Analiz Paketleri

Analiz Paketleri bölümünde açıklandığı gibi, analiz paketleri, Cuckoo'nun analizör bileşeninin bir konuk ortamındaki belirli bir dosya için analiz prosedürünü nasıl yürütmesi gerektiğini açıklayan yapılandırılmış Python sınıflarıdır.

Zaten bildiğiniz gibi, kendi paketlerinizi oluşturabilir ve bunları varsayılan olanlarla birlikte ekleyebilirsiniz. Yeni paketler tasarlamak çok kolaydır ve sadece temel düzeyde programlama ve Python dilini anlama gerektirir.

Başlarken

Bir örnek olarak, genel Windows uygulamalarını analiz etmek için varsayılan paketi inceleyeceğiz. Bu paket, `$CWD/analyzer/windows/packages/exe.py` konumunda bulunur (Git deposunda `cuckoo/data/analyzer/windows/packages/exe.py` olarak çevrilir):

```
from lib.common.abstracts import Package

class Exe(Package):
    """EXE analysis package."""

    def start(self, path):
        args = self.options.get("arguments")
        return self.execute(path, args)
```

Bu gerçekten kolay görünüyor, Package nesnesinden alınan tüm yöntemler sayesinde. Bir analiz paketinin Package nesnesinden aldığı başlıca yöntemlere bir göz atalım:

```
from lib.api.process import Process
from lib.common.exceptions import CuckooPackageError

class Package(object):
    def start(self):
        raise NotImplementedError
```

```
        def check(self):
            return True

        def execute(self, path, args):
            dll = self.options.get("dll")
            free = self.options.get("free")
            suspended = True
            if free:
                suspended = False

            p = Process()
            if not p.execute(path=path, args=args, suspended=suspended):
                raise CuckooPackageError(
                    "Unable to execute the initial process, analysis aborted."
                )

            if not free and suspended:
                p.inject(dll)
                p.resume()
                p.close()
                return p.pid

        def finish(self):
            if self.options.get("procmemdump"):
                for pid in self.pids:
                    p = Process(pid=pid)
                    p.dump_memory()
            return True
```

Kodu inceleyelim:

- Satır 1: Windows işlemlerini oluşturup manipüle etmek için kullanılan Process API sınıfını içe aktarın.
- Satır 2: Paketin yürütülmesi sırasında analizciye sorunları bildirmek için kullanılan CuckooPackageError istisnasını içe aktarın.
- Satır 4: Ana sınıfı tanımlayın, object sınıfından miras alacak şekilde.
- Satır 5: path adlı dosyanın yürütülmesini sağlayan start() fonksiyonunu tanımlayın. Her analiz paketi tarafından uygulanmalıdır.
- Satır 8: check() fonksiyonunu tanımlayın.
- Satır 13: Sürecin izlenip izlenmeyeceğini tanımlayan free seçeneğini edinin.
- Satır 18: Bir Process örneğini başlatın.
- Satır 19: Zararlı yazılımı çalıştırmaya çalışın, başarısız olursa yürütmeyi durdurun ve analizciye bildirin.
- Satır 24: Sürecin izlenip izlenmemesi kontrol edilir.
- Satır 25: Süreci DLL ile enjekte edin.

- Satır 26: Süreci askıya alınmış durumdan devam ettirin.
- Satır 28: Yeni oluşturulan sürecin PID'sini analizciye döndürün.
- Satır 30: finish() fonksiyonunu tanımlayın.
- Satır 31: procmemdump seçeneğinin etkin olup olmadığını kontrol edin.
- Satır 32: Şu anda izlenen süreçler arasında döngü yapın.
- Satır 33: Bir Process örneğini açın.
- Satır 34: Sürecin belleğinden bir döküm alın."

start()

Bu fonksiyonda çalıştırmak istediğiniz tüm başlatma işlemlerini yerleştirmelisiniz. Bu, zararlı yazılım sürecini çalıştırmayı, ek uygulamaları başlatmayı, bellek anılarını almayı ve daha fazlasını içerebilir.

check()

Bu fonksiyon, zararlı yazılım çalışırken Cuckoo tarafından her saniye çalıştırılır. Bu fonksiyonu, her türlü tekrarlayan işlemi gerçekleştirmek için kullanabilirsiniz.

Örneğin, analizinizde yalnızca belirli bir gösterge oluşturulmasını bekliyorsanız (örneğin, bir dosya), koşulu bu fonksiyona yerleştirebilir ve False döndürürse analiz hemen sona erer.

Bu, "analiz devam etmeli mi yoksa durmalı mı?" olarak düşünülebilir.

Örneğin:

```
def check(self):
    if os.path.exists("C:\\config.bin"):
        return False
    else:
        return True
```

Bu `check()` fonksiyonu, `C:\\\\config.bin` oluşturulduğunda Cuckoo'nun analizi derhal sonlandırmasına neden olacaktır.

execute()

Kötü amaçlı yazılım yürütmeyi tamamlar ve DLL enjeksiyonu ile ilgilenir.

finish()

Bu fonksiyon, analizi sonlandırmadan ve makineyi kapatmadan önce Cuckoo tarafından basitçe çağrılır. Varsayılan olarak, bu fonksiyon tüm izlenen süreçlerin belleğini dökmek için isteğe bağlı bir özelliği içerir.

Ayarlar

Her paket, otomatik olarak tüm kullanıcı tarafından belirtilen seçenekleri içeren bir sözlüğe erişime sahiptir (bkz. Analiz).

Bu tür seçenekler, `self.options` özniteliğinde kullanılabilir hale getirilir. Örneğin, kullanıcının gönderim sırasında aşağıdaki dizesini belirttiğini varsayalım:

```
foo=1,bar=2
```

Seçilen analiz paketi, bu değerlere erişim sağlayacaktır:

```
from lib.common.abstracts import Package

class Example(Package):

    def start(self, path):
        foo = self.options["foo"]
        bar = self.options["bar"]

    def check():
        return True

    def finish():
        return True
```

Bu seçenekler, paketinizin içinde yapılandırmanız gerekebilecek her şey için kullanılabilir.

Process API

Process sınıfı, çeşitli işlemle ilgili özelliklere ve işlevlere erişim sağlar. Analiz paketlerinizde şu şekilde içe aktarabilirsiniz:

```
from lib.api.process import Process
```

Daha sonra bir örneği şu şekilde başlatırsınız:

```
p = Process()
```

Bir yeni süreç oluşturmak yerine mevcut bir süreci açmak istiyorsanız, birden fazla argüman belirleyebilirsiniz:

- pid: Çalışmak istediğiniz sürecin PID'si.
- h_process: Çalışmak istediğiniz sürecin işlem tanıtıcısı.
- thread_id: Çalışmak istediğiniz sürecin thread ID'si.
- h_thread: Çalışmak istediğiniz sürecin thread işlem tanıtıcısı.

Metotlar

Process.open()

Bir çalışan süreç için bir işlem tanıtıcısı açar. İşlemin başarılı olup olmadığı durumunda True veya False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
p = Process(pid=1234)
p.open()
handle = p.h_process
```

Process.exit_code()

Açılan sürecin çıkış kodunu döndürür. Daha önce yapılmadıysa, exit_code() işlem tanıtıcısını edinmek için open() çağrısını gerçekleştirir.

Return type:	ulong

Örnek kullanım:

```
p = Process(pid=1234)
code = p.exit_code()
```

Process.is_alive()

exit_code() fonksiyonunu çağırır ve dönen kodun STILL_ACTIVE olduğunu kontrol eder, bu da verilen sürecin hala çalıştığı anlamına gelir. True veya False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
```

```
p = Process(pid=1234)
if p.is_alive():
    print("Still running!")
```

Process.get_parent_pid()

Açılan sürecin üst sürecinin PID'sini döndürür. Daha önce yapılmadıysa, get_parent_pid() işlem tanıtıcısını edinmek için open() çağrısını gerçekleştirir.

Return type:	int

Örnek kullanım:

```
p = Process(pid=1234)
ppid = p.get_parent_pid()
```

Process.execute(*path*[, *args=None*[, *suspended=False*]])

Belirtilen yol üzerindeki dosyayı çalıştırır. İşlem başarılı olursa True, başarısız olursa False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
p = Process()
p.execute(path="C:\\WINDOWS\\system32\\calc.exe", args="Something",
suspended=True)
```

Process.resume()

Açılan süreci askıya alınmış durumdan devam ettirir. İşlem başarılı olursa True, başarısız olursa False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
```

```
p = Process()
p.execute(path="C:\\WINDOWS\\system32\\calc.exe", args="Something",
suspended=True)
p.resume()
```

Process.terminate()

Açılan süreci sonlandırır. İşlem başarılı olursa True, başarısız olursa False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
p = Process(pid=1234)
if p.terminate():
    print("Process terminated!")
else:
    print("Could not terminate the process!")
```

Process.inject([*dll*[, *apc=False*]])

Açılan sürece DLL'imizi enjekte eder. İşlem başarılı olursa True, başarısız olursa False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
p = Process()
p.execute(path="C:\\WINDOWS\\system32\\calc.exe", args="Something",
suspended=True)
p.inject()
p.resume()
```

Process.dump_memory()

Verilen sürecin bellek alanının bir anlık görüntüsünü alır. İşlem başarılı olursa True, başarısız olursa False değerini döndürür.

Return type:	boolean

Örnek kullanım:

```
p = Process(pid=1234)
p.dump_memory()
```


Revision #1
Created 19 January 2024 11:47:05 by Ertan Sözer
Updated 19 January 2024 11:53:06 by Ertan Sözer