştirildiği, kabalaştırmalar coarsening  olarak bilinen bir basitleştirme hiyerarşisi oluşturur.. Sfdp aracı, kabalaştırmalarını tamamladıktan sonra, grafiğin çok daha az düğüm ve kenara sahip birleştirilmiş sürümlerini düzenler ve bu da yerleşim sürecini önemli ölçüde hızlandırır. Bu şekilde sfdp, ağdaki en iyi konumları bulmak için daha az hesaplama gerçekleştirebilir. Sonuç olarak, sfdp tipik bir dizüstü bilgisayarda on binlerce düğümü yerleştirebilir ve bu da onu çok büyük kötü amaçlı yazılım ağlarını yerleştirmek için açık ara en iyi algoritma yapar. Ancak bu ölçeklenebilirliğin bir bedeli vardır: sfdp, bazen fdp'deki aynı boyutlu ağların düzenlerinden daha az net olan düzenler üretir. Örneğin, sfdp kullanarak oluşturduğum Şekil 4-6'yı, Şekil 4-5'te gösterilen fdp ile oluşturulmuş ağ ile karşılaştırın.



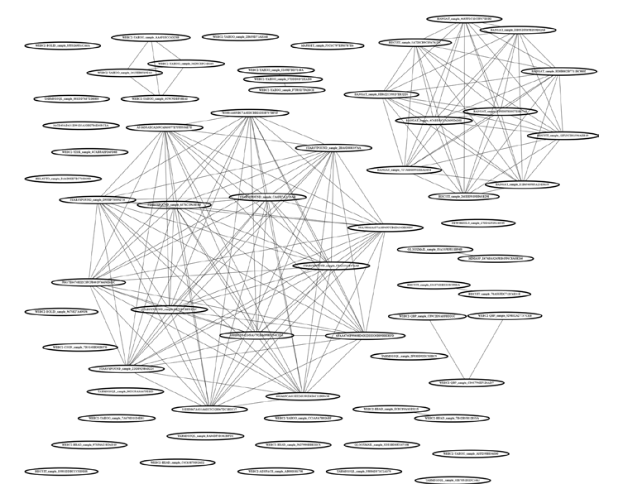
Gördüğünüz gibi, Şekil 4-6'daki her bir kümenin üzerinde biraz daha fazla gürültü var, bu da neler olduğunu görmeyi biraz daha zorlaştırıyor. Bu ağı oluşturmak için, bu kitaba eşlik eden verilerin ch4 dizinini girin ve ardından Şekil 4-6'da gösterilen sfdp\_servers.png görüntü dosyasını oluşturmak için aşağıdaki kodu girin:

$ sfdp callback\_servers\_malware\_projection.dot -T png -o sfdp\_servers.png –Goverlap=false

Bu koddaki ilk öğenin, önceki fdp yerine sfdp aracını kullandığımızı nasıl belirttiğine dikkat edin. Diğer her şey aynı, çıktı dosya adını kaydedin.

\*neato:

Neato aracı, nesneleri ideal konumlara itmeye yardımcı olmak için tüm düğümler arasında (bağlı olmayan düğümler dahil) simüle edilmiş yaylar oluşturan farklı bir kuvvet yönelimli ağ düzeni algoritmasının GraphViz uygulamasıdır, ancak ek hesaplama pahasına. Neato'nun belirli bir ağ için en iyi düzeni ne zaman üreteceğini bilmek zor: Benim tavsiyem, onu fdp ile birlikte denemeniz ve hangi düzeni daha çok beğendiğinizi görmenizdir. Şekil 4-7, APT1 paylaşılan geri arama sunucusu ağında neato düzeninin nasıl göründüğünü gösterir.



Gördüğünüz gibi, bu durumda neato, fdp ve sfdp tarafından üretilenlere benzer bir ağ düzeni üretir. Bununla birlikte, bazı veri kümeleri için, neato'nun daha iyi veya daha kötü bir düzen ürettiğini göreceksiniz - sadece veri kümenizle denemeniz ve görmeniz gerekir. Neato'yu denemek için, bu kitaba eşlik eden verilerin ch4 dizininden aşağıdakileri girin; bu, Şekil 4-7'de gösterilen neato\_servers.png ağ görüntü dosyasını oluşturmalıdır:

$ neato callback\_servers\_malware\_projection.dot -T png -o neato\_servers.png –Goverlap=false

Bu ağı oluşturmak için, neato aracını kullanmak istediğimizi belirtmek için Şekil 4-6'yı oluştururken kullandığımız kodu revize ediyoruz ve sonra .png'yi neato\_servers.png'ye kaydediyoruz. Artık bu ağ görselleştirmelerini nasıl oluşturacağınızı bildiğinize göre, bunları iyileştirmenin yollarına bakalım.

->Adding Visual Attributes to Nodes and Edges:Düğümlere ve Kenarlara Görsel Nitelikler Ekleme:

Genel ağ düzeninize karar vermenin ötesinde, tek tek düğümlerin ve kenarların nasıl işleneceğini belirtmek faydalı olabilir. Örneğin, iki düğüm arasındaki bağlantının gücüne göre kenar kalınlığını ayarlamak veya düğüm rengini her kötü amaçlı yazılım örnek düğümünün ilişkili olduğu tehlikeye göre ayarlamak isteyebilirsiniz; bu, kötü amaçlı yazılım kümelerini daha iyi görselleştirmenize olanak tanır. NetworkX ve GraphViz, basitçe bir öznitelik kümesine değerler atayarak düğümlerin ve kenarların görsel özniteliklerini belirlemenize izin vererek bunu yapmayı kolaylaştırır. Sonraki bölümlerde bu türden yalnızca birkaç özelliği tartışıyorum, ancak bu konu tüm kitabı dolduracak kadar derin.

\*Edge Width:

GraphViz'in düğümlerin etrafına çizdiği sınırın genişliğini veya kenarlar için çizdiği çizgiyi ayarlamak için, Liste 4-4'te gösterildiği gibi düğümlerin ve kenarların kalem genişliği özniteliğini istediğiniz bir sayıya ayarlayabilirsiniz.

#!/usr/bin/python

import networkx

from networkx.drawing.nx\_agraph import writedot

(1) g = networkx.Graph()

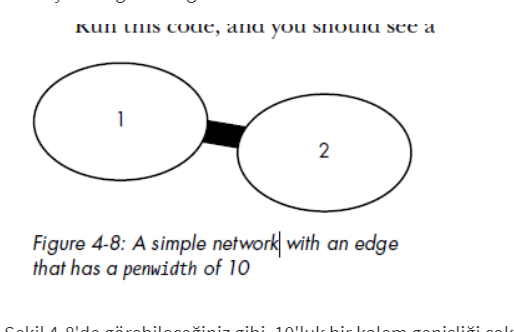
g.add\_node(1)

g.add\_node(2)

g.add\_edge(1,2,(2)penwidth=10) # make the edge extra wide

write\_dot(g,'network.dot')

Burada, iki düğümün bir uçla birbirine bağlı olduğu basit bir ağ (1) oluşturuyorum ve kenarın kalem genişliği özelliğini 10 (2) olarak ayarlıyorum (varsayılan değer 1'dir). Bu kodu çalıştırın ve Şekil 4 8'e benzeyen bir görüntü görmelisiniz.



Şekil 4-8'de görebileceğiniz gibi, 10'luk bir kalem genişliği çok kalın bir kenarla sonuçlanır. Kenarın genişliği (veya bir düğümün uç genişliğini ayarlarsanız düğümün kenarlığının kalınlığı), kalem genişliği niteliğinin değeriyle orantılı olarak ölçeklenir, bu nedenle uygun şekilde seçin. Örneğin, kenar gücü değerleriniz 1 ila 1000 arasında değişiyorsa, ancak tüm kenarları görebilmek istiyorsanız, kenar gücü değerlerinizin günlük ölçeklemesine göre kalem genişliği nitelikleri atamayı düşünebilirsiniz.

\*Node and Edge Color:

Bir düğümün kenarlığının veya kenarının rengini ayarlamak için renk özelliğini kullanın. 4-5 listesi bunun nasıl yapılacağını gösterir.

#!/usr/bin/python

import networkx

from networkx.drawing.nx\_agraph import write\_dot

g = networkx.Graph()

g.add\_node(1,ucolor="blue") # make the node outline blue

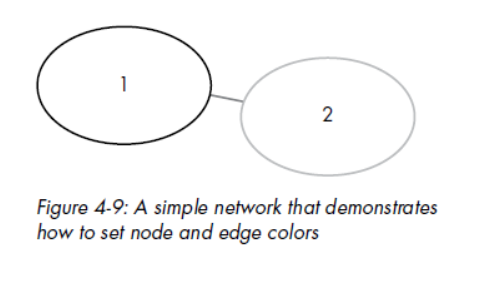
g.add\_node(2,vcolor="pink") # make the node outline pink

g.add\_edge(1,2,wcolor="red") # make the edge red

write\_dot(g,'network.dot')

Burada, Liste 4-4'te oluşturduğum aynı basit ağı, iki düğüm ve bunları birbirine bağlayan bir uçla oluşturuyorum. Oluşturduğum her düğüm için renk değerini ((1) ve (2)) ayarlıyorum. Ayrıca oluşturduğumda kenar (3) için renk değerini de ayarlıyorum.

Şekil 4-9, Liste 4-5'in sonucunu göstermektedir. Beklendiği gibi, ilk düğümün (kenar) ve ikinci düğümün her birinin benzersiz bir renge sahip olduğunu görmelisiniz. Kullanabileceğiniz tam bir renk listesi için http://www.graphviz.org/doc/ info / colours.html adresine bakın.



Renkler, farklı düğüm ve kenar sınıflarını göstermek için kullanılabilir.

\*Node Shape:Düğüm Şekli

Bir düğümün şeklini ayarlamak için, http://www.GraphViz.org/doc/info/s apes.html adresinde tanımlandığı gibi, bir şekil belirten bir dizeyle şekil özniteliğini kullanın. Yaygın olarak kullanılan değerler kutu, elips, daire, yumurta, elmas, üçgen, beşgen ve altıgendir. 4-6 listesi, bir düğümün şekil özniteliğinin nasıl ayarlanacağını gösterir.

#!/usr/bin/python

import networkx

from networkx.drawing.nx\_agraph import write\_dot

g = networkx.Graph()

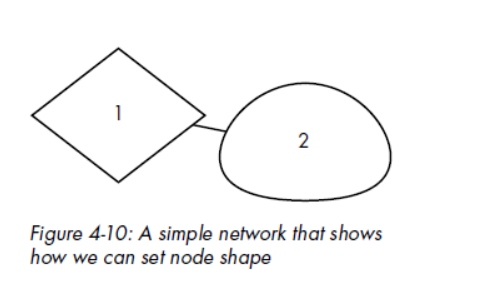
g.add\_node(1,ushape='diamond')

g.add\_node(2,vshape='egg')

g.add\_edge(1,2)

write\_dot(g,'network.dot')

Bir düğümün rengini belirleme şeklimize benzer şekilde, her bir düğümün almasını istediğimiz şekli belirtmek için add\_node () işlevinde şekil anahtar sözcüğü bağımsız değişkenini kullanırız. Burada birinci düğümü bir elmas şekline (1) ve ikinci düğümü bir yumurta şekline (2) ayarlıyoruz. Bu kodun sonucu Şekil 4-10'da gösterilmektedir.



Elmas şeklindeki bir düğümü ve yumurta şeklindeki bir düğümü gösteren sonuçlar, Liste 4-6'da belirttiğimiz şekilleri yansıtmaktadır.

\*Text Labels:

Son olarak, GraphViz ayrıca etiket özniteliğiyle düğümlere ve kenarlara etiket eklemenize izin verir. Düğümler, atanan kimliklerine göre otomatik olarak etiketlense de (örneğin, 123 olarak eklenen bir düğümün etiketi 123 olacaktır), etiket = <etiketim özniteliğim> kullanarak etiketleri belirtebilirsiniz. Düğümlerden farklı olarak, kenarlar varsayılan olarak etiketlenmez, ancak etiket özelliğini kullanarak bunlara etiketler atayabilirsiniz. Liste 4-7, artık tanıdık iki düğümlü ağımızın nasıl oluşturulacağını, ancak her iki düğüme ve bağlantı ucuna eklenen etiket niteliklerini gösterir.etiket labels

#!/usr/bin/python

import networkx

from networkx.drawing.nx\_agraph import write\_dot

g = networkx.Graph()

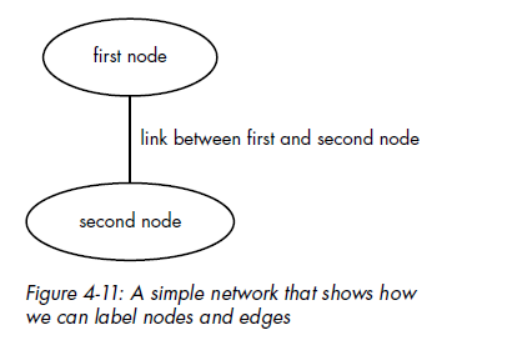
g.add\_node(1,(1)label="first node")

g.add\_node(2,(2)label="second node")

g.add\_edge(1,2,(3)label="link between first and second node")

write\_dot(g,'network.dot')

Sırasıyla birinci düğümü (1) ve ikinci düğümü (2) etiketliyoruz. Bunları bağlayan kenarı da birinci ve ikinci düğüm (3) arasındaki bağlantı olarak etiketliyoruz. Şekil 4-11 beklediğimiz grafik çıktıyı göstermektedir.



Artık düğümlerin ve kenarların temel özelliklerini nasıl kullanacağınızı bildiğinize göre, ağları sıfırdan oluşturmaya hazırsınız.

-->Building Malware Networks:

Kötü amaçlı yazılım ağları oluşturma tartışmamıza, Şekil 4-1'de gösterilen paylaşılan geri arama sunucusu örneğini yeniden oluşturarak ve genişleterek başlayacağız ve ardından kötü amaçlı yazılımın paylaşılan görüntü analizini inceleyeceğiz. Aşağıdaki program, geri arama etki alanı adlarını kötü amaçlı yazılım dosyalarından çıkarır ve ardından kötü amaçlı yazılım örneklerinden oluşan iki taraflı bir ağ oluşturur. Daha sonra, hangi kötü amaçlı yazılım örneklerinin ortak geri arama sunucularını paylaştığını göstermek için ağın bir projeksiyonunu gerçekleştirir ve yaygın kötü amaçlı yazılım örnekleri tarafından hangi geri arama sunucularının çağrıldığını göstermek için başka bir projeksiyon gerçekleştirir. Son olarak, program üç ağı (orijinal iki taraflı ağ, kötü amaçlı yazılım örnek projeksiyonu ve geri arama sunucusu projeksiyonu) GraphViz ile görselleştirilebilmeleri için dosyalar olarak kaydeder. Program boyunca size parça parça rehberlik ediyorum. Kodun tamamı, bu kitaba eşlik eden verilerde ch4 / callback\_server\_network.py dosya yolunda bulunabilir. 4-8 listesi, gerekli modülleri içe aktararak nasıl başlayacağınızı gösterir.

#!/usr/bin/python

import pefile(1)

import sys

import argparse

import os

import pprint

import networkx(2)

import re

from networkx.drawing.nx\_agraph import write\_dot

import collections

from networkx.algorithms import bipartite

İçe aktardığımız gerekli modüllerden en önemlileri, hedef PE ikili dosyalarını ayrıştırmak için kullandığımız pefile PE ayrıştırma modülü (1) ve kötü amaçlı yazılım öznitelik ağını oluşturmak için kullandığımız networkx kitaplığıdır (2). Sonra, kodu Liste 4-9'a ekleyerek komut satırı argümanlarını ayrıştırıyoruz.

args = argparse.ArgumentParser("Visualize shared DLL import relationships

between a directory of malware samples")

args.add\_argument((1)"target\_path",help="directory with malware samples")

args.add\_argument((2)"output\_file",help="file to write DOT file to")

args.add\_argument((3)"malware\_projection",help="file to write DOT file to")

args.add\_argument((4)"resource\_projection",help="file to write DOT file to")

args = args.parse\_args()

4.9 list:

Bu argümanlar arasında target\_path (1) (analiz ettiğimiz kötü amaçlı yazılımın bulunduğu dizine giden yol), output\_file (2) (tüm ağı yazdığımız yol), malware\_projection (3) (indirgenmiş bir grafiğin sürümü ve hangi kötü amaçlı yazılım örneklerinin öznitelikleri paylaştığını gösterir) ve resource\_projection (4) (grafiğin azaltılmış bir sürümünü yazdığımız ve kötü amaçlı yazılım örneklerinde hangi özniteliklerin birlikte görüldüğünü gösterdiğimiz yol). Artık programın özüne core geçmeye hazırız. 4-10 listesi, program için bir ağ oluşturma kodunu gösterir.

#!/usr/bin/python

import pefile

(1) import sys

import argparse

import os

import pprint

import networkx

import re

from networkx.drawing.nx\_agraph import write\_dot

import collections

from networkx.algorithms import bipartite

args = argparse.ArgumentParser(

"Visualize shared hostnames between a directory of malware samples"

)

args.add\_argument("target\_path",help="directory with malware samples")

args.add\_argument("output\_file",help="file to write DOT file to")

args.add\_argument("malware\_projection",help="file to write DOT file to")

args.add\_argument("hostname\_projection",help="file to write DOT file to")

args = args.parse\_args()

network = networkx.Graph()

valid\_hostname\_suffixes = map(

lambda string: string.strip(), open("domain\_suffixes.txt")

)

valid\_hostname\_suffixes = set(valid\_hostname\_suffixes)

(2)def find\_hostnames(string):

possible\_hostnames = re.findall(

r'(?:[a-zA-Z0-9](?:[a-zA-Z0-9\-]{,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,6}',

string)

valid\_hostnames = filter(

lambda hostname: hostname.split(".")[-1].lower() \

in valid\_hostname\_suffixes,

possible\_hostnames

)

return valid\_hostnames

# search the target directory for valid Windows PE executable files

for root,dirs,files in os.walk(args.target\_path):

for path in files:

# try opening the file with pefile to see if it's really a PE file

try:

pe = pefile.PE(os.path.join(root,path))

except pefile.PEFormatError:

continue

fullpath = os.path.join(root,path)

# extract printable strings from the target sample

(3) strings = os.popen("strings '{0}'".format(fullpath)).read()

# use the search\_doc function in the included reg module

# to find hostnames

(4) hostnames = find\_hostnames(strings)

if len(hostnames):

# add the nodes and edges for the bipartite network

network.add\_node(path,label=path[:32],color='black',penwidth=5,

bipartite=0)

for hostname in hostnames:

(5) network.add\_node(hostname,label=hostname,color='blue',

penwidth=10,bipartite=1)

network.add\_edge(hostname,path,penwidth=2)

if hostnames:

print "Extracted hostnames from:",path

pprint.pprint(hostnames)

Listing 4-10: Creating the network:

Önce networkx.Graph () yapıcısını (1) çağırarak yeni bir ağ oluşturuyoruz. Ardından dizelerden (2) ana bilgisayar adlarını çıkaran find\_hostnames () işlevini tanımlarız. Bu işlevin mekaniği hakkında çok fazla endişelenmeyin: Bu aslında bir normal ifade ve etki alanlarını tanımlamak için elinden gelenin en iyisini yapmaya çalışan bir dizi filtreleme kodudur. Daha sonra, hedef dizindeki tüm dosyaları yineleyerek, pefile.PE sınıfının bunları yükleyip yükleyemediğini görerek bunların PE dosyası olup olmadığını kontrol ederiz (yoksa, dosyaları analiz etmeyiz). Son olarak, önce dosyadan (3) tüm yazdırılabilir dizeleri çıkararak ve ardından katıştırılmış ana bilgisayar adı kaynakları (4) için dizeleri arayarak ana bilgisayar adı niteliklerini mevcut dosyadan çıkarıyoruz. Herhangi birini bulursak, bunları ağımıza düğümler olarak ekleriz ve ardından mevcut kötü amaçlı yazılım örneği için düğümden ana bilgisayar adı kaynak düğümlerine (5) kenarlar ekleriz. Şimdi, Liste 4-11'de gösterildiği gibi programı tamamlamaya hazırız.

# write the dot file to disk

(1) write\_dot(network, args.output\_file)

(2) malware = set(n for n,d in network.nodes(data=True) if d['bipartite']==0)

(3) hostname = set(network)-malware

# use NetworkX's bipartite network projection function to produce the malware

# and hostname projections

(4) malware\_network = bipartite.projected\_graph(network, malware)

hostname\_network = bipartite.projected\_graph(network, hostname)

# write the projected networks to disk as specified by the user

(5) write\_dot(malware\_network,args.malware\_projection)

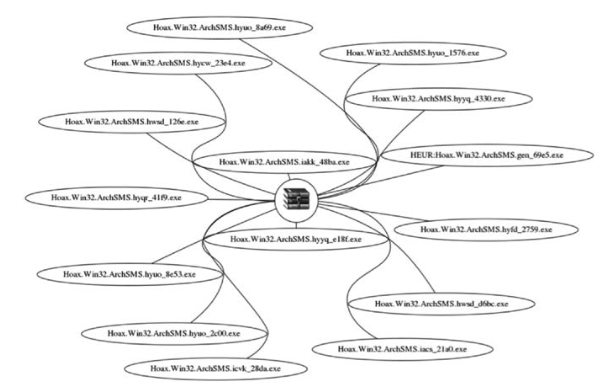
write\_dot(hostname\_network,args.hostname\_projection)

Listing 4-11: Writing networks to files:

Ağımızı komut satırı argümanlarında (1) belirtilen konumda diske yazarak başlıyoruz. Ardından, kötü amaçlı yazılım ilişkilerini ve ana bilgisayar adı kaynak ilişkilerini gösteren iki azaltılmış ağı (bu bölümde daha önce anlatılan "tahminler projections") oluşturuyoruz. Bunu, önce kötü amaçlı yazılım düğümlerinin (2) kimliklerini içeren bir Python seti ve kaynak düğümlerinin (3) kimlikleri için başka bir Python seti oluşturarak yapıyoruz. Daha sonra kötü amaçlı yazılım ve kaynak kümeleri için projeksiyonlar almak için NetworkXspecific projected\_graph () işlevini (4) kullanırız ve bu ağları belirtilen konumlardaki (5) diske yazarız. Ve bu kadar! Dosyalara gömülü paylaşılan ana bilgisayar adı kaynakları arasındaki kötü amaçlı yazılım ilişkilerini görmek için bu programı bu kitaptaki kötü amaçlı yazılım veri kümelerinin herhangi birinde kullanabilirsiniz. Bu analiz modu aracılığıyla hangi tehdit istihbaratını toplayabileceğinizi görmek için kendi veri kümelerinizde bile kullanabilirsiniz.

->Building a Shared Image Relationship Network:Paylaşılan Bir Görüntü İlişkisi Ağı Oluşturmak:

Kötü amaçlı yazılımları, paylaşılan geri arama sunucularına göre analiz etmenin yanı sıra, bunları, paylaşılan simgeler ve diğer grafik varlıkları kullanımlarına göre de analiz edebiliriz. Örneğin, Şekil 4-12 ch4 / data / Truva atlarında bulunan Truva atları için paylaşılan görüntü analizi sonuçlarının bir bölümünü gösterir.



Şekil 4-12: Bir dizi Truva atı için paylaşılan görüntü varlığı ağının görselleştirmesi shared image asset

Tüm bu Truva atlarının arşiv dosyaları gibi göründüğünü ve çalıştırılabilir olsalar bile aynı arşiv dosyası simgesini (şeklin ortasında gösterilir) kullandıklarını görebilirsiniz. Kullanıcıyı oyun oynama çabalarının bir parçası olarak tamamen aynı görüntüyü kullanmaları, muhtemelen aynı saldırgandan geldiklerini gösterir. Bunu, hepsine aynı aile adını (ArchSMS) atayan Kaspersky antivirüs motoru aracılığıyla kötü amaçlı yazılım örneklerini çalıştırarak doğruladım.

Daha sonra, kötü amaçlı yazılım örnekleri arasındaki paylaşılan görüntü ilişkilerini görmek için size Şekil 4-12'de gösterilen görselleştirme türünü nasıl oluşturacağınızı göstereceğim. Görüntüleri kötü amaçlı yazılımdan çıkarmak için, image\_network.py programını oluşturmak için wrestool'a (Bölüm 1'de ele alınmıştır) dayanan yardımcı kitaplık görüntülerini kullanıyoruz. Wrestool'un Windows çalıştırılabilir dosyalarından görüntüleri çıkardığını hatırlayın.

Kodun Liste 4-12'de gösterilen ilk bölümünden başlayarak, paylaşılan bir görüntü ağı oluşturma sürecini inceleyelim.

#!/usr/bin/python

import pefile

import sys

import argparse

import os

import pprint

import logging

import networkx

import collections

import tempfile

from networkx.drawing.nx\_agraph import write\_dot

from networkx.algorithms import bipartite

# Use argparse to parse any command line arguments

args = argparse.ArgumentParser(

"Visualize shared image relationships between a directory of malware samples"

)

args.add\_argument("target\_path",help="directory with malware samples")

args.add\_argument("output\_file",help="file to write DOT file to")

args.add\_argument("malware\_projection",help="file to write DOT file to")

args.add\_argument("resource\_projection",help="file to write DOT file to")

args = args.parse\_args()

network = networkx.Graph()

(1) class ExtractImages():

def \_\_init\_\_(self,target\_binary):

self.target\_binary = target\_binary

self.image\_basedir = None

self.images = []

def work(self):

self.image\_basedir = tempfile.mkdtemp()

icondir = os.path.join(self.image\_basedir,"icons")

bitmapdir = os.path.join(self.image\_basedir,"bitmaps")

raw\_resources = os.path.join(self.image\_basedir,"raw")

for directory in [icondir,bitmapdir,raw\_resources]:

os.mkdir(directory)

rawcmd = "wrestool -x {0} -o {1} 2> \

/dev/null".format(

self.target\_binary,raw\_resources

)

bmpcmd = "mv {0}/\*.bmp {1} 2> /dev/null".format(

raw\_resources,bitmapdir

)

icocmd = "icotool -x {0}/\*.ico -o {1} \

2> /dev/null".format(

raw\_resources,icondir

)

for cmd in [rawcmd,bmpcmd,icocmd]:

try:

os.system(cmd)

except Exception,msg:

pass

for dirname in [icondir,bitmapdir]:

for path in os.listdir(dirname):

logging.info(path)

path = os.path.join(dirname,path)

imagehash = hash(open(path).read())

if path.endswith(".png"):

self.images.append((path,imagehash))

if path.endswith(".bmp"):

self.images.append((path,imagehash))

def cleanup(self):

os.system("rm -rf {0}".format(self.image\_basedir))

# search the target directory for PE files to extract images from

image\_objects = []

for root,dirs,files in os.walk(args.target\_path):(2)

for path in files:

# try to parse the path to see if it's a valid PE file

try:

pe = pefile.PE(os.path.join(root,path))

except pefile.PEFormatError:

continue

Liste 4-12: Paylaşılan görüntü ağı programımızda ilk bağımsız değişkeni ve dosya yükleme kodunu ayrıştırma

Program az önce tartıştığımız ana bilgisayar adı grafik programına çok benzer (Liste 4-8'den başlayarak). İlk önce pefile ve networkx dahil olmak üzere bir dizi modülü içe aktarır. Ancak burada, hedef kötü amaçlı yazılım örneklerinden grafiksel varlıkları çıkarmak için kullandığımız ExtractImages yardımcı sınıfını (1) da tanımlıyoruz. Ardından program, tüm hedef kötü amaçlı yazılım ikili dosyalarını yinelediğimiz bir döngüye girer (2).

Artık döngümüzde olduğumuza göre, ExtractImages sınıfını kullanarak hedef kötü amaçlı yazılım ikili dosyalarından grafik varlıkları ayıklamanın zamanı geldi (bu, Bölüm 1'de tartışılan icoutils programlarının etrafına bir sarmalayıcıdır). 4-13 listesi, kodun bunu yapan kısmını gösterir.

fullpath = os.path.join(root,path)

(1) images = ExtractImages(fullpath)

(2) images.work()

image\_objects.append(images)

# create the network by linking malware samples to their images

(3) for path, image\_hash in images.images:

# set the image attribute on the image nodes to tell GraphViz to

# render images within these nodes

if not image\_hash in network:

(4) network.add\_node(image\_hash,image=path,label='',type='image')

node\_name = path.split("/")[-1]

network.add\_node(node\_name,type="malware")

(5) network.add\_edge(node\_name,image\_hash)

Liste 4-13: Hedef kötü amaçlı yazılımdan grafiksel varlıkları çıkarma

İlk olarak, ExtractImages sınıfına (1) hedef kötü amaçlı yazılım ikili dosyasına giden yolu iletiriz ve ardından ortaya çıkan örneğin work () yöntemini (2) çağırırız. Bu, ExtractImages sınıfının kötü amaçlı yazılım görüntülerini sakladığı geçici bir dizin oluşturmasına ve ardından görüntüler sınıfı özniteliğinde her görüntü hakkında verileri içeren bir sözlüğü depolamasına neden olur. Artık ExtractImages'dan çıkarılan görüntülerin listesine sahip olduğumuza göre, üzerinde yineliyoruz (3), karmasını daha önce görmediysek bir görüntü için yeni bir ağ düğümü oluşturuyoruz (4) ve şu anda işlenen kötü amaçlı yazılım örneğini ağdaki görüntü (5).

Artık içerdikleri görüntülerle bağlantılı kötü amaçlı yazılım örnekleri ağımızı oluşturduğumuza göre, Liste 4-14'te gösterildiği gibi grafiği diske yazmaya hazırız.

# write the bipartite network, then do the two projections and write them

(1) write\_dot(network, args.output\_file)

malware = set(n for n,d in network.nodes(data=True) if d['type']=='malware')

resource = set(network) - malware

malware\_network = bipartite.projected\_graph(network, malware)

resource\_network = bipartite.projected\_graph(network, resource)

(2) write\_dot(malware\_network,args.malware\_projection)

write\_dot(resource\_network,args.resource\_projection)

Bunu, Liste 4-11'de yaptığımızla tamamen aynı şekilde yapıyoruz. İlk olarak, tüm ağı diske (1) yazıyoruz ve ardından iki projeksiyonu (kötü amaçlı yazılım için projeksiyon ve burada kaynak olarak adlandırdığımız görüntüler için projeksiyon) diske (2) yazıyoruz. İmage\_network.py'yi, bu kitaptaki kötü amaçlı yazılım veri kümelerinin herhangi birindeki grafik varlıkları analiz etmek veya seçtiğiniz kötü amaçlı yazılım veri kümelerinden istihbarat çıkarmak için kullanabilirsiniz.veri kümesi dataset

Özet:

bipartite iki taraflı ağ

Bu bölümde, kendi kötü amaçlı yazılım veri kümelerinizde paylaşılan öznitelik analizi yapmak için gerekli araçları ve yöntemleri öğrendiniz. Özellikle, ağların, iki taraflı ağların ve iki taraflı ağ projeksiyonlarının kötü amaçlı yazılım örnekleri arasındaki sosyal bağlantıları belirlemeye nasıl yardımcı olabileceğini, ağ düzeninin neden ağ görselleştirme için merkezi olduğunu ve zorla yönlendirilmiş ağların nasıl çalıştığını öğrendiniz. Ayrıca, Python ve NetworkX gibi açık kaynaklı araçları kullanarak kötü amaçlı yazılım ağlarını nasıl oluşturacağınızı ve görselleştireceğinizi de öğrendiniz. 5. Bölümde, örnekler arasında paylaşılan kod ilişkilerine dayalı olarak kötü amaçlı ağların nasıl oluşturulacağını öğreneceksiniz.

UYGULAMASINI YAPAMADIM