

T.C
FIRAT ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

MALWARE ANALİZİ

HAZIRLAYAN
ÜMMÜ DERYA ÇELİK
180509041

DANIŞMAN
DOÇ.DR. FATİH ERTAM

ADLI BİLİŞİM MÜHENDİSLİĞİ
BİTİRME ÖDEVİ

HAZİRAN 2022
ELAZIĞ

ÖNSÖZ/TEŞEKKÜR

Çalışmalarım süresince her türlü yardım ve fedakarlığı yapan bilgisiyle bana yol gösteren danışman hocam Fatih ERTAM’a ilgi ve alakası için teşekkür ederim.

Ümmü Derya ÇELİK

HAZİRAN 2022, ELAZIĞ

ÖZET

Bu projede amaç günümüzde artan siber saldırıların en etkili kaynaklarından biri olan malware yani zararlı yazılımlar ve analiz yöntemleri hakkında bilgi vererek farkındalığı arttırmaktır. İçerik olarak malware türlerine, bulaşma yollarına, analiz tespit yöntemlerine, bulaşmayı önleme yollarına teorik olarak değinilip Yara ve Volatility araçları ile de uygulamalı olarak malware analizi yapılmıştır.

TABLÖLAR LİSTESİ

Tablo 1 İmza tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	18
Tablo 2 Davranış tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	18
Tablo 3 Model denetleme tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	19
Tablo 4 Derin öğrenme tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	19
Tablo 5 Bulut tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	20
Tablo 6 Mobil ve IoT tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları.....	20

ŞEKİLLER LİSTESİ

Şekil 1 Virüsün şekil değiştirerek yayılması.....	8
Şekil 2 Derleme ve tersine mühendislik algoritması.....	10
Şekil 3 Eşdeğeri ile kod değişimi örneği.....	11
Şekil 4 Komutların yer değiştirilmesi örneği.....	12
Şekil 5 kodların rastgele yer değiştirmesi örneği.....	12
Şekil 6 Orijinal kod bloğu ve Base64 kodlama kullanılarak abfuscate edilmiş yani formatı değiştirilmiş kod bloğu.....	13
Şekil 7 Malware tespit yaklaşımları akış şeması.....	16
Şekil 8 Yara kuralı tanımlama temel görünümü.....	24
Şekil 9 Text String Wide kullanım örneği	25
Şekil 10 Text String Fullword kullanım örneği.....	25
Şekil 11 Koşullu string örneği	25
Şekil 12 hexadecimal örneği.....	26
Şekil 13 Conditions örnek kullanımı.....	26
Şekil 14 Conditions:String sayısı örneği.....	27
Şekil 15 Conditions:Offset örneği.....	27
Şekil 16 Yara aracı genel görünümü.....	28
Şekil 17 Basit düzey yara kuralı görünümü.....	28
Şekil 18 Değerler listesi.....	28
Şekil 19 Kural.txt arama komutu.....	29
Şekil 20 Kural.txt -s arama komutu.....	29
Şekil 21 PE Studio üzerinde malware görünümü.....	29
Şekil 22 PE Studio String üzerinden url elde etme.....	30
Şekil 23 Url eklenen yara kuralı.....	30
Şekil 24 Malware tespiti için çalıştırılan komut.....	30
Şekil 25 Hex editörde malware yapısının görünümü.....	31
Şekil 26 Oluşturulan yara kuralına bulunan MZ'lerin eklenmesi	31
Şekil 27 Güncellenen yara kuralı ile tarama yapma ve MZ değerinin görüntülenmesi.....	31
Şekil 28 Cuckoo modülünün kullanım şekli.....	32
Şekil 29 Magic modülünün kullanım şekli.....	32
Şekil 30 Hash modülünün kullanım şekli.....	33
Şekil 31 Dotnet modülünün kullanım şekli.....	33
Şekil 32 YaraGen yardımcı aracın görünümü.....	33

Şekil 33 Yabin yardımcı aracın görünümü.....	34
Şekil 34 Hybrid Analysis yardımcı aracın görünümü.....	34
Şekil 35 Binalyze yardımcı aracın görünümü.....	35
Şekil 36 Volatility imageinfo komutu.....	36
Şekil 37 Volatility imageinfo komutu çıktısı.....	36
Şekil 38 Volatility pstree parametre.....	36
Şekil 39 Volatility pstree parametre çıktısı.....	37
Şekil 40 Volatility connscan parametresi.....	37
Şekil 41 Volatility connscan parametre çıktısı.....	38
Şekil 42 Volatility dlllist parametresi.....	38
Şekil 43 Volatility malfind parametresi.....	38
Şekil 44 Volatility malfind parametre çıktısı.....	39
Şekil 45 Dump edilen dll dosyalarının derya klasöründeki görünümü.....	39
Şekil 46 Volatility sha256sum komutu.....	40
Şekil 47 VirusTotal’de taratılan Hashin görünümü.....	40
Şekil 48 Volatility modscan parametresi.....	40
Şekil 49 Volatility modscan parametre çıktısı.....	41
Şekil 50 Volatility moddump parametresi.....	41
Şekil 51 dump edilen sürücünün dosya içindeki görünümü.....	41
Şekil 52 Sürücünün VirusTotal taraması görünümü.....	42

İÇİNDEKİLER

MALWARE	8
MALWARE VE TÜRLERİ	8
MALWARE GİZLENME TEKNİKLERİ	10
MALWARE ANALİZİ	13
MALWARE ANALİZİ TÜRLERİ.....	13
STATİK ANALİZ.....	13
DİNAMİK ANALİZ	14
STATİK İLE DİNAMİK ANALİZ KARŞILAŞTIRMASI	14
MALWARE TESPİTİ VE İZLENME YÖNTEMLERİ.....	15
İmza Tabanlı Tespit.....	16
Davranış Tabanlı Tespit	17
Model Denetleme Tabanlı Tespit.....	18
Derin Öğrenme Tabanlı Tespit.....	18
Bulut Tabanlı Tespit	18
Mobil ve IoT Tabanlı Tespit	19
MALWARE TESPİT YAKLAŞIMLARININ DEĞERLENDİRİLMESİ	19
STATİK ANALİZDE KULLANILAN ARAÇLAR	21
DİNAMİK ANALİZDE KULLANILAN ARAÇLAR.....	22
UYGULAMALI MALWARE ANALİZİ.....	23
YARA İLE MALWARE ANALİZİ	24
VOLATİLİTY İLE MALWARE ANALİZİ	35
ZARARLI YAZILIMLARA KARŞI ALINABİLECEK ÖNLEMLER.....	42
MAKİNELERDE ALINABİLECEK ÖNLEMLER	42
AĞDA ALINABİLECEK ÖNLEMLER	43
GÜVENLİK CİHAZLARI İLE ALINABİLECEK ÖNLEMLER	45
KAYNAKÇA	46

MALWARE

Bölümde malware ve malware türlerinden, malware gizleme tekniklerinden ve şimdiye kadar gerçekleşmiş büyük çaplı malware saldırılarından bahsedilmiştir.

MALWARE VE TÜRLERİ

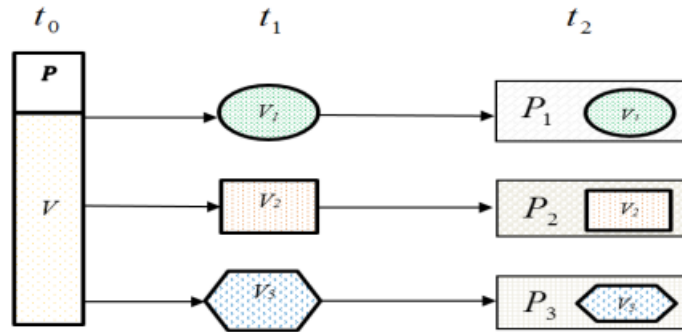
Malware yani zararlı yazılımlar bilgisayar ve internet kullanımının yaygınlaşmasıyla birlikte günümüzde çokça duyduğumuz bir kavram haline gelmiştir.

Malware, geniş çaplı ve küresel boyutta hedef odaklı olan cihazda bulunan ayarları değiştirebilmek, kontrol edebilmek, kişisel bilgilerinize ulaşmayı hedeflemek, cihazınızı saldırılara karşı tamamen açık hale getirmek gibi kötü amaçları olan zararlı yazılımlardır.

Yaygın olarak bilinen malware türleri ve özellikleri aşağıdaki gibidir;

➤ VİRÜS

En yaygın bilinen zararlı yazılım türüdür. Bilgisayardan bilgisayara ya da tüm ağ üzerinden geçirilebilen bir yazılımdır. Bulunduğu sistemi yavaşlatır. Sisteme girip etkinleştirilmek için bir programa veya uygulamaya ihtiyaç duymaktadır. Başlıca yayılma ortamları e-posta eklentileri, sosyal medya ortamları, ücretsiz yazılım platformları, USB belleklerdir.



Şekil 1 Virüsün şekil değiştirerek yayılması

Şekil 1’ de virüsün şekil değiştirerek çoğalmasını göstermektedir. Böylece virüs imzası değişecek ve tespit edilmesi zorlaşacaktır.

Virüsler yazılma amaçlarına göre yedi farklı kategoriye ayrılabilir;

- 1) **Önyükleme sektör virüsleri (boot sector viruses):** Bu tür virüsler floppy disklerin(disket) ve hard disklerin önyükleme sektörlerine yerleşmektedir. Bilgisayarın açılmasıyla aktifleşirler ve sistemden temizlemek oldukça zordur.
- 2) **Makro virüsleri (macro viruses):** Microsoft belgelerini, excel ve Word gibi, kullanarak yayılan virüs türüdür.

- 3) **Yerleşik virüsler (resident viruses):** Bu tür virüsler öncelik olarak kendini sisteme entegre eder ve daha sonra çalışmaya başlar. Yani orijinal kod silinse bile yerleşik kod çalışmaya devam eder. Tespit edilmeleri ve temizlenmeleri oldukça zordur.
- 4) **Dosya bulaşma virüsleri (file infector viruses):** Yerleştiği dosyanın kodlarında değişiklik yaparak veya zararlı kodları dosya kodlarına entegre ederek başka dosyalara bulaşma yoluyla yayılırlar.
- 5) **Çok biçimli virüsler (polymorphic viruses):** Her çalıştırıldığında kendini değiştiren virüs türüdür. Tespiti oldukça zordur.
- 6) **Tarayıcı ele geçiren virüsler (browser hijacker):** Hedef web sitesinin popülerliğini ve ziyaretçi sayısını arttırmak amacıyla web tarayıcılara yerleşerek başlangıç ve arama sayfalarını değiştiren virüs türüdür.
- 7) **Çok bölümlü virüsler (multipartite virus):** Bu tür virüsler ilk olarak program dosyalarına daha sonra program açıldığında da önyükleme (boot) kayıtlarına bulaşır. Program yeniden başlatıldığında hafızaya yerleşerek diğer programlara bulaşır. Hem önyükleme sektörüne hem de yürütülebilir dosyalara aynı anda bulaşabilmektedir. Bundan dolayı diğer virüslere göre bulaştığı sisteme vereceği zarar daha fazladır.

➤ **SOLUCANLAR (Worms)**

Kendini çoğaltmak ve yaymak için bilgisayar ağını kullanan bir malware türüdür. Etkinleştirilmek için virüslerde olduğu gibi insan eylemine yani programın çalıştırılmasına ihtiyaç duymazlar. her hangi bir güvenlik açığından faydalanıp girdiği sisteme arka kapılar açarak sistemi ve cihazı güvensiz hale getirir. Yetkisiz erişimler sıkça görülür.

➤ **ARKA KAPI (Back Door)**

Kimlik doğrulama prosedürlerini, internet gibi bir ağa yapılan bağlantı üzerinden atlatmak için kullanılır. Bu sistem gelecekte hedef sistem üzerinde fark edilmeden çalışarak daha fazla arka kapının oluşmasına ve yetkisiz erişimlere neden olur. Genellikle Truva atları ve solucanlar kullanılarak sisteme yerleştirilir.

➤ **TRUVA ATLARI (Trojan Horse)**

Normal programlar içine gömülüp sisteme kendisini yararlı, kullanışlı gibi gösterip kurbanı kandırarak kendisini kurdurtmayı hedefleyen programlardır. Truva atları genellikle sosyal mühendislik teknikleri ile yayılır. Günümüzde arka kapılar gibi davranarak saldırının yetkisiz erişim almasına neden olur.

➤ **KÖK KULLANICI TAKIMI (RootKit)**

RootKit yazılım paketleri, işletim sistemi üzerinde değişiklikler yaparak kötü amaçlı yazılımların veya işlemlerin kullanıcıdan gizlenmesini sağlar. Yapılan işlemlerin işlem listesinde görünmesini engelleyebilir ve dosyalarını işletim sisteminden gizleyebilir. Başka zararlı yazılımlarla birlikte çalışır.

➤ **FİDYE YAZILIMI (Ransomware)**

Bu tür zararlı yazılımlar bulaştığı sistemdeki verileri şifreler, verilerini tekrar görüntüleyebilmek için kullanıcının fidye ödemesinin gerektiği sistemlerdir. Fidyeye virüsleri, geçerli bir dosya olarak kendini gizleyen Truva atı tekniğini kullanır.

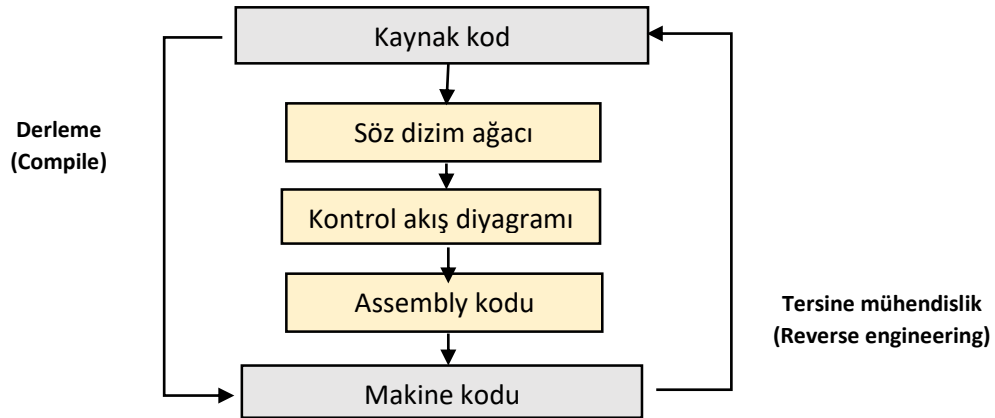
➤ **CASUS YAZILIM (Spyware)**

Casus yazılımlar kullanıcı bilgilerini ve genellikle kredi kartı bilgilerini ve kullanıcı alışkanlıklarını kullanıcının bilgisi olmadan toplayarak üçüncü taraf kişilere aktarılmasını takip eden saldırılardır. Virüslerden ve solucanlardan farklı olarak kendini çoğaltmayı hedeflemez sadece sistemde gizli kalarak istediği bilgilere ulaşmayı hedefler.

Bazı ticari firmalar bile internet üzerinden casus yazılımları yayarak kullanıcı alışkanlıklarını saptamayı hedefleyebilmektedir. Korunmak içinse antivirüs programları yeterli kalmayacaktır. Bunun yerine sadece casus yazılımlara odaklı üretilmiş antispyware yani casussavar yazılım ürünlerini kullanmak gerekmektedir.

MALWARE GİZLENME TEKNİKLERİ

Tersine mühendislik (reverse engineering) yöntemlerinden tamamen olmasa da belli bir derece de korunabilmek ve üçüncü şahıslar tarafından görülmesini veya anlaşılmasını engellemek için yazılımcılar programların kaynak kodunu kod karmaşılaştırma gibi işlemlerle gizli tutmayı amaçlamaktadır. Buna rağmen tersine mühendislik yöntemiyle kodun bir kısmı ya da tamamına erişilebilmektedir.



Şekil 2 Derleme ve tersine mühendislik algoritması

Obfuscation tekniđi yani karmaşıklştırma, gizleme işleminin, programcılar tarafından yazılım üzerinde kodun kurcalanmasını önleme, kodun mülkiyet hakkını koruma ve yazılımın güvenliğini artırma amaçları ile kullanılırken zararlı yazılım geliştiricileri tarafından bulaştıkları sistemde fark edilmeden daha uzun kalabilmek için kullanılmaktadır. Obfuscation tekniğinin çalışma prensibi kaynak kodun bir kısmını veya tamamını şifreleyerek, yararlı sınıfları ve değişken adlarını anlamsız rastgele etiketlerle yeniden adlandırarak veya uygulamada gerekli olmayan kod bütünleri ekleyerek kodu karmaşıklştırıp gizlemektir.

Yaygın olarak kullanılan Obfuscation teknikleri, register ataması, komut yer değiştirme, kod ekleme, kod aktarımı, kod entegrasyonu, altıyordam sıralaması ve kodlama teknikleridir.

Register Ataması: Zararlı yazılımların kullandığı registerlerin ya da değişkenlerin bir kısmının veya tamamının kullanılmayan register ve değişkenlerle değiştirilme yöntemidir. Programın çalışma davranışı yapılan değişikliklere rağmen aynı kalmaktadır.

Kod Ekleme: Gerekli olmayan kodları veya kod bloklarını programın çalışmasını etkilemeyecek şekilde ekleyip program dizilimini değiştirerek statik analizi ve program imzası değiştiđi için antivirüs tarayıcılarını atlatmayı hedefleyen zararlı yazılım gizleme yöntemidir.

Komut Yer Deđiştirme: Kod bloklarının eşdeğeri ile yer değiştirilmesi yöntemidir. Yapılan değişikliklere rağmen programın çalışmasında bir değişiklik olmamaktadır.

Kod Blođu	Yeni Kod Blođu
<pre>Static void Main(string[] args){ int x = 50; int y = 60; int carpim = x*y; }</pre>	<pre>Static void Main(string[] args){ int x = 100/2; int y = 120/2; int carpim = ((2*x) * (2*y)); }</pre>

Şekil 3 Eşdeğeri ile kod deđişimi örneđi

Kod Aktarımı: Komutların programın işleyişini etkilemeyecek şekilde değiştirilmesi yöntemidir. Uygulanması zor fakat uygulanabildiği takdirde zararlı yazılımın tespiti bir hayli zorlaşmaktadır.

<u>Kod Bloğu</u>	<u>Yeni Kod Bloğu</u>
<pre>x = 2; y = 40; carpım = x * y;</pre>	<pre>x = 2; y = 40; carpım = y * x;</pre>

Şekil 4 Komutların yer değiştirilmesi örneği

Kod Entegrasyonu: Program kodlarının çalışma işleyişini çok etkilemeyecek şekilde zararlı kod bloğuyla birleştirilerek kullanılan gizleme yöntemidir.

Altyordam Sıralaması: Program alt kodlarının sıralamasının değiştiren yöntemdir. Yapılan kod dizilim değişikliklerine rağmen programın çalışma işleyişi benzer kalmaktadır.

Kod bloğu

Yeni kod bloğu

```
protected void onCreate(Bundle savedInstanceState) {
    number1 = (EditText) findViewById(R.id.number1);
    number2 = (EditText) findViewById(R.id.number2);
    resultText = (TextView) findViewById(R.id.textView);

    public void sum (View view){

        if(number1.getText().toString().matches("") ||
        number2.getText().toString().matches("")) {
            resultText.setText("Değer Giriniz");
        } else {
            int a = Integer.parseInt(number1.getText().toString());
            int b = Integer.parseInt(number2.getText().toString());
            int resultInteger = a + b;

            resultText.setText("Result: " + resultInteger);
        }
    }
}
```

```
protected void onCreate(Bundle savedInstanceState) {
    number2 = (EditText) findViewById(R.id.number2);
    number1 = (EditText) findViewById(R.id.number1);
    resultText = (TextView) findViewById(R.id.textView);

    public void sum (View view){

        if(number1.getText().toString().matches("") ||
        number2.getText().toString().matches("")) {
            resultText.setText("Değer Giriniz");
        } else {
            int b = Integer.parseInt(number2.getText().toString());
            int a = Integer.parseInt(number1.getText().toString());
            int resultInteger = a + b;

            resultText.setText("Result: " + resultInteger);
        }
    }
}
```

Şekil 5 kodların rastgele yer değiştirmesi örneği

Kodlama(Encoding): Base64 gibi kodlama teknikleri kullanılarak kodun değiştirilmesi yöntemidir.

Orijinal kod bloğu	Obfuscate edilmiş kod bloğu
<pre>int main() { std::cout << "Hello World!"; return 0; }</pre>	<pre>aW50IG1haW4oKQew== IHsK c3RkOjpb3V0IDw8ICJIZWxsbyBxb3JsZCEiOw== cmV0dXJuIDA7 fQo=</pre>

Şekil 6 Orijinal kod bloğu ve Base64 kodlama kullanılarak obfuscate edilmiş yani formatı değiştirilmiş kod bloğu

MALWARE ANALİZİ

Bu bölümde malware analizine ve malware analizi türlerine değinilmiştir.

Malware analizi, zararlı yazılımların çeşitli yazılımlar ve teknikler kullanılarak tespit edilmesi ve vereceği zararı en kısa sürede azaltmayı hedefleyen çalışmalara verilen addır.

Malware analizi yapan analistler bu durumları tespit etmeyi amaçlarlar:

- Sistemde açığa neden olan ve saldırının başladığı yeri;
- Verilerden hangilerinin zarar gördüğü;
- Saldırının çapını yani hangi makinelerin zarar gördüğünü ve hangi uygulamaların, programların etkilendiğini;
- Zarar gören program ve makinelerin nasıl eski haline döndürüleceğini;
- Yapılabilecek olağan saldırıları.

Malware analizi iki ana gruba ayrılır: Statik analiz ve dinamik analiz.

MALWARE ANALİZİ TÜRLERİ

STATİK ANALİZ

Statik analiz, temel statik analiz ve ileri düzey statik analiz olarak iki kategoride incelenebilir.

TEMEL STATİK ANALİZ

Statik analizde zararlı yazılım bulaşmış program çalıştırılmadan inceleme yapıldığı için sistemde herhangi bir zarara neden olmamaktadır.

Temel statik analiz zararlı yazılımın genel özellikleri hakkında bilgi toplamak için yapılmaktadır. Bu aşamada antivirüs tarayıcıları, virustotal gibi, ve statik analiz araçları kullanılmaktadır. Bu araçlarla yazılımda kullanılan parametreler, hash değeri ve hash değerinin daha önceki malware hashleri ile kıyaslanması, dosya başlıkları gibi bilgilere ulaşılabilmektedir.

Zararlı yazılımları tanımlamak ve etiketlemek için yaygın olarak kullanılanlar MD5 ve SHA-256 hash imzalarıdır.

İLERİ DÜZEY STATİK ANALİZ

İleri düzey statik analizde ise zararlı yazılımlar, programlar kullanılarak detaylı olarak incelenmektedir. İleri düzey statik analiz kodların tersine mühendislik yöntemleriyle çevrilip incelenmesini içermektedir. İlk olarak makine dili kodları assembly kodlarına daha sonra üst seviyeli dillere dönüştürerek yani decompile (tersine derleme) ederek elde edilen kodlar incelenmektedir. Tersine derleme sırasında üst seviyeli dillerde(C,C#,Java) kaynak kodu elde etmeye çalışırken kayıplar yaşanmaktadır bu yüzden assembly seviyesindeki diller analiz için tercih edilmektedir.

DİNAMİK ANALİZ

Dinamik analiz iki temel kategoride temel dinamik analiz ve ileri düzey dinamik analiz olarak incelenebilir.

TEMEL DİNAMİK ANALİZ

İzleme programları kullanılarak zararlı yazılımların davranışlarının incelendiği analiz türüdür. Yaygın olarak API Monitor, Wireshark, Sandbox, Process Monitor gibi araçlar kullanılır.

İLERİ DÜZEY DİNAMİK ANALİZ

Analiz ortamında zararlı yazılım çalıştırılarak sistem üzerindeki etkilerinin incelendiği analiz türüdür. Zararlı yazılım kodlarında değişkenlerin, parametlerin ve hafıza alanlarının içeriğini görüntüleme ve değişiklik yapabilmek için hata ayıklama araçları (Debugger) kullanılır.

STATİK İLE DİNAMİK ANALİZ KARŞILAŞTIRMASI

Statik analiz ile programlar hakkında genel bir bilgi elde etmek ve daha önceden karşılaşılmış zararlı yazılımlarla kıyaslayarak analiz etmek kolaydır. Fakat gizlenme tekniği kullanılmış bir zararlı yazılımda analiz için dinamik analiz türü kullanılmalıdır çünkü statik analiz ile zararlı yazılımın tespiti bile oldukça zordur. Statik analizin avantajlarından ve dezavantajlarından bahsedecek olunursa;

Statik analizin;

Avantajları:

- ✓ Genel bir fikir sahibi olma fırsatı sunar.
- ✓ Zaman ve kaynak tüketiminde tasarruf sağlar.
- ✓ Tekrarlanabilir.
- ✓ Analiz edilen sistem zarar görmez.

Dezavantajları:

- Tersine mühendislik işlemleri önünde engel oluşturur.
- Gizlenme tekniklerine (Obfuscation) karşı savunmasızdır.
- Karmaşıktır.
- Karmaşık zararlı yazılım analizlerinde faydasızdır.

Dinamik analizin;

Avantajları;

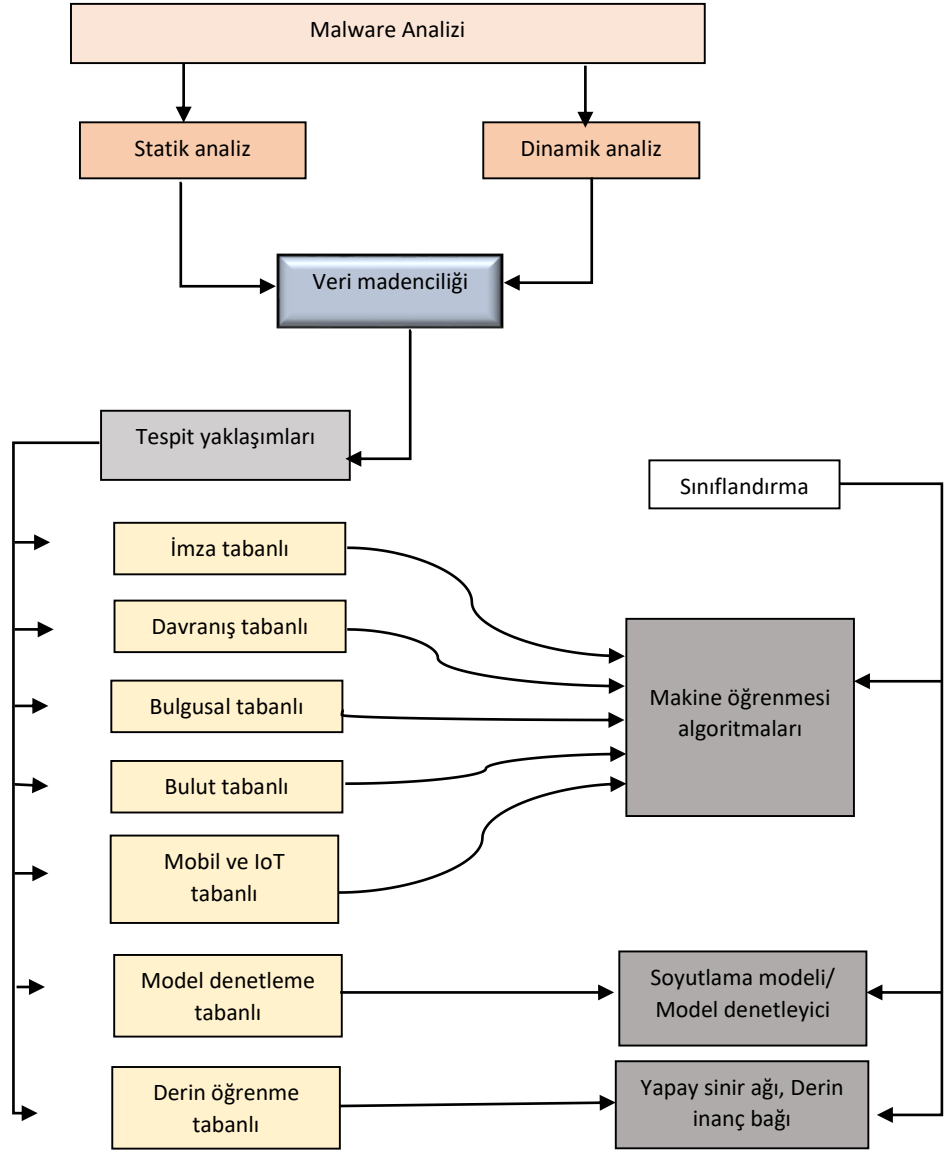
- ✓ Kesin sonuçlar üretir.
- ✓ Gerçek zamanlı davranış bilgileri elde edilir.
- ✓ Gizlenme tekniklerine (obfuscation) karşı etkilidir.
- ✓ Daha önce karşılaşılmamış yeni nesil zararlı yazılımları tespit edebilir.

Dezavantajları;

- Analiz ortamında çalıştırılırken farklı davranışlar gösterebilir.
- Analiz otomatikleştirildiğinde etkileşimli davranışlar gözden kaçırılabilir.
- Zaman ve kaynak tüketimi bir hayli fazladır.
- Sanal ortamlarda zararlı yazılımlar olduğundan farklı davranış tüm davranışlarını göstermeyebilirler.

MALWARE TESPİTİ VE İZLENME YÖNTEMLERİ

Zararlı yazılımlardaki karmaşıklık zaman ilerledikçe artmaya başlamıştır ve yeni nesil zararlı yazılımların tespiti ve analizinde zorluklar yaşanmaya başlanmıştır. Bu nedenlerle araştırmacılar, davranış, bulgusal/sezgisel, model denetleme tabanlı, bulut mobil cihazlar ve iot tabanlı gibi tespit yaklaşımları geliştirmişlerdir.



Şekil 7 Malware tespit yaklaşımları akış şeması

Günümüzde giderek karmaşılaşan zararlı yazılımların analizinde ve tespitinde veri madenciliğinden de yararlanılmaya başlanmıştır. Veri madenciliği önceden rastlanmamış değerler çıkartılarak anlamlandırılması işlemidir.

İmza Tabanlı Tespit

Program imzaları, programdaki belirli kod bölümlerinden program yapısı çıkarılarak ve belirli uzunlukta olan özel karakter dizilimlerdir. İmza her için benzersiz şekilde üretilir bu yüzden malware analizinde tespit amaçlı sıkça kullanılır.

Diğer tespit yaklaşımlarına göre daha hızlı ve düşük hata oranına sahiptir. Ancak yeni nesil zararlı yazılımlarda yani bilinmeyen imzalarda tespit aşamasında yetersiz kalmaktadır. Bu dezavantajın yanında aynı

aileye ait zararlı yazılımlar az bir değişiklik ile imza tabanlı tespit yaklaşımını kolayca atlatabilmektedir.

Bu yaklaşım iyi bilinen veya aynı aileye sahip zararlı yazılımlar için kullanılır. İmza çıkarma işlemi zaman alan zorlu bir süreç olmakla beraber imza mümkün olduğunca kısa ve tek imzayla zararlı yazılım tespit edilebilir ve imza paketleme aynı zamanda gizlenme tekniklerine karşı dayanıklı olmalıdır.

Geçmişte yapılan imza tabanlı tespit çalışmalarında kullanılan bazı özellik çıkarma yöntemleri ve amaçları aşağıdaki gibidir:

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Griffin vd.	Dizi imzaları, bulgusal tarama	Zararlı yazılımların farklı formlarını tespit
Tang vd.	Exploit tabanlı imzalar	Çok çeşitli solucanların tespiti
Borojerdi ve Abadi	Dizi kümeleme, benzer örüntüler	Çok biçimli zararlı yazılımları %90.83 ile tespit
Liu ve Sandhu	Kriptografik hash tabanlı imzalar	Donanım içine yerleştirilmiş zararlı yazılımların tespiti

Tablo 1 İmza tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

Davranış Tabanlı Tespit

Davranış tabanlı tespit yaklaşımı programın davranışlarına bakarak programın normal ya da zararlı yazılım olup olmadığını belirleyen bir yöntem kullanır. Çalışma şeklinde programın kodlarına bakılmadığı için programın kodlarına zararlı yazılım işlenirse de davranışı değişmeyeceği için birçok zararlı yazılım tespit edilebilmektedir. Bu yaklaşımda dezavantaj ise sanal makine de kurulan ve çalıştırılan programın tüm davranışlarını göstermemesi ve davranışlarının tamamıyla görülememesidir. Çalışma aşamaları ilk adım davranışların çıkarılması, ikinci aşama davranışların belirlenmesi ve programın özelliklerinin belirlenmesi, üçüncü aşama ise makine öğrenmesi algoritmaları kullanılarak programın zararlı ya da normal yazılım olarak belirlenmesidir.

Programın davranışları sistem çağrıları, registry ve ağ üzerinde meydana gelen değişiklikler kullanılarak belirlenmektedir. Sistem çağrılarının veya dosya registry işlemlerinin hangi sırada ve ne sıklıkla yapıldıkları incelenerek davranışlar oluşturulmaktadır. Bu davranışlar gruplandırılıp dizilimler oluşturularak özellikler elde edilir.

Davranış tabanlı tespit yaklaşımında dosya değişiklikleri incelenerek, kayıt defterinin anlık görüntülerini karşılaştırarak, ağ aktiviteleri ve sistem çağrıları izlenerek zararlı yazılım analizi yapılır.

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Park vd.	Yaygın davranışlar	Zararlı yazılımların varyasyonlarının tespiti
Shan ve wang	Davranış tabanlı kümeleme	Bilinmeyen zararlı yazılımların %71 ile tespiti
Wagener vd.	Sistem çağrıları	Yeni nesil zararlı yazılımlar ve farklı varyasyonlarının tespiti

Tablo 2 Davranış tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

Model Denetleme Tabanlı Tespit

Model tabanlı tespit yaklaşımı, zararlı olan yazılım ile normal yazılımın özellikleri belirlenerek belirli özelliği gösterecek şekilde doğrusal mantık formülleri kullanılarak kodlayan ve bu sonuçları daha önceden belirlenen niteliklerle karşılaştırarak yazılımın zararlı olup olmadığına karar veren yaklaşımdır.

Bu yaklaşım diğer yaklaşımlara göre paketleme ve gizleme tekniklerine karşı dirençlidir ve bazen yeni zararlı yazılımları tespit edebilir. Fakat bu yazılım genellikle program doğrulamak için kullanılmış olup zararlı yazılım tespiti için kullanılmamıştır. Çünkü karmaşık zararlı yazılımları tespit etmekte yetersiz kalmıştır. Bu yaklaşım başka yaklaşımlarla birlikte kullanıldığında performans artırıcı olarak etkin rol oynayabilir.

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Holzer vd.	Sonlu durum modeli	Benzer saldırıları kullanan zararlı yazılımları bulma
Kinder vd.	Açıklamalı kontrol akışı	Zararlı yazılımların farklı formatlarının tespiti
Battista vd.	Java bytecode, Payload	DroidKungFu ve opfake zararlı yazılım ailelerinin tespiti

Tablo 3 Model denetleme tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

Derin Öğrenme Tabanlı Tespit

Derin öğrenme, örneklerle öğrenen ve yapay sinir ağlarını miras alan makine öğrenmesinin alt dalıdır. Zararlı yazılım tespiti için henüz yeterince çalışma bulunmamaktadır. Derin öğrenme tabanlı tespit yaklaşımı yüksek performansla çalışmaktadır ve özellik alanını oldukça daraltmaktadır fakat gizli katman oluşturmak zaman alan bir süreç olmakla birlikte ilave katmanlar model performansını az çok etkilemektedir. Atlatma saldırılarına karşı dayanıklı değildir.

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Saxe ve Berlin	Bağlamsal bayt özellikleri, PE özellikleri	Zararlı yazılımların tespiti
Huang ve Stokes	API çağırısı, çok görevli öğrenme	Zararlı yazılımların tespiti

Tablo 4 Derin öğrenme tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

Bulut Tabanlı Tespit

Bulut tabanlı tespit yaklaşımı günümüzde sağladığı kullanışlı imkanlarla birlikte gelişimi oldukça artmaktadır ve farklı alanlarda

kullanılmaya başlanmıştır. Bu yaklaşım mobil cihazlar ve bilgisayarlar için tespit oranını artırır ve büyük veri tabanları ile geniş hesaplama alanları imkanı sunar. Kullanıcı dosyayı bulut ortamına yükler ve sistem yüklenen dosyanın zararlı yazılım olup olmadığı hakkında kullanıcıya bir rapor gönderir. Diğer tespit yaklaşımlarına göre daha fazla gecikmenin yaşanması bu yüzden de işlemci ile Iot ve mobil cihazlar özellikle optimize edilmelidir.

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Martignoni vd.	Sistem çağrıları	Zararlı yazılımların çoklu ortamda çalıştırılması
Yadav	Kümeleme ve sınıflandırma modülü kullanımı	Zararlı yazılımlarda yüksek tespit oranı

Tablo 5 Bulut tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

Mobil ve IoT Tabanlı Tespit

IoT mimarisi ağ kameraları ve sensörler gibi çok çeşitli internet bağlantılı akıllı cihazların kullanımı arttıkça saldırganlar tarafından da hedef haline gelme oranı gittikçe artmaktadır. Bu yaklaşım zararlı yazılım tespiti için makine öğrenmesi algoritmaları ve veri madenciliği algoritmalarını kullanmaktadır. Genellikle sistem çağrıları, API'ler, Android izinleri, bilgi akış ve kontrol akış yapıları özellikleri oluşturmakta kullanılan belirleyicilerdir. Bu yaklaşım zararlı yazılımları tespit etmede her ne kadar etkili olsa da yeni nesil zararlı yazılımları tespit etmede yetersiz kalmaktadır.

ÇALIŞMA	ÖZELLİK ÇIKARMA YÖNTEMİ	AMAÇ
Saracino	Çekirdek, kullanıcı özellikleri	Zararlı yazılımların tespiti
Azmoodeh vd.	Iot enerji tüketim desenleri	Zararlı yazılımların tespiti
Alazab vd.	API çağrısı, izin istekleri	Zararlı yazılımların tespiti

Tablo 6 Mobil ve IoT tabanlı tespit yaklaşımları çalışmaları yöntem ve amaçları

MALWARE TESPİT YAKLAŞIMLARININ DEĞERLENDİRİLMESİ

- İmza tabanlı tespit yaklaşımları:
 - ✓ Bilinen zararlı yazılımlar için etkilidir.
 - ✓ Aynı aileye ait zararlı yazılımları tespit etmede etkilidir.
 - ✓ Uzun zamandır kullanılmakta olan bir yaklaşımdır.
 - İmza çıkarma işlemi uzun sürdüğü için zaman kaybı artar
 - Yeni nesil zararlı yazılımları tespit etmede yetersiz kalmaktadır.
 - Gizlenme tekniklerine karşı savunmasız
- Davranış tabanlı tespit yaklaşımları:

- ✓ Zararlı yazılımın işlevlerini belirleyebilir.
- ✓ Yeni nesil zararlı yazılımları tespit edebilmektedir.
- ✓ Zararlı yazılımların farklı görünümelerini tespit edebilir.
- ✓ Gizlenme tekniklerine karşı güçlüdür.
- ✓ Zero day saldırılarını tespit edebilir.

- Zararlı yazılımların sadece sınırlı görünümünü sunar
- Bazı davranışlar zararlı yazılım ile normal yazılımların davranışlarında benzerlik göstermekte bu da karışıklığa neden olmaktadır.
- Zararlı yazılımların bütün davranışlarını belirlemek mümkün değildir.
- Yanlış tahmin oranı yüksektir.

- Bulgusal tabanlı tespit yaklaşımları:
 - ✓ Yeni nesil zararlı yazılımları tespit edebilir.
 - ✓ Hem statik hem dinamik özellikleri kullanabilmektedir.
 - ✓ Zero day saldırılarını tespit edebilir.

 - Çok sayıda kural mevcuttur ve eğitim aşaması uzun sürmektedir.
 - Gizlenme tekniklerine karşı savunmasız kalmaktadır.

- Model denetleme tabanlı tespit yaklaşımları:
 - ✓ Aynı aileye ait zararlı yazılımları tespit edebilir.
 - ✓ Gizlenme tekniklerine karşı güçlüdür.
 - ✓ Yeni nesil zararlı yazılımları tespit edebilmektedir.
 - ✓ Zero day saldırılarını tespit edebilir.

 - Karmaşık bir yapıya sahiptir.
 - Zararlı yazılımın sınırlı bir görünümünü sunar
 - Her zararlı yazılıma özgü model belirleme uzun sürmektedir.

- Derin öğrenme tabanlı tespit yaklaşımları:
 - ✓ Diğer yaklaşımlara göre daha güçlü ve etkilidir.
 - ✓ Özellik alanını oldukça daraltmaktadır.
 - ✓ Zero day saldırılarını tespit edebilir.

 - Gizlenme tekniklerine karşı hassastır.
 - Faaliyete geçirmek uzun sürmektedir.

- Bulut tabanlı yaklaşımlar:
 - ✓ Bilgisayar ve mobil cihazlar için tespit oranını arttırmaktadır.
 - ✓ Büyük veri tabanı ve geniş hesaplama alanı sağlamaktadır.

- ✓ Erişimi ve yönetilebilirliği kolaydır.
 - ✓ Zero day saldırılarını tespit edebilir.
 - Gerçek zamanlı izleme şansı sunmamaktadır.
 - Gizli verileri ifşa etme riski taşımaktadır.
 - İstemci ve sunucu arasında zaman gecikmesi yaşanmaktadır.
- Mobil ve IoT tabanlı tespit yaklaşımları:
- ✓ Yeni nesil zararlı yazılımları tespit edebilmektedir.
 - ✓ Hem statik hem dinamik özellikleri kullanabilmektedir.
 - ✓ Zero day saldırılarını tespit edebilir.
 - Gizlenme tekniklerine karşı savunmasızdır.
 - Çoğu uygulamayı gözden kaçırmaktadır.

STATİK ANALİZDE KULLANILAN ARAÇLAR

Statik analiz zararlı yazılım kodları çalıştırılmadan zararlı yazılım hakkında bilgi edinmeyi amaçlar. Statik analizde dosya adı, dosya boyutu, dosya tipi ve zararlı yazılım imzası gibi bilgileri edinmeyi hedefler.

PEiD (PE IDentifier)

Yürütülebilir dosyalarda bulunan şifreleyici ve paketleyicileri algılamak için kullanılan yani paketlenmiş dosyaları saptayan bir analiz aracıdır. Dosyaların türlerini, kodlandığı dilleri bile algılayabilir.

PE Explorer

Sistemdeki PE (EXE, DLL, ActiveX kontrolleri gibi) dosyalarını tarayıp dosyaların yapısını ve içeriğini görüntüleyebilme imkanı sunan ve bu dosyalar ile adını değiştirmek, kopyalamak, farklı bir klasöre taşımak gibi işlemleri yapabileceğimiz araçtır.

BinText

Bu araç sayesinde derlenmiş olan bir dosyanın içerisindeki string verisi okunabilir hale gelmektedir.

UPX

Kısaca dosya sıkıştırıcı programdır. Zararlı yazılım örnekleri sıkıştırılabilir. Aynı zamanda sıkıştırılan program dosyalarının decompile edilmesini önleme imkanı sunar.

Dependency Walker

Zararlı yazılım tarafından içe aktarılan DLL dosyaları ve metotları gösteren araçtır.

IDA Pro

Zararlı yazılım analistleri tarafından tersine mühendislik işlemleri için kullanılan karmaşık kodları yüksek seviyeli dillere çevirerek daha anlaşılabilir hale getiren disassembler aracıdır.

Hex Editors

İkili veri içeren dosyaların görüntülenmesine ve düzenlenmesine imkan sağlayan araçtır.

Hex-Rays Decompiler

Assembly kodunu anlaşılabilir C benzeri yüksek seviyeli dillere dönüştüren IDA Pro eklentisi olan araçtır.

CFF Explorer

Exe dosyasının genel bilgisini almak için kullanılan analiz aracıdır. PE dosyası üzerinde genel bilgileri, bölüm bilgilerini ve kullanılan API bilgileri gibi bilgileri elde etmemizi sağlar.

DİNAMİK ANALİZDE KULLANILAN ARAÇLAR

Zararlı yazılımın çalıştırılarak sistemde gösterdiği davranışları ve etkileşimleri izleyerek yapılan analiz türüdür. Temel analizde zararlı yazılım davranışları izleme programları kullanılarak, İleri düzey analizde ise debugger yani hata ayıklama araçlarının kullanılarak değişkenlerin parametrelerin ve hafıza alanlarının içerikleri görüntülenip eğer istenirse değiştirilerek yapılan analizdir.

Process Explorer

Sistemde olup biten gerçekleşen tüm olayları görüntüleyebilmemizi ve yönetmemizi sağlayan programdır. İzleme sırasında görüntülenen uygulamaları VirüsTotal' tarama imkanı sunar. Şüphelenilen bir uygulamayı onlarca antivirüs programı ile taramadan geçirme şansı verir.

Process Monitor

Gerçek zamanlı olarak sistemde gerçekleşen olayları görüntüleyen araçtır.

API Monitor

Sistemdeki uygulamalar ve hizmetler tarafından gerçekleştirilen API çağrılarını görüntüleyen ve denetleme imkanı sunan programdır.

Netcat

Sisteme gelen ve giden ağ bağlantılarını görüntüleme imkanı veren araçtır. TCP ve UDP iletişim kurallarını kullanarak ağ bağlantılarını görüntüler. Diğer programlar tarafından kolayca kullanılabilen ağ hata ayıklama ve araştırma arkauc aygıtıdır. Port tarama ve dinleme gibi özelliklere sahiptir ve arka planda akıdaymışçasına çalışır.

Wireshark

Ağ takip aracıdır. Network trafiğini uygulamanın kurulu olduğu bilgisayar üzerinden anlık olarak izler.

Capture BAT

Bu araç ile sistem durumunu görüntüleyebiliriz. Capture Bat tam olarak malware'ın ne yapmaya çalıştığını belirleyebilmek için malware'ın sistemde yaptığı değişiklikleri izler. Sadece kötü amaçlı yazılımın sistem üzerindeki davranışlarını izleyerek analiz eden davranışsal bir analiz aracıdır.

WinDBG

Kullanıcı ve çekirdek modunda çalışabilen bir hata ayıklama programıdır. Zararlı yazılım çalışırken alınmış bir dump üzerinde analiz fırsatı sunar.

BurpSuite

Web uygulamalarında zafiyetleri saptayabilme imkanı veren programdır.

Sandboxes

Programların üzerinde çalıştıkları sistemi etkilemeden yürütülebilmelerini sağlayan yalıtılmış bir test ortamı aracıdır. Sandbox sistemler aynı zamanda program kodunun çalışma mekanizmasını inceleyerek zararlı yazılım tehditlerini analiz eder ve sisteme zarar verecek herhangi bir olay olmadan önlem alır.

UYGULAMALI MALWARE ANALİZİ

Bu bölümde uygulamalı olarak malware analizinin aşamaları konu alınmıştır.

YARA İLE MALWARE ANALİZİ

Yara aracı Virüstopal firması tarafından geliştirilen malware türlerini tanımlamaya ve sınıflandırmaya yardımcı olan bir araçtır. Yara aracı metinsel ve binary patternlere dayalı malware aile kuralları kişisel olarak oluşturma imkanı sunar.

Yara kuralları bir text içerisinde string aramak gibi basit kuralları kapsadığı gibi belirli bir sanal bellek üzerinde arama yapma gibi fonksiyonları da bulunmaktadır. Yara, Malware analiz sürecini kolaylaştırıp tespit etmemizi sağlar. Aynı zamanda tespit işleminden sonra elde edilen verileri kategorize eder. Linux, Windows ve Mac OS X platformlarında çalışmaktadır.

Günümüzde imza tabanlı koruma yeterli gelmemektedir. Bypass işlemi imza tabanlı koruma sistemini kolayca atlatabilmektedir. Bu bypass işlemi fark edebilmek için yara aracı kullanılmaktadır.

Yara Kural Tanımlama

Temel olarak yara kuralları aşağıdaki şekilde görüldüğü gibi meta, string, condition kısımlarından oluşmaktadır.

```
rule Kural_ismi
{
  meta:
    author(Yazar) = "kullanıcı"
    description(Tanım) = "Yara kuralları öğrenme"

  strings:
    $a = "zararlı yazılım" wide ascii nocase
    $b = "yara kuralları" wide ascii nocase

  condition($art):
    $a or $b
}
```

Şekil 8 Yara kuralı tanımlama temel görünümü

○ Strings

Yara içerisinde üç adet strings türü bulunmaktadır. Bunlar Text,Regex ve Hexadecimal Strings olarak kategorize edilir.

- Text Strings:

Düz yazı şeklinde yazılan basit bir ASCII düzenidir. Burada aratmak istediğimiz anahtar kelime büyük ve küçük harfe duyarlı olacaktır.

#Büyük/Küçük Harfe Duyarlı Dizeler:

Deneme= \$text_case_example = "Metin"

#Büyük/Küçük Harfe Duyarsız Dizeler:

Deneme= \$text_nocase_example = "Metin" nocase

- **Text Strings:Wide**

Karakterin başında 2 bayt ile kodlanmış dize veya dizeleri bulmak için kullanılabilir. Çoğunlukla çalıştırılabilir Binary dosyalar üzerinde kullanılır.

```
rule OrnekWide
{
  strings:
    $a = "Dekor" wide nocase

  condition:
    $a
}
```

Şekil 9 Text String Wide kullanım örneği

Wide parametresi ile bu şekilde arama yapıldığında 'Dekor' kelimesini iki bayt ile kodlanmış halde ve verilen nocase parametresi ile büyük küçük harfe duyarlı olarak arar.

- **Text Strings:Fullword**

Fullword olarak tanımlama yaptığımızda kural içerisinde ki metni direk aratıp onu bulmaya çalışır.

```
rule FullWord
{
  strings:
    $a = "dekor" fullword
  condition:
    $a
}

www.dekor.com -> Yakalar.
www.dekor.blogspot.com -> Yakalayamaz.
```

Şekil 10 Text String Fullword kullanım örneği

- **Text Strings:Koşullu Strings**

String araması yaparken koşul koyarak arama gerçekleştirebiliriz. Koşulumuz Hexadecimal şekilde olacaktır.

```
rule StringKosullu
{
  strings:
    $a = {AA 30 (DB BC | C2) AA }
  condition:
    $a
}

Böyle bir koşul tanımlanırsa;
>>AA 30 DB BC AA
>>AA 30 C2 AA Değerlerini yakalar.
```

Şekil 11 Koşullu string örneği

- **Text Strings:Hexadecimal String**

Çıktı üzerinde hex karakterle eşleşen değerler bu şekilde adlandırılır.

Wildcard karakter:

"?" ile gösterilen bu kalıp bazı baytların bilinmediğini ve herhangi bir yapı ile eşleşmesi gerektiğini belirtir.

Örnek vermek gerekirse;
\$example_hex = (A1 A2 ? A8)

○ Jumps

Kalıp uzunluğu değiştiğinde de kuralın çalışması istenilirse aşağıdaki örnek şeklinde atlatılabilir.

Örnek:

\$example_jump = (B1 B2[6-8] 24):

Bu işlem ile 6 bayttan 8 bayta kadar olan herhangi bir dizinin olduğunu belirtir.

Wildcard ve jumps kavramlarından sonra hexadecimal üzerinde bir örneğe değinilebilir.

```
rule OrnekHex
{
  strings:
    $strings_hex = {AA 14 ?? 6D B? BF}
  condition:
    $strings_hex
}
```

Şekil 12 hexadecimal örneği

Örnek üzerinde "??" koyulan kısma 01 ve AA arasında bulunan sayısal veya alfabetik değer eşleşecektir.

○ Conditions

Yara kurallarında üzerinde bulunan durum kısmıdır. Kısaca koşullar olarak adlandırılabilir. Koşul kümesi bool ifadeleri çalıştırır.

```
rule KuralAdı
{
  meta:
    author = "Kullanıcı"
    description = "Tanım"

  strings:
    $a= "zararlı yazılım" wide ascii nocase
    $a= "yara kuralları" wide ascii nocase

  condition:
    $d or $e
}
```

Şekil 13 Conditions örnek kullanımı

Örnek üzerinden bakılırsa satır sonundaki \$d veya \$e parametresi "true" ya da "false" olmasını belirler.

- **CONDITION:String Sayısı**

Condition üzerinde String sayısı belirterek yakalama işlemi gerçekleştirilebilir.

```
$string_test1 = k
$string_test2 = d

$string_test1 = 2 and $string_test2 = <10

Yakalar : ->>>> kkkkkkkk,kkkkkk,kkkkkkkkk
Yakalayamaz : ->>>>> kkkkkkkkkk,ddddddddddddddddddddddk
```

Şekil 14 Conditions:String sayısı örneği

- **Conditions Seçenekleri : Offset**

String Offset: Belirtilen string'in dosya içerisinde belirli offset ya da memory üzerinde belirli bir yerde olup olmadığını öğrenme imkanı sunar.

-"\$k at 100 and \$d at 500;" Örneğinde \$k isimli string'in process memory space içerisinde sanal adresi 100 ve \$d'nin bir dosya üzerinde 500 offset'i üzerinde olup olmayacağını belirtir.

Bu iki ihtimalinde gerçekleşmesi gerekmektedir. Ayrıca Offset'ler decimal olarak tanımlanır.

```
rule Offset
{
  string:
    $k = "Deneme"
    $d = "rule"

  condition:
    $k at 0x64 and $d at 0xC8
}
```

Şekil 15 Conditions:Offset örneği

Burada k ve d değişkenlerine bir değer atandı. Koşullar kısmına k değişkenini "0x64" offsetinde ve d değişkenini ise "0xC8" offsetinde aranır.

- **Condition Seçenekleri : Filesize**

Dosya boyutunu kontrol etmek için "filesize> 100kb" şeklinde bir dize girilebilir.

Yara kurallarından bahsedildiğine göre Yara aracını indirip analiz aşamasına geçilebilir.

```
YARA 4.2.0, the pattern matching swiss army knife.
Usage: yara [OPTION]... [NAMESPACE:]RULES_FILE... FILE | DIR | PID

Mandatory arguments to long options are mandatory for short options too.

  --atom-quality-table=FILE      path to a file with the atom quality table
  -C, --compiled-rules           load compiled rules
  -c, --count                    print only number of matches
  -d, --define=VAR=VALUE         define external variable
  --fail-on-warnings             fail on warnings
  -f, --fast-scan                fast matching mode
  -h, --help                     show this help and exit
  -i, --identifier=IDENTIFIER    print only rules named IDENTIFIER
  --max-process-memory-chunk=NUMBER set maximum chunk size while reading process memory (default=10000)
  -l, --max-rules=NUMBER         abort scanning after matching a NUMBER of rules
  --max-strings-per-rule=NUMBER set maximum number of strings per rule (default=10000)
  -x, --module-data=MODULE=FILE pass FILE's content as extra data to MODULE
  -n, --negate                   print only not satisfied rules (negate)
  -N, --no-follow-symlinks       do not follow symlinks when scanning
  -w, --no-warnings              disable warnings
  -m, --print-meta               print metadata
  -D, --print-module-data        print module data
```

Şekil 16 Yara aracı genel görünümü

Yara aracı kurulum gerektirmeyen ve komut istemi üzerinden çalışmaktadır. "yara64.exe --help" komutu çalıştırıldığında araç ile ilgili bir kullanım kılavuzuna erişilebilir.

Yara aracını çalıştırdıktan sonra not defteri üzerinde basit bir yara kuralı tanımlayalım. Burada yazılacak kuralın yara aracı ile aynı konumda olmasına dikkat edilmelidir.

```
*Kural.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
rule deneme
{
  strings:
    $a = "dekor"
    $b = "deneMe"
  condition:
    $a or $b
}
```

Şekil 17 Basit düzey yara kuralı görünümü

\$a ve \$b değerlerine bulmak istenilen değer girilir.

Condition kısmında \$a or \$b kısmında ikisinden biri geçtiğinde kuralın eşleşmesi sağlanır.

```
Not.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
dekor
deneMe
asd
123
kural
örnek
deneme
roket
winrar
Hack
```

Şekil 18 Değerler listesi

İkinci adımda ise belirtilen değerleri bulabilmesi için bir .txt dosyası daha oluşturulur. İçerisine rastgele ve belirtilen metinler olmak üzere değerler eklenir.

```
C:\Users\...\Desktop\DenemeYara>yara64 Kural.txt .  
deneme .\Kural.txt  
deneme .\Not.txt
```

Şekil 19 Kural.txt arama komutu

txt dosyasının ardından "yara64 kural.txt ."komutu ile arama gerçekleştirilir. Bu arama sonrasında eşleştirilen dosya bilgisine erişilir.

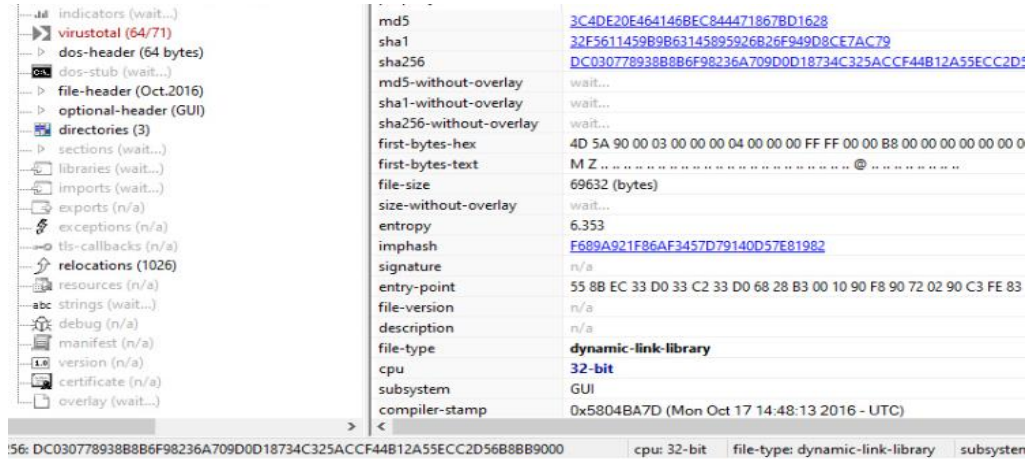
```
C:\Users\...\Desktop\DenemeYara>yara64 Kural.txt . -s  
deneme .\Kural.txt  
0x21:$a: dekor  
0x30:$b: deneMe  
deneme .\Not.txt  
0x0:$a: dekor  
0x7:$b: deneMe
```

Şekil 20 Kural.txt -s arama komutu

İçerisinde eşleşen bilgileri görüntülemek için yapılan aramanın sonuna "-s" parametresini ilave edilerek offset değerleri de görüntülenebilir.

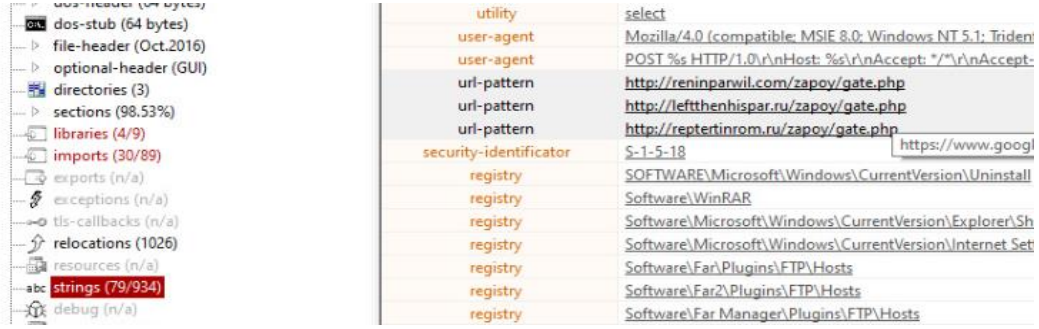
Sıradaki aşamada ise tarayıcı üzerinden deneme için bir zararlı yazılım bulup bu yazılımı "PE Studio" aracı ile dosya yapısında ki strings baz alınarak engellenmeye çalışılacak.

"https://www.winitor.com/tools/pestudio/current/pestudio.zip" bağlantısı üzerinden "PE Studio" aracı indirilir.



Şekil 21 PE Studio üzerinde malware görünümü

Oluşturulan zararlı yazılımı "PE Studio" aracının kısa yolunun üzerine koyulduğunda malware hakkında detaylı bilgi sahibi olunabilir.



Şekil 22 PE Studio String üzerinden url elde etme

PE Studio içerisinde strings kısmından URL elde edilmeye çalışılır. Şekil22’ malware içerebilecek olan url’ler belirlenir.

```
rule deneme
{
strings:
    $a = "http://reninparw11.com/zapoy/gate.php"
    $b = "http://leftthenhispar.ru/zapoy/gate.php"
    $c = "http://reptertinrom.ru/zapoy/gate.php"
condition:
    $a or $b or $c
}
```

Şekil 23 Url eklenen yara kuralı

Zararlı yazılımın bağlantı adresini de bulduktan sonra hemen bir yara kuralı oluşturulur. Oluşturulan bu yara kuralında \$a,b,c değerlerine PE studio aracılığıyla strings kısmında bulunan url’ler eklenir.

-Oluşturulan bu kuralı "Yara" aracının bulunduğu dizine aktarılır. "-r" parametresi kullanılarak istenilen koşullar sağlandığı takdirde işlemin tekrar etmesi istenir.

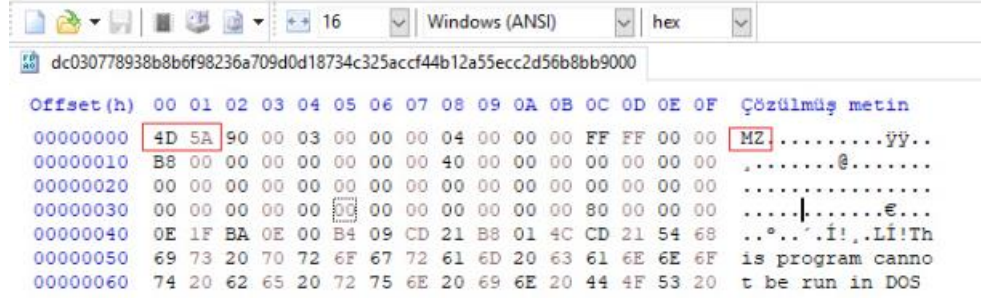
```
C:\Users\ \Downloads>yara64.exe -s -r kural.yara C:\Users\ \OneDrive\Desktop\dc030778938b8t
25accf44b12a55ecc2d56b8bb9000
deneme C:\Users\ \OneDrive\Desktop\dc030778938b8b6f98236a709d0d18734c325accf44b12a55ecc2d56b8b
0xce3a:$a: http://reninparwil.com/zapoy/gate.php
0xce60:$b: http://leftthenhispar.ru/zapoy/gate.php
0xce88:$c: http://reptertinrom.ru/zapoy/gate.php
```

Şekil 24 Malware tespiti için çalıştırılan komut

Yara aracı daha önceki örnekte olduğu gibi kural ile birlikte çalıştırılır. Çalıştırdıktan sonra başarılı bir şekilde zararlı yazılımı tespit ettiği görülmektedir.

Tespit işleminin ardından MZ yapısını kontrol etmekte fayda var. Bu işlem için "https://mh-nexus.de/en/hxd/" bağlantısı üzerinden Hex Editör aracı indirilir. MZ olarak adlandırılan yapı Hex Editör ile açılabilen iki baytlık yapılardır.

İndirilen zararlı yazılım çalıştırılabilir olmasına rağmen o şekilde gözüküyor. Bu yapı PE olarak adlandırılır.



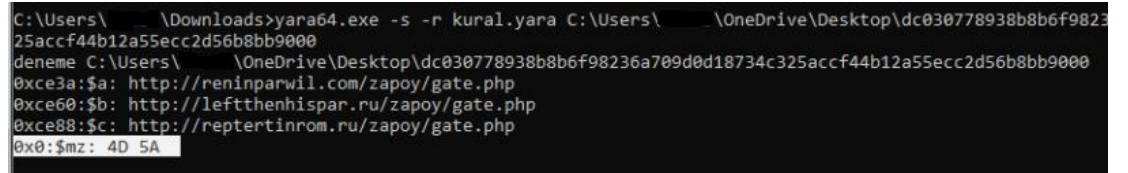
Şekil 25 Hex editörde malware yapısının görünümü

Zararlı yazılım hex editör ile açıldığında ilk 2 baytı görüntülenebilir şekilde. Bu baytlar sayesinde PE olup olmadığı anlaşılabilir. 4D 5A Offsetinde MZ yapısı görüntülenebiliyor.

```
rule deneme
{
  strings:
    $a = "http://reninparwil.com/zapoy/gate.php"
    $b = "http://leftthenhispar.ru/zapoy/gate.php"
    $c = "http://reptertinrom.ru/zapoy/gate.php"
    $mz = {4D 5A}
  condition:
    $a or $b or $c or $mz
}
```

Şekil 26 Oluşturulan yara kuralına bulunan MZ'lerin eklenmesi

Yara üzerinde oluşturulan yapıya geçiş yapıldıktan sonra yapı içerisine bulunan "MZ" değeri de eklenir. Condition kısmında ise "MZ" eklenerek eşleştirilmesi sağlanır.



Şekil 27 Güncellenen yara kuralı ile tarama yapma ve MZ değerinin görüntülenmesi

Güncellenen Yara kuralı ile yeniden bir tarama yapıldığında MZ değerinin elde edildiği görüntülenmektedir.

Yara aracı üzerinde hayal gücüne bağlı olarak kurallar yazılabilir. Ayrıca "https://github.com/InQuest/awesome-yara" bağlantısı üzerinden düzenli olarak güncellenen yara kuralları da güzel bir alternatiftir.

YARA Modülleri

- PE ve Cuckoo gibi modüller yara içerisine eklenip birlikte kullanılabilir.
- Modül ekleme işlemi Python diline çok benzemektedir.
 - import “pe”
 - import “cuckoo”

- **PE Modülü**

Bu modül, dosyanın format özelliklerini kullanarak PE dosyaları için çok daha ayrıntılı kurallar oluşturulmasına olanak sağlar. PE başlığı altında bulunan alanlardan çoğunun bulunmasını sağlar.

- **Cuckoo Modülü**

Cuckoo modülü, Sandbox üzerinde ki davranışlara göre Yara kuralı oluşturulmasına olanak sağlar.

```
import "cuckoo"

rule Test
{
  condition:
    cuckoo.network.http_request(\http://deneme.com\)
}
/ => Escape
\ => Regex begin-end
```

Şekil 28 Cuckoo modülünün kullanım şekli

- **Magic Modülü**

Bu modül “Standart Unix” komutu temelli dosya tipi ve türünü tespit etmemize olanak sağlar.

```
magic.mime_type()=="application/pdf"
```

Şekil 29 Magic modülünün kullanım şekli

- **Hash Modülü**

Bu modül dosya hash değerlerini hesaplarken aynı zamanda imza oluşturmamızı da sağlar.


```
import "hash"

rule CryWanna {

condition:
    hash.md5(0,filesize) == "db3491234asdfkfg439dk23kas1234k"

}
```

Şekil 30 Hash modülünün kullanım şekli

- **Dotnet Modülü**

Bu modül .NET tabanlı dosyalarda nitelikleri ve özelliklerin kullanılmasına olanak sağlar. Bu özellikleri kullanarak çok daha ince kurallar oluşturulabilir.

```
dotnet.version=="v2.0.872384"
```

Şekil 31 Dotnet modülünün kullanım şekli

Kural Geliştirmede Yardımcı Araçlar

- **Yaragen**

Bu araç zararlı yazılım üzerinde ki dizeleri kaldırırken aynı zamanda bu zararlı yazılımlardan yara kuralları oluşturur.

```
rule ef027405492bc0719437eb58c3d2774cc87845f30c40040bbebbcc09a4e3dd18 {
  meta:
    description = "Auto-generated rule - file ef027405492bc0719437eb58c3d2774cc87845f30c40040bbebbcc09a4e3dd18"
    author = "Florian Roth"
    reference = "http://blog.talosintelligence.com/2017/10/cyber-conflict-decoy-document.html"
    date = "2017-10-23"
    hash1 = "ef027405492bc0719437eb58c3d2774cc87845f30c40040bbebbcc09a4e3dd18"
  strings:
    $x1 = "netwf.dll" fullword wide
    $s1 = "%s - %s - %2.2x" fullword wide
    $s2 = "%s - %lu" fullword ascii
    $s4 = "%s \\"%s\\", %s" fullword wide
    $s5 = "%j%Xjsf" fullword ascii
```

Şekil 32 Yaragen yardımcı aracın görünümü

- **Yabin**

Bu araç, kötü amaçlı yazılım içerisinde ki çalıştırılabilir kod üzerinden Yara imzası oluşturulmasına olanak sağlar.

```
python yabin.py --help
usage: yabin.py [-h] [-y YARA] [-yh YARAHUNT] [-d] [-w ADDTOWHITELIST]
               [-f FUZZYHASH] [-m MALWAREADD] [-s MALWARESEARCH]

Yabin - Signatures and searches malware

optional arguments:
  -h, --help            show this help message and exit
  -y YARA, --yara YARA  Generate yara rule for the file or folder
  -yh YARAHUNT, --yaraHunt YARAHUNT
                        Generate wide yara rule (any of, not all of). Useful
                        for hunting for related samples or potentially
                        malicious files that share any of the code - but
                        liable to false positive
  -d, --deleteDatabase  Empty the whitelist and malware database
  -w ADDTOWHITELIST, --addToWhitelist ADDTOWHITELIST
                        Add a file or folder to the whitelist
```

Şekil 33 Yabin yardımcı aracın görünümü

Python derleyicisinde çalışan bu aracın github üzerinde 140 MB boyutunda bir beyaz listesi bulunmaktadır.

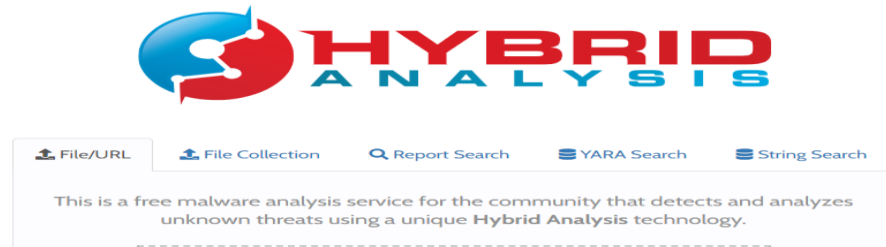
▪ Paste Hunter

Kopyalanan veri taşıyan siteleri sorgulamak için tasarlanan bir Python3 uygulamasıdır. Ham içerik arayan kullanıcılar için bir yara kuralı oluşturur. Ayrıca ElasticSearch, CSV, JSON SMTP gibi çıkış modüllerini de destekler.

▪ Hybrid Analysis

Kaçan en kötü zararlı yazılımlar için bile bellek dökümü analizleriyle bir yaklaşıma girer ve tüm veriler otomatik olarak işlenir.

Bu araç Sandbox teknolojisinden de yararlanmaktadır.

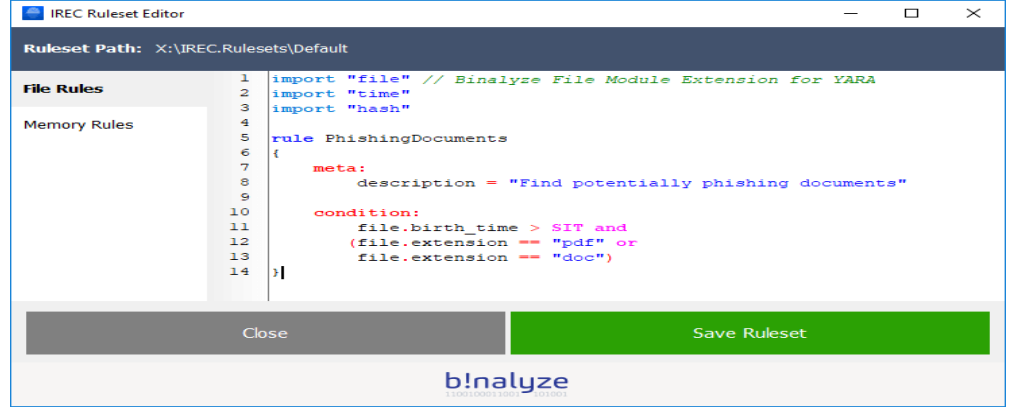


Şekil 34 Hybrid Analysis yardımcı aracın görünümü

▪ Binalyze – IREC

Ram İmajı, \$MFT olarak CSV,DNS Ön belleği gibi bilgileri toplar. Taşınabilir. Herhangi bir yüklemeye gerek duymaz.

Windows'un tüm sürümlerinde başarılı şekilde çalışmaktadır. Yara aracını destekler. Tek bir tıklama ile canlı sistemden kritik kanıtlar toplamamıza olanak sağlar.



Şekil 35 Binalyze yardımcı aracın görünümü

Etkin Yara Kullanımı

- Tersine mühendislik, maliyet olarak zararlı yazılıma uygulanacak en pahalı yoldur. Bu sebeple maliyetin düşmesi amaçlanır. Maliyetin değerinin düşmesi için analiz sürecini kısaltmak en mantıklı seçenektir.
- Yara'nın bir diğer kullanımı ise zararlı yazılımda ki kaynakları tespit etme ve bu kaynakları kurallara eklemektir. Bu kaynaklarda Simge, Konfigürasyon ayarları ve fonksiyonlar içerir.
- Bu kaynakları Yara imzası olarak kullanmak için Resource Hacker gibi bir araç kullanarak çıkartma işlemi uygulanır. Ardından ise doğrudan Yara'da temsil edilen Hexadecimal String döndürülür.
- Yarayı etkin kullanmanın en iyi yolu, malware tarafından gelen fonksiyonları temsil eden baytları kullanmaktır.

VOLATİLİTY İLE MALWARE ANALİZİ

Volatility, açık kaynak kodlu bir hafıza analiz aracıdır. Memory dump yani bellek dökümü dosyalarını Windows, Linux, MacOS platformlarında analiz edebilme fırsatı sunmaktadır.

Uygulamalı olarak İran nükleer santrallerine saldırıda kullanılan StuxNet zararlı yazılımının Linux üzerinden RAM bellek dökümünün analizi yapılacaktır.

```
(root@kali)~[~/Desktop/volatility]
# python vol.py imageinfo -f stuxnet.vmem
Volatility Foundation Volatility Framework 2.6.1
```

Şekil 36 Volatility imageinfo komutu

İlk olarak eldeki dump dosyasının hangi işletim sistemine ait olduğu bulunmakla başlanır.

```
INFO : volatility.debug : Determining profile based on KDBG search...
      Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
      AS Layer1 : IA32PagedMemoryPae (Kernel AS)
      AS Layer2 : FileAddressSpace (/root/Desktop/volatility/stuxnet.vmem)
      PAE type : PAE
      DTB : 0x319000L
      KDBG : 0x80545ae0L
      Number of Processors : 1
      Image Type (Service Pack) : 3
      KPCR for CPU 0 : 0xffdff000L
      KUSER_SHARED_DATA : 0xffdff000L
      Image date and time : 2011-06-03 04:31:36 UTC+0000
      Image local date and time : 2011-06-03 00:31:36 -0400
```

Şekil 37 Volatility imageinfo komutu çıktısı

Komutun çıktısından görüldüğü üzere Windows XP Service Pack 2 işlemci mimarisine aitmiş. Bu bilgiye ulaşıldıktan sonra arka planda çalışan süreçleri hem daha rahat gözükmeleri için ağaçlandırma tekniği ile görüntülemek için aşağıdaki komut çalıştırılır;

```
(root@kali)~[~/Desktop/volatility]
# python vol.py pstree -f stuxnet.vmem
```

Şekil 38 Volatility pstree parametre

Şüphelenilen süreçleri biraz daha sınamak için ağ soketlerine de bakılabilir.

Offset(V)	PID	Port	Proto	Protocol	Address	Create Time
0x81dc2008	680	500	17	UDP	0.0.0.0	2010-10-29 17:09:05 UTC+0000
0x82061c08	4	445	6	TCP	0.0.0.0	2010-10-29 17:08:53 UTC+0000
0x82294aa8	940	135	6	TCP	0.0.0.0	2010-10-29 17:08:55 UTC+0000
0x821a5008	188	1025	6	TCP	127.0.0.1	2010-10-29 17:09:09 UTC+0000
0x81cb3d70	1080	1141	17	UDP	0.0.0.0	2010-10-31 16:36:16 UTC+0000
0x81da4d18	680	0	255	Reserved	0.0.0.0	2010-10-29 17:09:05 UTC+0000
0x81fdb98	1032	123	17	UDP	127.0.0.1	2011-06-03 04:25:47 UTC+0000
0x81c79778	1080	1142	17	UDP	0.0.0.0	2010-10-31 16:36:16 UTC+0000
0x81c20898	1200	1900	17	UDP	127.0.0.1	2011-06-03 04:25:47 UTC+0000
0x82060008	680	4500	17	UDP	0.0.0.0	2010-10-29 17:09:05 UTC+0000
0x81cb9e98	1580	5152	6	TCP	127.0.0.1	2010-10-29 17:09:05 UTC+0000
0x81da54b0	4	445	17	UDP	0.0.0.0	2010-10-29 17:08:53 UTC+0000

```
(root@kali)~[~/Desktop/volatility]
#
```

Şekil 41 Volatility connsnscan parametre çıktısı

Gerçek olduğu bilinen 680 numaralı PID UDP 500 ve UDP 4500 portlarını kullandığı görülmektedir. Fakat yine diğer lsass süreçlerinin hiç port açmamış olduğu görülmektedir.

```
(root@kali)~[~/Desktop/volatility]
# python vol.py -f stuxnet.vmem dlllist -p 680 |wc -l
Volatility Foundation Volatility Framework 2.6.1
87

(root@kali)~[~/Desktop/volatility]
# python vol.py -f stuxnet.vmem dlllist -p 868 |wc -l
Volatility Foundation Volatility Framework 2.6.1
38

(root@kali)~[~/Desktop/volatility]
# python vol.py -f stuxnet.vmem dlllist -p 1928 |wc -l
Volatility Foundation Volatility Framework 2.6.1
58
```

Şekil 42 Volatility dlllist parametresi

Bu sefer de süreçlerin kullandığı DLL sayılarına bakılırsa gerçek olduğu bilinen 680 PID ile diğerleri arasında fark görülmektedir.

Yeterince kanıt görüldükten sonra işlem sırası enjekte edilen kodu tespit etmeye geldi.

```
(root@kali)~[~/Desktop/volatility]
# python vol.py -f stuxnet.vmem malfind -p 1928 -D derya
Volatility Foundation Volatility Framework 2.6.1
```

Şekil 43 Volatility malfind parametresi

Malfind parametresi ile 868 ve 1928 PID numaralı süreçlerin kullanmış olduğu DLL dosyaları derya klasörüne indirilebilir.

```

WARNING : volatility.debug : For best results please install distorm3
Process: lsass.exe Pid: 1928 Address: 0x80000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x000000000080000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x000000000080010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
0x000000000080020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000000000080030 00 00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00 .....

Process: lsass.exe Pid: 1928 Address: 0x1000000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 2, Protection: 6

0x000000000100000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x000000000100010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
0x000000000100020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000000000100030 00 00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00 .....

Process: lsass.exe Pid: 1928 Address: 0x6f0000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x00000000006f000 29 87 7f ae 00 00 00 00 ff ff ff ff 77 35 00 01 ).....w5..
0x00000000006f010 4b 00 45 00 52 00 4e 00 45 00 4c 00 33 00 32 00 K.E.R.N.E.L.3.2.
0x00000000006f020 2e 00 44 00 4c 00 4c 00 2e 00 41 00 53 00 4c 00 ..D.L.L...A.S.L.
0x00000000006f030 52 00 2e 00 30 00 33 00 36 00 30 00 62 00 37 00 R...0.3.6.0.b.7.

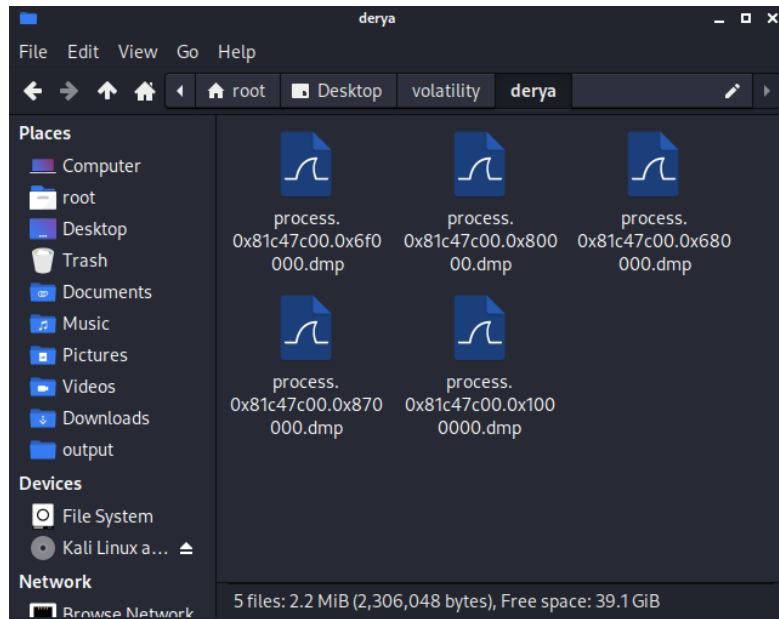
Process: lsass.exe Pid: 1928 Address: 0x680000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x000000000068000 90 06 68 00 c6 07 68 00 24 00 68 00 a5 04 00 00 ..h...h.$..h....
0x000000000068010 f2 04 68 00 48 06 00 00 c9 04 68 00 29 00 00 00 ..h.H.....h.)...
0x000000000068020 00 00 6f 00 e8 13 00 00 00 5a 77 4d 61 70 56 69 ..o.....ZwMapVi
0x000000000068030 65 77 4f 66 53 65 63 74 69 6f 6e 00 5a 51 81 c1 ewOfSection.ZQ..

Process: lsass.exe Pid: 1928 Address: 0x870000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

```

Şekil 44 Volatility malfind parametre çıktısı



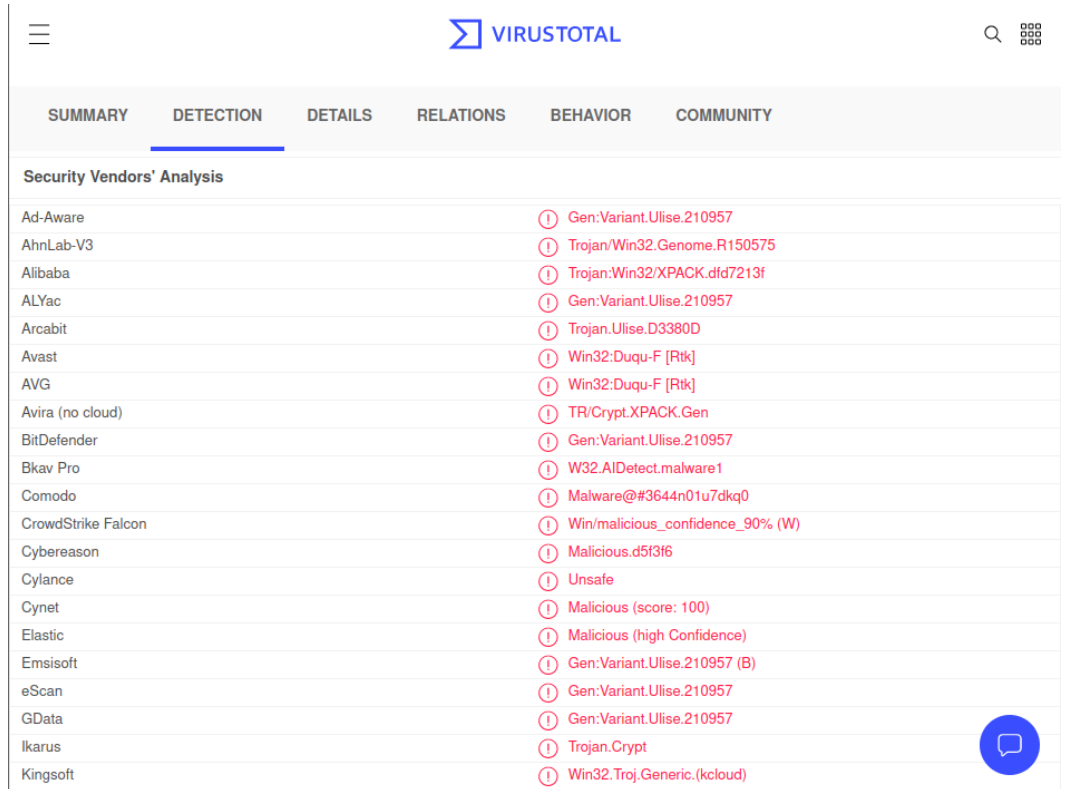
Şekil 45 Dump edilen dll dosyalarının derya klasöründeki görünümü

Enjekte edilen kodu tespit etmek için sha256sum aracı kullanılarak dosya uzantısı dmp olan bütün dosyaların hashi alınıp VirusTotal sitesinde taratmak bir seçenektir.

```
(root@kali)~[~/Desktop/volatility/derya]
# sha256sum *.dmp
abce3e79e26b5116fe7f3d40d21eaa4c8563e433b6086f9ec07c2925593f69dc process.0x81c47c00.0x6f0000.dmp
2b2945f7cc7cf5b30ccdf37e2adbb236594208e409133bcd56f57f7c009ffe6d process.0x81c47c00.0x800000.dmp
163b7da37df4ae6dafb5bf88b319dabf7846cee73d4192c6a7593e835857a8 process.0x81c47c00.0x680000.dmp
10f07b9fbbc6a8c6dc4abf7a3d31a01e478accd115b33ec94fe885cb296a3586 process.0x81c47c00.0x870000.dmp
e97d61f7393ac5838a1800f3e9aa22c6205f4d7e2bde494573d35c57bc9b7819 process.0x81c47c00.0x1000000.dmp
```

Şekil 46 Volatility sha256sum komutu

Alınan hashler VirusTotal ile taratılır. Taratma sonucunda “e97d61f7393ac583a1800f3e9aa22c6205f4d7e2bde494573d35c57bc9b7819” değeri antivirüs programları tarafından yakalandı.



SUMMARY	DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
Security Vendors' Analysis					
Ad-Aware					Gen:Variant.Ulise.210957
AhnLab-V3					Trojan/Win32.Genome.R150575
Alibaba					Trojan:Win32/XPack.dfd7213f
ALYac					Gen:Variant.Ulise.210957
Arcabit					Trojan.Ulise.D3380D
Avast					Win32:Duqu-F [Rtk]
AVG					Win32:Duqu-F [Rtk]
Avira (no cloud)					TR/Crypt.XPACK.Gen
BitDefender					Gen:Variant.Ulise.210957
Bkav Pro					W32.AIDetect.malware1
Comodo					Malware@#3644n01u7dkq0
CrowdStrike Falcon					Win/malicious_confidence_90% (W)
Cybereason					Malicious.d5f3f6
Cylance					Unsafe
Cynet					Malicious (score: 100)
Elastic					Malicious (high Confidence)
Emsisoft					Gen:Variant.Ulise.210957 (B)
eScan					Gen:Variant.Ulise.210957
GData					Gen:Variant.Ulise.210957
Ikarus					Trojan.Crypt
Kingsoft					Win32.Troj.Generic.(kcloud)

Şekil 47 VirusTotal’de taratılan Hashin görünümü

Açık bir şekilde Avast ve AVG tarafından varılan sonuca göre Enjekte edilen kod StuxNet’in gelişmiş hali olan Duqu zararlı yazılımıdır.

```
(root@kali)~[~/Desktop/volatility]
# python vol.py -f stuxnet.vmem modscan
Volatility Foundation Volatility Framework 2.6.1
```

Şekil 48 Volatility modscan parametresi

Bilindiği üzere bazı zararlı yazılımlar sürücülerin arasına gizlenebilir. Bu şüphelenilen sürücüler de bir kontrol etmek için modscan parametresi kullanılır.

Offset(P)	Name	Base	Size	File
0x000000001e2a530	mrxnet.sys	0xb21d8000	0x3000	\\??\C:\WINDOWS\system32\Drivers\mrxnet.sys
0x000000001e9a578		0x00000001	0x1	
0x000000001e9a898		0x00480048	0x44005c	
0x000000001f9ce78	HIDCLASS.SYS	0xf872a000	0x9000	\SystemRoot\system32\DRIVERS\HIDCLASS.SYS
0x000000001fa6890	dxg.sys	0xbf9c3000	0x12000	\SystemRoot\System32\drivers\dxg.sys
0x000000001faa490	mouhid.sys	0xf745c000	0x3000	\SystemRoot\system32\DRIVERS\mouhid.sys
0x000000001fbb820	RDPCDD.sys	0xf8bb6000	0x2000	\SystemRoot\System32\DRIVERS\RDPCDD.sys
0x000000001fbf130	rasppoe.sys	0xf87ea000	0xb000	\SystemRoot\system32\DRIVERS\rasppoe.sys
0x000000001fc1130	CmBatt.sys	0xf8b5a000	0x4000	\SystemRoot\system32\DRIVERS\CmBatt.sys
0x000000001fce178	flpydisk.sys	0xf89da000	0x5000	\SystemRoot\system32\DRIVERS\flpydisk.sys
0x000000001fde0d8	netbt.sys	0xb2d4c000	0x28000	\SystemRoot\system32\DRIVERS\netbt.sys
0x000000001fde408	tcpip.sys	0xb2d74000	0x59000	\SystemRoot\system32\DRIVERS\tcpip.sys
0x000000001fdea90	ipsec.sys	0xb2dcd000	0x13000	\SystemRoot\system32\DRIVERS\ipsec.sys
0x000000001fdec98	Npfs.SYS	0xf89fa000	0x8000	\SystemRoot\System32\Drivers\Npfs.SYS
0x000000001fe8300	intelppm.sys	0xb256c000	0x9000	\SystemRoot\system32\DRIVERS\intelppm.sys
0x000000002019f18	ParVdm.SYS	0xf8bf8000	0x2000	\SystemRoot\System32\Drivers\ParVdm.SYS
0x00000000207c360	vmmemctl.sys	0xf8bfa000	0x2000	\\??\C:\Program Files\VMware\VMware Tools\Drivers\memctl
0x000000002084130	audstub.sys	0xf8d60000	0x1000	\SystemRoot\system32\DRIVERS\audstub.sys
0x0000000020a54a0	HTTP.sys	0xb24fb000	0x41000	\SystemRoot\System32\Drivers\HTTP.sys
0x0000000020a5908	wdmaud.sys	0xb23ce000	0x15000	\SystemRoot\system32\drivers\wdmaud.sys
0x0000000020abb40	netbios.sys	0xf885a000	0x9000	\SystemRoot\system32\DRIVERS\netbios.sys
0x0000000020bce08	wanarp.sys	0xf88aa000	0x9000	\SystemRoot\system32\DRIVERS\wanarp.sys

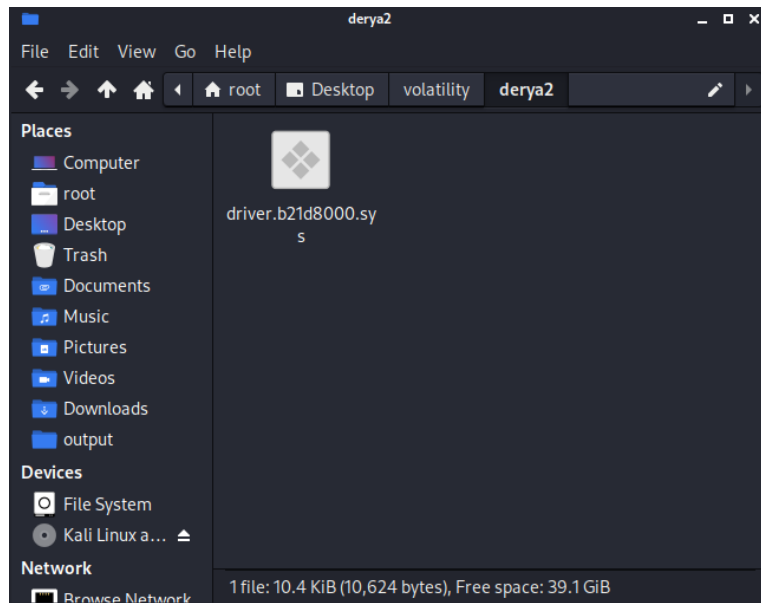
Şekil 49 Volatility modscan parametre çıktısı

Açık ara bir sürücünün göze battığı görülmektedir. Bu yüzden ilk olarak bu sürücü test edilecek.

```
(root@kali) - [~/Desktop/volatility]
# python vol.py -f stuxnet.vmem moddump --base 0xb21d8000 -D derya2
Volatility Foundation Volatility Framework 2.6.1
```

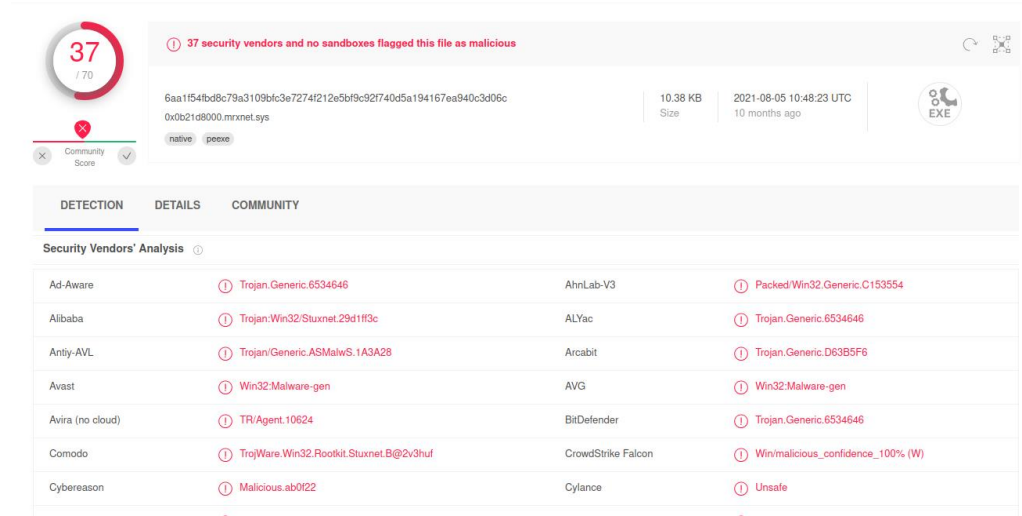
Şekil 50 Volatility moddump parametresi

-base parametresi ile dump edilecek hedef sürücünün base değerini verip moddump parametresi ile sürücü dump edilir.



Şekil 51 dump edilen sürücünün dosya içindeki görünümü

Dump edilen sürücü yine VirusTotal testinden geçirilir.



DETECTION	DETAILS	COMMUNITY	
Security Vendors' Analysis			
Ad-Aware	Trojan.Generic.6534646	AhnLab-V3	Packed/Win32.Generic.C153554
Alibaba	Trojan.Win32/Stuxnet.29d1f8c	ALYac	Trojan.Generic.6534646
Antiy-AVL	Trojan.Generic.ASMalwS.1A3A28	Arcabit	Trojan.Generic.D63B5F6
Avast	Win32/Malware-gen	AVG	Win32/Malware-gen
Avira (no cloud)	TR/Agent.10624	BitDefender	Trojan.Generic.6534646
Comodo	TrojWare.Win32.Rootkit.Stuxnet.B@2v3huf	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.ab0f22	Cylance	Unsafe

Şekil 52 Sürücünün VirusTotal taraması görünümü

Görüldüğü üzere bu sürücünün malware olduğu sonucuna ulaşılmıştır.

ZARARLI YAZILIMLARA KARŞI ALINABİLECEK ÖNLEMLER

Bu bölümde zararlı yazılımlara karşı nasıl önlem alınması gerektiğinden ve korunma yöntemlerinden bahsedilmiştir.

MAKİNELERDE ALINABİLECEK ÖNLEMLER

Makinelerde alınabilecek önlemlerin başında güncellemelerin düzenli yapılması gelmektedir. Güvenlik yamalarının güncellenmesi düşünüldüğünde daha fazla önem taşımaktadır. Her güncelleme de tespit edilmiş bir açığı kapatmak gibi yenilemeler sunulmaktadır. Microsoft işletim sistemi kurulu bilgisayarın en son çıkmış sürümü ile kurulması da önerilmektedir.

Kurumsal ağlarda malware riskini azaltmak için yamalar merkezi bir makineye aktarılıp buradan da diğer makinelere dağıtılması sağlanabilir. Bu yöntem yama güncellemesini tüm makinelere dağıtarak gözden kaçma olayını engelleyebilir.

Yine kurumun kullanıcılarını antivirüs yazılımları hakkında bilgilendirip kullanmaya teşvik etmesi ve güncellemelerin önemi hakkında bilgi vermesi oldukça etkili bir önlem sağlayacaktır.

Kullanılan internet tarayıcısına göre güvenlik ayarları yapılmalı ve yamalar düzenli olarak takip edilip uygulanmalıdır. Kurumlarda ise güvenli

olduğu düşünölen tarayıcı çalışanlarına sunup gerekli ve güncel güncellemeleri çalışanlarına ulaştırabilir. Ayrıca “noScript” gibi eklentiler tarayıcılara eklenerek güvenliğin artırılması sağlanabilir.

Windows SP2 ile kullanıcının veya IP adresinin oluşturabileceği bağlantı sayısı denetlenebilir. Böylece DOS ve DDOS ataklarına karşı bir önlem de oluşturulmuş olur.

AĞDA ALINABİLECEK ÖNLEMLER

L2 ve L3 Cihazlar ile Alınabilecek Önlemler

OSI’nin 2. Ve 3. Katmanlarında çalışan yerel ağ cihazlarında alınabilecek önlemleri içermektedir.

2.Katman için MAC adresi bazında güvenlik ve broadcast/multicast sınırlandırılması, 3.Katman için ise VLAN bazlı güvenlik çözümleri, erişim listeleri ile alınabilecek çözümler gibi konulara değinilecektir.

MAC Adresi Bazında Güvenlik: Ağa kontrolsüz bilgisayar erişimini engelleyerek, kötü yazılım bulaşmış bilgisayarın tespitini kolaylaştırmaktadır. Tüm önlemlere rağmen IP ve MAC adreslerini değiştirerek kendini gizleyebilmektedir. Bu küçük aldatma da yönetilebilir anahtarlama cihazları ile engellenebilir.

```
Interface <int adı> <int.no>
switchport port-security
switchport port-security maximum <toplaml PC sayısı>
switchport port-security violation <protect | restrict | shutdown>
switchport port-security mac-address <PC'nin MAC adresi>
```

Yukarıdaki yapılandırma örneği Cisco marka bir anahtarlama cihazına aittir.

Broadcast/Multicast Sınırlandırması: Broadcast yani yayın adresi, ağlar arası iletişimde gönderilen bilgi paketinin tüm ağda bulunan cihazlar tarafından alınması için belirlenen adrestir. DoS saldırıları broadcast adresi üzerinden de yapılabilmektedir. Bu saldırıları da önleyebilmek için broadcast sınırlaması yapılmalıdır. Broadcast için kullanılan en yaygın protokoller IP ve Ethernet’tir.

Broadcast/Multicast trafiğinin 1 saniyede belirlenen düzeyi aşması durumuna broadcast/Multicast fırtınası denmektedir. Yine bu sorun ağ anahtarlama cihazları ile giderilebilmektedir. Ara yüz ağ genişliğinin belirlenen seviyeden çok olması durumunda aşan kısmın bloklanması veya loglanması sağlanabilir.

```
interface <int adı> <int.no>
storm-control multicast level <Yüzde.Küsüratı>
storm-control broadcast level <Yüzde.Küsüratı>
storm-control unicast level <Yüzde.Küsüratı>
storm-control action <shutdown | trap>
```

Yukarıdaki yapılandırma örneği Cisco marka bir anahtarlama cihazına aittir.

VLAN Bazlı Güvenlik Çözümleri: üçüncü katmanda alınabilecek önlemlerden olan Vlan bazlı güvenlik ağ cihazlarında yapılacak sınırlandırılmalarla sağlanabilmektedir. Bazı Cisco marka ağ cihazı konfigürasyon ayarları;

- *ip verify unicast source reachable-via rx allowdefault:* VLAN’da belirtilmiş IP adresleri dışında başka IP adresi ile o VLAN’dan trafik çıkmasını engeller.
- *no ip redirects:* ICMP redirect desteğini kapatır. ICMP, internet kontrol mesaj protokolü, sorunları haberleşen birimlere bildiren geri besleme mekanizmasıdır.
- *no ip unreachable:* ICMP unreachable paketlerinin geri yollanması engeller.
- *no ip proxy-arp:* Ağ geçidi tanımlamamış ve yanlış tanımlamış bir istemcinin yönlendirici tarafından tespit edilip o istemcilere ağ geçidi hizmetinin otomatik verilmesi özelliğini kapatır.

Erişim Listeleri ile Alınabilecek Önlemler: Erişim listeleri kullanılarak yönlendiricilerde alınan önlemlerle kötü amaçlı yazılımların ağ üzerindeki yükünü azaltarak kendilerini yaymalarını da engelleyebilir.

- Yönlendiriciye gelen paketlerdeki IP adresleri kontrol edilmelidir.
- Güvenlik açıklarının kullandığı bilinen portlar kapatılmalı veya kısıtlanmalıdır.TCP 135, 137, 139, 445 gibi..

- Erişimi engellenen trafik loglanarak saldırganın kimliği tespit edilebilir.

GÜVENLİK CİHAZLARI İLE ALINABİLECEK ÖNLEMLER

Güvenlik duvarları, antivirüs geçitleri, IDS sistemleri bu kategoride yer almaktadır.

➤ **Güvenlik Duvarları**

Güvenlik duvarları durum korumalı çalıştığı için düzgün ayarlanırsa zararlı yazılım içeren birçok bağlantıyı engelleyebilecektir. Düzgün ayarlara sahip olabilmek için kurum içinden dışarı tüm trafiğe izin verilip kurum dışından içeri sunucu portlarından erişim hariç tüm trafik engellenmelidir.

➤ **Antivirüs Geçitleri**

Akan trafiği zararlı içeriğe göre kontrol eden mekanizmalardır. Özellikle büyük ağlarda eposta trafiği için kullanılabilmektedir. Malware bulaşma yollarından en etkillerinden biri olan eposta için etkili bir önlemdir.

➤ **IDS**

Türkçe karşılığı saldırı tespit sistemi anlamına gelen IDS, ağ trafiği içerisindeki güvenlik ihlallerini tespit etmek için kullanılan yazılımsal veya donanımsal sistemlerdir. IDS, ağı malware karşı koruma altına alması yanında malware kaynağı tespitini de büyük ölçüde hızlandırmaktadır. Snort gibi açık kaynak kodlu sistemler kullanılabilir.

KAYNAKÇA

- ❖ Zararlı Yazılımların Göstermiş Oldukları Davranışlara Göre Analiz ve Tespit Edilmesi/ Ömer ASLAN
- ❖ Zararlı Yazılım Tespiti İçin Yazılım Davranış Analizi/ Salih Yesir
- ❖ Kötücül ve Casus Yazılımlar/ Gürol CANBEK ve Şeref Sağıroğlu
- ❖ Siber Saldırıları İzleme Yöntemleri ve Zararlı Yazılım Analizi/ Elif Tuğba KILIÇ
- ❖ <https://blog.btrisk.com>
- ❖ <https://illinois.touro.edu>
- ❖ <https://www.slideshare.net>
- ❖ <https://www.mertsarica.com>
- ❖ <https://www.bgasecurity.com>
- ❖ <https://medium.com>
- ❖ <https://slideplayer.biz.tr>
- ❖