İmzalar

Cuckoo ile analiz sonuçlarına karşı çalıştırabileceğiniz özel imzalar oluşturabilir ve bu imzaları kullanarak belirli bir zararlı davranışı veya ilgilendiğiniz bir göstergeyi tanımlayabilirsiniz.

Bu imzalar, analizlere bir bağlam sağlamak için çok kullanışlıdır: sonuçların yorumunu basitleştirmeleri yanı sıra ilgi çekici kötü amaçlı yazılım örneklerini otomatik olarak tanımlamak açısından da önemlidir.

Cuckoo'nun imzalarını kullanabileceğiniz bazı örnekler:

- Bazı benzersiz davranışları (dosya adları veya mutex'lar gibi) izleyerek ilgilendiğiniz belirli bir kötü amaçlı yazılım ailesini tanımlama.
- Kötü amaçlı yazılımın sistemi üzerinde gerçekleştirdiği ilginç değişiklikleri tespit etme, örneğin cihaz sürücüleri kurulumu.
- Bankacılık Truva atları veya fidye yazılımı gibi belirli kötü amaçlı yazılım kategorilerini izole ederek tanımlama, bunlar tarafından genellikle gerçekleştirilen tipik eylemleri izole ederek.
- Örnekleri kötü amaçlı yazılım/bilinmeyen kategorilere sınıflandırma (temiz örnekleri tanımlamak mümkün değildir).

Kendi oluşturduğumuz ve diğer Cuckoo kullanıcıları tarafından oluşturulmuş imzaları Community reposunda bulabilirsiniz.

Başlarken

İmza oluşturma süreci oldukça basittir ve sadece Python programlamaya iyi bir anlayış gerektirir.

Öncelikle, tüm imzaların Cuckoo'nun `cuckoo/cuckoo/signatures/` dizininde veya Community reposunun `modules/signatures/` dizininde (Community reposu hala eski dizin yapısını kullanıyor) bulunması gerekmektedir.

Aşağıdaki temel bir örnek imzadır:

```
from cuckoo.common.abstracts import Signature

class CreatesExe(Signature):
    name = "creates_exe"
    description = "Creates a Windows executable on the filesystem"
    severity = 2
```

```
    categories = ["generic"]
    authors = ["Cuckoo Developers"]
    minimum = "2.0"

    def on_complete(self):
        return self.check_file(pattern=".*\\.exe$", regex=True)
```

Gördüğünüz gibi, yapı gerçekten basit ve diğer modüllerle tutarlıdır. Daha sonra detaylara gireceğiz, ancak Cuckoo'nun 1.2 sürümünden itibaren imza oluşturma sürecini çok daha kolay hale getiren bazı yardımcı işlevler sağlamaktadır.

Bu örnekte, özet içinde erişilen tüm dosyaları kontrol ediyoruz ve 'exe' ile biten bir şey olup olmadığını kontrol ediyoruz: bu durumda eşleşen bir imza olduğu anlamına gelir ve `True` değerini döndürür, aksi takdirde `False` değerini döndürür.

`on_complete` fonksiyonu, cuckoo imza sürecinin sonunda çağrılır. Diğer işlevler, belirli olaylarda önce çağrılacak ve daha sofistike ve hızlı imzalar yazmanıza yardımcı olacaktır.

İmza eşleşirse, global konteynerin "signatures" bölümüne yaklaşık olarak aşağıdaki gibi yeni bir giriş eklenir:

```
"signatures": [
    {
        "severity": 2,
        "description": "Creates a Windows executable on the filesystem",
        "alert": false,
        "references": [],
        "data": [
            {
                "file_name": "C:\\d.exe"
            }
        ],
        "name": "creates_exe"
    }
]
```

Yeni İmza Oluşturmak

Sizinle birlikte çok basit bir imza oluşturma sürecini daha iyi anlamanızı sağlamak için birlikte oluşturacağız ve adımları ile kullanılabilir seçenekleri inceleyeceğiz. Bu amaçla, analiz edilen kötü amaçlı yazılımın 'i_am_a_malware' adında bir mutex açıp açmadığını kontrol eden bir imza oluşturacağız.

İlk yapmanız gereken bağımlılıkları içe aktarmak, bir iskelet oluşturmak ve bazı başlangıç özniteliklerini tanımlamaktır. Şu anda ayarlayabileceğiniz öznitelikler şunlardır:

- name: imza için bir tanımlayıcı.
- description: imzanın neyi temsil ettiğine dair kısa bir açıklama.
- severity: eşleşen olayların ciddiyetini belirleyen bir sayı (genellikle 1 ile 3 arasında).
- categories: eşleşen olayın türünü tanımlayan bir kategori listesi (örneğin "banker", "enjeksiyon" veya "anti-vm").
- families: imzanın özellikle bilinen bir kötü amaçlı yazılım ailesiyle eşleşiyorsa, bir liste halinde kötü amaçlı yazılım aile adları.
- authors: imzayı oluşturan kişilerin listesi.
- references: imzaya bağlam sağlamak için referansların (URL'lerin) listesi.
- enable: False olarak ayarlanırsa imza atlanacaktır.
- alert: True olarak ayarlanırsa, imzanın raporlanması gerektiğini belirtmek için kullanılabilir (belki de özel bir raporlama modülü tarafından).
- minimum: Bu imzanın başarıyla çalıştırılabilmesi için gereken minimum Cuckoo sürümü.
- maximum: Bu imzanın başarıyla çalıştırılabilmesi için gereken maksimum Cuckoo sürümü."

Örneğimizde aşağıdaki iskeleti oluşturacağız:

```
from cuckoo.common.abstracts import Signature

class BadBadMalware(Signature): # We initialize the class inheriting Signature.
    name = "badbadmalware" # We define the name of the signature
    description = "Creates a mutex known to be associated with
Win32.BadBadMalware" # We provide a description
    severity = 3 # We set the severity to maximum
    categories = ["trojan"] # We add a category
    families = ["badbadmalware"] # We add the name of our fictional malware
family
    authors = ["Me"] # We specify the author
    minimum = "2.0" # We specify that in order to run the signature, the user will
simply need Cuckoo 2.0

    def on_complete(self):
        return
```

Bu, tamamen geçerli bir imzadır. Henüz bir şey yapmaz, bu yüzden şimdi imzanın eşleşmesi için koşulları tanımlamamız gerekiyor.

Dediğimiz gibi, belirli bir mutex adını eşleştirmek istiyoruz, bu nedenle şu şekilde devam ederiz:

```
```

```
from cuckoo.common.abstracts import Signature

class BadBadMalware(Signature):
    name = "badbadmalware"
    description = "Creates a mutex known to be associated with
Win32.BadBadMalware"
    severity = 3
    categories = ["trojan"]
    families = ["badbadmalware"]
    authors = ["Me"]
    minimum = "2.0"

    def on_complete(self):
        return self.check_mutex("i_am_a_malware")
```

Böylece, şimdi imzamız, analiz edilen kötü amaçlı yazılımın belirtilen mutex'u açılırken gözlemlenip gözlemlenmediğini döndürecektir.

Daha açık olmak ve doğrudan global konteynere erişmek istiyorsanız, önceki imzayı şu şekilde çevirebilirsiniz:

```
from cuckoo.common.abstracts import Signature

class BadBadMalware(Signature):
    name = "badbadmalware"
    description = "Creates a mutex known to be associated with
Win32.BadBadMalware"
    severity = 3
    categories = ["trojan"]
    families = ["badbadmalware"]
    authors = ["Me"]
    minimum = "2.0"

    def on_complete(self):
        for process in self.get_processes_by_pid():
            if "summary" in process and "mutexes" in process["summary"]:
                for mutex in process["summary"]["mutexes"]:
                    if mutex == "i_am_a_malware":
                        return True

        return False
```

Evented İmzalar

1.0 sürümünden itibaren Cuckoo, daha yüksek performanslı imzalar yazma olanağı sunmaktadır. Geçmişte, her imzanın analiz sırasında toplanan tüm API çağrıları koleksiyonu üzerinde döngü yapması gerekiyordu. Bu, bu tür bir koleksiyonun büyük boyutta olması durumunda gereksiz performans sorunlarına neden oluyordu.

1.2'den itibaren Cuckoo, yalnızca "evented imzaları" desteklemektedir. Eski run fonksiyonuna dayalı imzalar, `on_complete` kullanarak port edilebilir. Temel fark, bu yeni formatta tüm imzaların paralel olarak yürütülecek olması ve API çağrıları koleksiyonu üzerinde tek bir döngü ile her imza için `on_call()` adlı bir geri çağrı fonksiyonunun çağrılacak olmasıdır.

Bu tekniği kullanan örnek bir imza şudur:

```
from cuckoo.common.abstracts import Signature

class SystemMetrics(Signature):
    name = "generic_metrics"
    description = "Uses GetSystemMetrics"
    severity = 2
    categories = ["generic"]
    authors = ["Cuckoo Developers"]
    minimum = "2.0"

    # Evented signatures can specify filters that reduce the amount of
    # API calls that are streamed in. One can filter Process name, API
    # name/identifier and category. These should be sets for faster lookup.
    filter_processnames = set()
    filter_apinames = set(["GetSystemMetrics"])
    filter_categories = set()

    # This is a signature template. It should be used as a skeleton for
    # creating custom signatures, therefore is disabled by default.
    # The on_call function is used in "evented" signatures.
    # These use a more efficient way of processing logged API calls.
    enabled = False

    def on_complete(self):
        # In the on_complete method one can implement any cleanup code and
        #  decide one last time if this signature matches or not.
        #  Return True in case it matches.
        return False

    # This method will be called for every logged API call by the loop
    # in the RunSignatures plugin. The return value determines the "state"
    # of this signature. True means the signature matched and False it did not
```

```
this time.
    # Use self.deactivate() to stop streaming in API calls.
    def on_call(self, call, pid, tid):
        # This check would in reality not be needed as we already make use
        # of filter_apinames above.
        if call["api"] == "GetSystemMetrics":
            # Signature matched, return True.
            return True

        # continue
        return None
```

Satır içi yorumlar zaten kendi kendini açıklayıcıdır.

Bir imza eşleştiğinde başka bir olay tetiklenir.

```
required = ["creates_exe", "badmalware"]

def on_signature(self, matched_sig):
    if matched_sig in self.required:
        self.required.remove(matched_sig)

    if not self.required:
        return True

    return False
```

Bu tür bir imza, anormallikleri tanımlayan birkaç imzayı birleştirmek için kullanılabilir ve örneği sınıflandıran bir imza (malware uyarısı) oluşturabilir.

İşaretler ve Yardımcılar

1.2 sürümünden itibaren imzalar, imzayı tetikleyen şeyi tam olarak kaydetme yeteneğine sahiptir. Bu, kullanıcıların bu imzanın günlükte neden bulunduğunu daha iyi anlamalarına ve kötü amaçlı yazılım analizine daha iyi odaklanabilmelerine olanak tanır.

İşaretler ve yardımcılar konusundaki örnekler için şu anda Cuckoo Community'e başvurun.

Revision #1
Created 19 January 2024 11:56:09 by Ertan Sözer
Updated 19 January 2024 11:59:30 by Ertan Sözer