Executable Packing

Zararlı yazılımın analizini doğru şekilde gerçekleştirebilmek için analize başlarken birkaç sorunun cevaplanması gerekiyor. Bunlar;

- Zararlı yazılım gerçekten paketlenmiş mi?
    - Paketlenmişse kanıtları neler?
- Zararlı yazılım **Self-Injection** mı yoksa **Remote-Injection** mı yapıyor?
- Zararlı yazılım Sefl-Overwriting yapıyor m?
- Payload nereye yazılıyor?
- Payload nasıl yürütülecek?
- Unpacking işleminden sonr aorjinal koda ulaşıldığına dair kanıtlar neler?
    - Zararlı yazılım tekrar tekrar (iç içe) paketlenmiş olabilir. İkinci kez paketlendiğine dair kanıtlar bulunuyor mu?

Cevaplanması gereken sorular doğrultusunda ilk soru olan zararlı yazılımın gerçekten paketlenip paketlenmediğine karar verebilmek için paketleme süreci hakkında bilgi sahibi olmak gerekiyor. Öğrenmeye paketleme yazılımlarının çıkış nedenini açıklamakla başlayabiliriz.

Paketleme Nedir Ve Neden Kullanılır?

Paketleme, çalıştırılabilir dosyaların fonksiyonlarını değiştirmeden sıkıştırma işlemidir. Paketleme işlemi disk alanlarının sınırlı ve değerli olduğu zamanlarda disk alanından tasarruf edebilmek için geliştirilmişti.

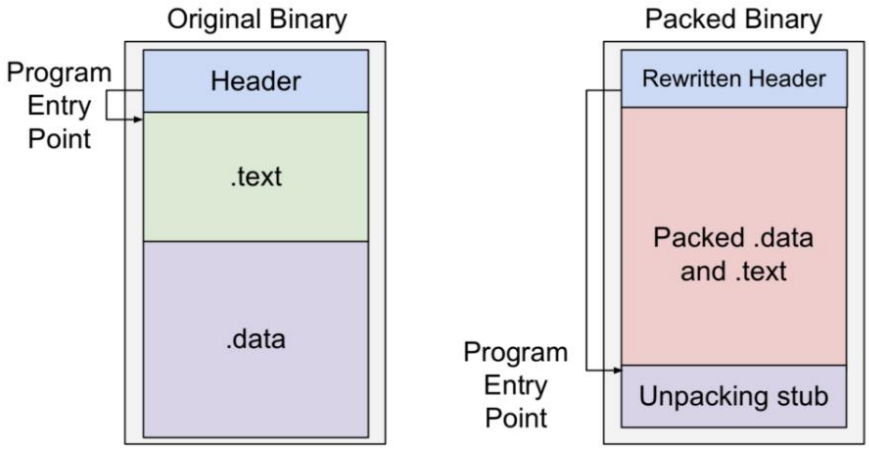
Paketleme işlemi günümüzde çalıştırılabilir dosya boyutlarını küçültmek ve bu dosyaları tersine mühendislik uygulamalarına karşı korumak için kullanılıyor. Legal uygulamalar için bu özellikler her ne kadar faydalı görünse de zararlı yazılımlarda kullanıldığında zararlı yazılımın tespit ve analiz süreçlerini zorlaştırmaktadır.

Program disk üzerine sıkıştırılmış olarak kayıt edilse de çalıştırıldığı zaman uygulama paket içerisinden çıkarılarak hafıza alanına orijinal haliyle yüklenir. Bu süreçte disk üzerinde bir tür sıkıştırılmış/şifreli durumda tutulup çalıştırılacağı orijinal haliyle belleğe yüklenmesinden kaynaklı paketleme yazılımlarının bir anlamda Crypter olarak anılmasına da neden olmaktadır.

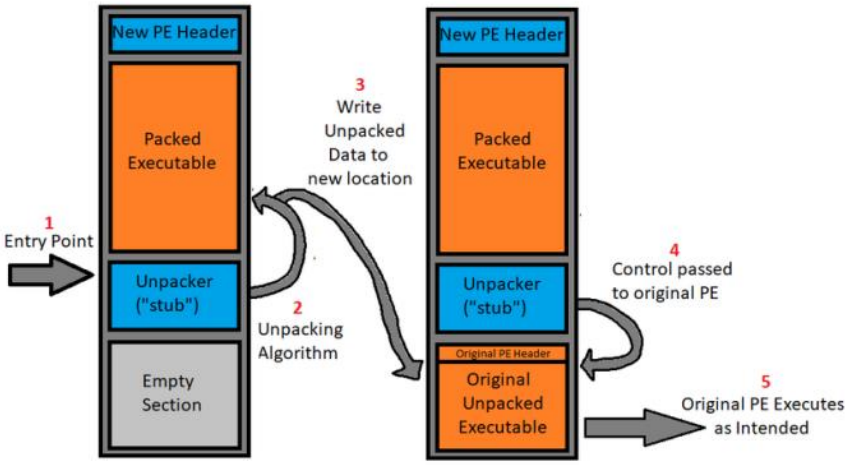
Paketleme Yapısı Nasıl Çalışır?

Paketleme sürecinde uygulamaların analizi zorlaştırabilmek için özel paketleme yazılımları geliştirilebiliyor. Dolayısıyla farklı mekanizmalar kullanılabiliyor. Bu nedenle her paketleme yazılımında aynı olmasa da çalışma yapısına örnek olarak yaygın kullanılan/bilinen UPX paketleme yazılımında kullanılan Stub-Payload mekanizması verilebilir (Bilindik paketleme algoritmaları genel olarak aynı temel prensibe göre çalışıyor).

Stub-Payload mekanizması, çalıştırılabilir program kodunu yeni bir çalıştırılabilir program yapısı içinde alma temeline dayanır. Bu temelde oluşturulan yeni çalıştırılabilir dosya yapısının içerisinde şifrelenmiş/sıkıştırılmış dosyanın kendisi ve sıkıştırılmış programı çıkarmak/deşifre etmek için küçük bir kod parçasının koyulduğu (Stub) iki kısım bulunur.



Paketlenmiş dosya çalıştırılmak istendiğinde normal çalıştırılabilir dosyadan farklı olarak orijinal çalıştırılabilir dosyayı deşifre edecek Stub noktasından itibaren çalıştırılmaya başlayacaktır. Stub noktasından orijinal veriyi deşifre edilerek farklı bir bellek alanına yazılır. Çalıştırılabilir programın orijinal hali elde edildikten sonra sağlıklı bir şekilde yürütülecektir.



Bir Uygulamanın Paketlenip Paketlenmediğini Nasıl Anlaşılır? (Belirtileri nelerdir?)

Bir zararlı yazılımın analizine başlanırken öncelikle zararlı hakkında genel anlamda bilgi toplanmaya çalışılır. Bu süreçte Virus Total, Hybrid Analysis gibi Online yazılımlar kullanılabileceği gibi çeşitli Offline yazılımlar da kullanılabiliyor. Bu yazılımlar sayesinde zararlının **Hash bilgileri**, **hangi zararlı ailesine ait olduğu**, **tespit edilebilen fonksiyon isimleri** gibi bilgiler elde edilebiliyor ve bu bilgiler doğrultusunda zararlının paketlenip paketlenmediğine (paketlenmişse hangi paketleyiciyle paketlenmiş olabileceğine) dair bir fikir verebiliyor. Bu fikir doğrultusunda çalıştırılabilir dosyanın gerçekten paketlenip paketlenmediğini anlayabilmek için zararlıyı inceleme sürecinde göz önünde bulundurulması gereken bazı durumlara bakıldığında;

- Normal bir çalıştırılabilir dosyaya nazaran daha az sayıda DLL ve fonksiyon Import edilecektir.
- Yüksek miktarda karartılmış/anlamsız metin/String tanımı bulunacaktır.
    - Şifrelenmiş ve açık metin (anlamlı) String değerleri karıştırılmış olabilir. Bu durumda çalıştırılabilir dosyanın paketlenip paketlenmediğine karar vermek biraz zor olacaktır.
- Bir çalıştırılabilir dosyada bulunan standart bölümler dışında bölümler içeriyor olabilir.

- Yaygın olmayan/anormal çalıştırılabilir bölümlere sahip olması (**sadece .text/.code bölümü yürütülebilir olmalıdır**).
- Belirli/bilindik sistem çağrılarına sahip olabilir.
- Beklenmedik yazılabilir bölümlere sahip olabilir.
- Yüksek Entropi değerlerine sahip bölümler içeriyor olabilir.
- Çalıştırılabilir dosyanın sahip olduğu bölümlerden herhangi birinin ham boyutu ile sanal boyutu arasında farka sahip olabilir.
- Sıfır bayt boyutuna sahip bölümler içeriyor olabilir.
- Olağan dışı dosya formatı ve başlıkları içeriyor olabilir.
- Network üzerinden bağlantılar kuruluyor olabilir. Bu süreçte kullanılması gereken API'ler eksik olabilir.
    - GetProcAddress() ve akabinde takip eden LoadLibrary() fonksiyonları kullanılarak API'ler ayrıca dinamik olarak çözümlenebilir.
    - Benzer şekilde zararlı yazılımın amacını gerçekleştirmek için gerekecek API'lerin eksik olması bir ipucu olabilir (Dinamik analizde gerçekleştirdiği aktiviteyi gerçekleştirmek için kullandığı fonksiyolar eksik olabilir).
- Alışılmadık dosya formatına ve başlıklara sahip olabilir.
- .text/.code bölümünden farklı bir bölümü başlangıç noktası olarak gösteriyor olması bir ipucu olabilir.
    - Başlık bilgileri yanlış biçimlendirildiğinde ilk analizde tespit edilmesi zor olabiliyor.
- Zararlı yazılım IDA gibi bir Disassemble yazılımında açıldığında büyük miktarda çözümlenememiş veri olduğunu görmek bir ipucu olabilir.
- Resource bölümünün (.rsrc) boyutu ve ardından kod içerisinde bulunan LoadSource() fonksiyonunun işlevi önemli olabilir.

Unutulmamalıdır ki bir çalıştırılabilir dosyanın bunlardan birini veya birkaçını sağlaması paketlenip paketlenmediğini kesinleştirmek için yeterli değildir. Paketleme yazılımında kullanılan tekniklere göre bu belirtiler farklılık gösterecektir. Yukarıdaki göz önünde bulundurulması gereken listede bırakılan alt açıklamalarda da olduğu gibi bu belirtiler zararlının analizini zorlaştırmak için uygulanan tekniklerden kaynaklanabileceği gibi paketlenmemiş bir uygulama kodunda da görülebilmektedir.

Paketlenmiş Programı Çıkarırken Karşılaşılabilecek Zorluklar

- Çeşitli Anti-Debugging teknikleri kullanılıyor olabilir.
    - Bunun için Anti-Debugging eklentileri bulunan araçların kullanılması gerekebiliyor. Örnek olarak **X64dbg/x23dbg** aracının **ScyllaHide** eklentisi veya **OllyDbg** aracını **StrongOD** eklentisi gibi daha birçok araç örnek olarak verilebilir.
- Anti-VM teknikleri kullanıyor olabilir (VMWare, Virtualbox gibi çeşitli sanallaştırma yazılımları üzerinde çalıştırılıp çalıştırılmadığını belirlemek için işletim sistemi üzerindeki çeşitli özellikleri kontrol ediyor olabilir). Anti-VM tekniklerine örnek olarak;
    - Üzerinde çalıştığı bilgisayarın adı, çalıştırılan dosyanın adı (dosyanın ismi Hash bilgisi koyulmamalıdır), oturum açılan kullanıcı adı kontrol ediliyor olabilir.
    - Sanal makineye ayrılan kaynak miktarı kontrol ediliyor olabilir.
        - Örnek olarak disk miktarı veya işlemci sayısı kontrol ediliyor olabilir. Bu nedenle sanal makinaya en azından 100GB'dan disk alanı ve 2 çekirdek ayrılması faydalı olacaktır. Özetle analiz yapılacak sanal makine Bare-Metal yapıya olabildiğince benzer şekilde oluşturulmalıdır.

    - Analiz yapılacak sanal makinanın açık kalma süresi kontrol edilebilir. Bu nedenle sanal makinenin görüntüsü alınırken en azında 20 dakikanın üzerinde çalışma süresine sahip olmalıdır.
- Analiz sürecini zorlaştırmak için birçok sahte/anlamsız sistem çağrıları veya API'ler içerebilir.
- Exception Handlers Anti-Debugging tekniği gibi kullanılabilir.
- Anti-Breakpoint teknikleri kullanılıyor olabilir.
- Analiz sürecinde kullanılan çeşitli araçların (Proccess Hacker, Proccess Monitor, PEStudio gibi) yüklü olup olmadığı kontrol ediliyor olabilir. Buna çözüm olarak analiz sürecinde kullanılacak araçlar çalıştırmadan önce isimleri değiştirilebilir.

Analiz sürecinde karşılaşılabilecek zorluklar bunlarla sınırlı değildir. Gün geçtikçe bu zorluklara bir yenisi daha ekleniyor. Bu nedenle her bir analizde bakış açımızı değiştirerek karşılaşılan zorlukların üstesinden gelmenin bir yolunu bulmamız gerekecek.

Well-Known Pakcer's Anti-Forensic and Obfuscation Characteristic

Paketleme yazılımlarında paketlenecek program kodunun analizini zorlaştırmak adına çeşitli Anti-Forensic ve Obfuscation tekniklerinin kullanıldığından bahsedilmişti. Bu teknikler her ne kadar paketleme yazılımını geliştirenlerin yeteneğine bağlı da olsa yaygın kullanılan paketleme yazılımlarıyla paketlenen zararlı yazılımların ortak karakteristik özelliklerine bakıldığında;

- 64 bitlik ikili dosyaları kullanırlar.
- IAT (Import Address Table) tabloları kaldırılmış olabilir veya tablo import edilen tek bir fonksiyon bulunabilir.
    - IAT tablosu, program çalıştırıldığında dinamik bağlanan kütüphanelerden (DLL'lerden) gelen fonksiyonların adreslerinin kaydedildiği tablodur.
- Çoğu zaman olduğu gibi program içeriği şifrelenmiştir.
- Bellek bütünlüğü korunur ve kontrol edilir. Bu nedenle orijinal kodun tamamını açık bir şekilde çözümlemek ve görüntülemek mümkün değildir.
- Komutlar sanallaştırılmıştır ve RISC komutlarına dönüştürülmüştür. Bu komutlar bellekte de şifreli tutulacaktır.
- Obfuscation Stack tabanlıdır. Bu nedenle statik yaklaşım kullanılarak kodu çözümlemek zordur.
- Sanallaştırılmış kodun çoğu Polymorphic'tir. Bu nedenle sanallaştırılmış kodun çoğu aynı orijinal komutu referans gösterir.
- Çok fazla sahte Push komutu (Stack'e veri depolamak için kullanılıyor) içerecektir. Tabi ki bunların çoğu kullanışsız ve ölü kod satırından ibarettir.
- Koşulsuz dallanma komutları kullanılarak kodun parçalar halinde yeniden sıralanmasını sağlanabilir.
- Modern paketleme yazılımların hepsi Code Flattening, birçok Anti-Debugging ve Anti-VM teknikleri kullanır.
- X64 komutların hepsi sanallaştırılmamıştır. Dolayısıyla sanallaştırılmış ve sanallaştırılmamış komutların karışımından oluşan kod bulunabilir. Bu durum analizi zorlaştıracaktır.
- Çoğu zaman fonksiyonların **Prologue** ve **Epilogue** aşamaları sanallaştırılamaz.
    - **Prologue**, yeni bir program çalıştırıldığında Stack alanını kullanmadan önce program başlangıcına ilişkin bir tür işaret bırakması için uygulanan adımlar bütünü olarak tanımlanabilir. Bu adımlara bakıldığında;

    - **Epilogues**, Prologue sürecinde gerçekleştirilen adımların tersinin gerçekleştirildiği (Stack yapısı program başlatılmadan önceki haline getirilir) süreçtir. Program sonlamadan önce gerçekleştirilir. Gerçekleştirilen adımlara bakıldığında;

```
push ebp
mov ebp, esp
sub esp, N
```
Prologue

```
mov esp, ebp
pop ebp
ret
```
Epilog

- Orijinal kod parçası, parçalara ayrılmış ve program geneline dağıtılmış olabilir. Böylece komutlar ve veriler karışacaktır.
- Import edilen fonsiyonlara ilişkin komutlar sıfırlanabilir veya "nop" (bekleme) komutuyla değiştirilebilir. Bu durumda fonksiyonlar dinamik olarak yüklenecektir.
    - Import edilecek adresleri sıfırlamak veya "nop" komutları yerleştirmek yerine RVA (Relative Virtual Address) kullanılarak dallanma komutları da konulabilir. Bu teknik "**IAT Obfuscation**" olarak da biliniyor.
- Shell Code'da kullanılabildiği gibi API adları da Hash halde oluşturulabiliyor.
- Birçok Native API, çağrıyı ileten Stub koda yönlendirir.
- Genelde **Constant Unfolding**, **Pattern-Based Obfuscation**, **Control Indirection**, **Inline Functions**, **Code Duplication** ve **Mainly Opaque Predicate** gibi karartma teknikleri kullanılır.

Notlar

- Shifting Decode Frame Technique
- API/DLL Resolving and Techniques

Kaynaklar

- https[:]//web.fe.up.pt/~ei08072/lib/exe/fetch.php?media=unpacking_gaspar_furtado.pdf
- https[:]//exploitreversing.files.wordpress.com/2021/12/mas_1_rev_1.pdf
- https[:]//medium.com/ax1al/packing-and-obfuscation-fe6b03bbc267
- https[:]//www.linkedin.com/pulse/packed-malware-basics-halashankara-k
- https[:]//www.slideshare.net/IOSR/l016117177
- https[:]//resources.infosecinstitute.com/topics/malware-analysis/top-13-popular-packers-used-in-malware/
- https[:]//github.com/packing-box/awesome-executable-packing/blob/main/README.md
- https[:]//www.darkreading.com/application-security/unpacking-packed-malware
- https[:]//www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/basic-packers-easy-as-pie/
- https[:]//www.goggleheadedhacker.com/blog/post/6
- https[:]//www.trendmicro.com/en_id/research/23/h/targetcompany-ransomware-abuses-fud-obfuscator-packers.html
- https[:]//www.shogan.co.uk/devops/packing-executable-files-to-reduce-distribution-size-with-upx/
- https[:]//copyprogramming.com/howto/possible-to-detect-packed-executable
- https[:]//arxiv.org/pdf/2302.09286.pdf
- https[:]//alpbatursahin.medium.com/portable-executable-pe-dosya-format%C4%B1-%C3%BCzerinden-malware-analizi-d3179b9ef353
- https[:]//www.ired.team/offensive-security/defense-evasion/windows-api-hashing-in-malware