DLL의 개념과 보안 취약점 분석

SWING 32기 이유진

서론

지난 4주차 포렌식 과제 EVTX 분석을 위해 파일을 하나 골라 공부하던 중이었다.

나는 RUNDLL32라는 파일을 택하였다. MITRE에 따라 정리한 사이트의 설명을 보면,

'RUNDLL32파일이란 Windows 운영체제에서 일반적으로 DLL 페이로드 실행과 관련이 있는 파일을 칭함'이라 적혀 있다. 이는 DLL 파일 내부의 함수를 호출하여 특정 작업을 수행한다는 뜻이다.

공격자는 해당 파일을 활용하여 **악성 DLL을 로드**하거나, 정상적인 DLL로 가장해 **악성 코드를 실행**함으로써 보안 공격 기법으로 활용 가능하다고 한다.

사실 DLL이라는 용어는 여태까지의 보안 공부에서 종종 보았던 단어였지만, 항상 이게 정확히 무엇인지 이해하지 못한 채 라이브러리의 일부구나라고 이해만 했다는 생각이 들었다.

그래서 지난 4주차 과제때에는 가볍게 넘어갔지만, 이번 SSR에서 DLL이 무엇이고 어떻게 악성 공격이 가능한지에 대해 세부적으로 공부해보고자 한다.

본론

우선 DLL이 무엇인지부터 조사를 시작했다.

지난 포렌식 과제 시 '동적 링크 라이브러리'라고까지는 조사를 했었다.

'동적'이라는 키워드는 내가 웹프로그래밍 과목을 수강할 때 나왔던 것을 연상하게 해,

'시스템 상에서 알아서 처리하는 링크 연결에 대한 함수 같은 정보를 담는 저장소'라고

한 번 추론을 하고 인터넷 조사를 시작했다.

[0 라이브러리]

라이브러리라는 키워드부터 정확히 알아보자. 라이브러리란 표준화된 기본적인 함수들이나 데이터 타입들을 모아둔 저장소이다. 이는 프로그램 사용자가 굳이 따로 함수들을 만들지 않도록 효율성을 높이는 동시에, 기본 함수들을 만들다가 중복 작성하지 않도록 방지해주는 역할을 해주기도 한다.

그럼 동적이란 키워드는 무엇일까?

이게 바로 라이브러리 종류를 나누는 기준이 될 것이다.

라이브러리는 언제 메인 프로그램에 연결되느냐에 따라 크게 [동적(DYNAMIC) 링크]와 [정적(STATIC)링크]로 나뉜다.

[STATIC 링크 라이브러리]: 컴파일을 하는 그 순간에 라이브러리가 링크에 연결되는 것. 즉 라이브러리 자체가 실행파일의 부분으로 들어가게 된다.

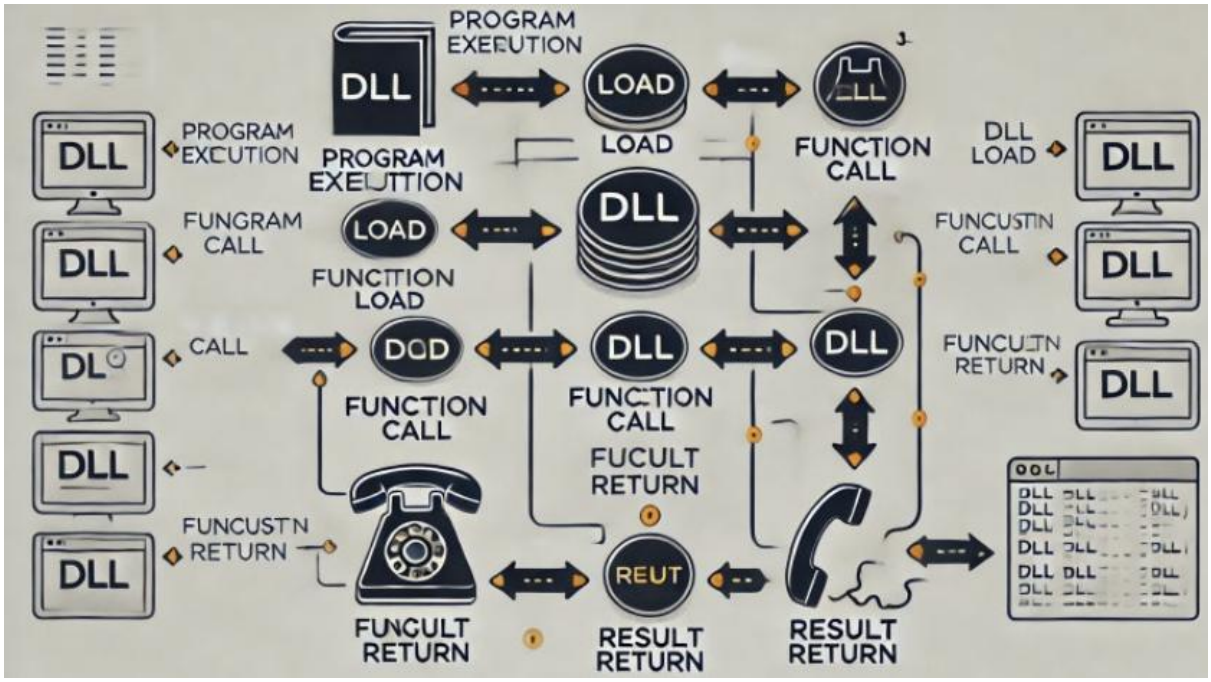
[DYNAMIC 링크 라이브러리(=DLL)]: 컴파일 시가 아닌 실행 중에 함수의 위치정보를 가지고 라이브러리 내 함수를 호출하는 한다.

앞에서 했던 내 추론을 다시 한 번 보면, '알아서 처리'라는 말 보다는 '함수를 따로 불러오는 것'으로 이해하는 게 더 적절하다고 수정할 수 있다.

[1 DLL의 기능]

DLL은 각 코드에서 따로 라이브러리 내용을 불러오는 것이라 앞에서 정의했다.

그럼 DLL의 구체적인 작동원리는 어떻게 될까?



프로그램이 실행되면, 실행 시 필요한 DLL들이 메모리에 로드된다. 이후 로드된 DLL을 활용하며 프로그램이 시작되고, 중간중간 필요한 DLL들을 가져와 메모리에 적재하고 사용한다. 이때 DLL은 마치 동적 링크 라이브러리가 아닌 정적처럼 정말 프로그램의 일부처럼 DLL이 작동한다.

프로그램이 종료되거나 혹은 해당 DLL이 더 이상 필요가 없다고 판단하게 되면, 메모리에서 해제하여 효율성을 유지한다.

[2 DLL의 장단점]

추가로 이에 따른 장점을 알아보자.

가장 기본적으로 DLL은 메모리, 디스크 상 공간 활용이 더 절약적이다는 점이다.

특히 각 파트로 나뉘어서 코드를 작성하는 모듈식 아키텍처의 경우, 작동이 더욱이 효율적으로 기능할 수 있다.

다음으로 단점은 어떤 게 있을까.

우선 DLL의 중요 특징인 '종속성'에 대해 알아야한다.

내가 '자바 프로그래밍'이라는 과목을 수강하면서도 종속성이라는 특징이 나온 적 있다.

다른 언어와 자바를 비교할 때, 다른 언어는 각 OS에 맞는 실행파일을 통해서만 프로그램 실행이 가능하다면, 자바는 반대로 '독립적'이라는 특징을 가져 모든 OS에 적용이 가능하다는 이야기였다.

그렇다면 여기 DLL의 특징 '종속성'은 어떤 뜻일까?

한 프로그램이 실행될 때, 코드에서 불러오는 함수가 어떤 특정한 DLL에 포함되어 있고 그걸 불러와야하는 구조이며, 만약 DLL이 없을 경우 실행 에러가 나는 경우를 우리는 DLL 종속성을 가지는 구조다라고 판단하는 것이다.

즉, 앞 OS에서 실행파일이 맞지 않으면 실행파일 기능을 불러올 수 없는 것처럼, 코드에서 맞는 DLL도 없으면 역시나 작동이 불가능 한 순간을 '종속적'이라 칭한다.

이 종속성 특징으로 인해, 프로그램이 특정 DLL파일을 사용하기 위해서 특정 위치에 저장을 해야 할 경우가 생긴다. 그래서 만약 이 위치가 잘못되기라도 하면, 종속성 때문에 에러가 난다는 DLL만의 단점이 있다.

[3 DLL 공격 종류]

DLL이 무엇인지 배웠다면, 이제 DLL을 활용한 취약점 공격 방식에는 무엇이 있는지 배워보자.

DLL 활용 공격 방식은 아래 표와 같이 나눌 수 있다.

분류	공격 기법	악용 방식
직접 실행	Rundll32.exe	Rundll32.exe를 이용하여 DLL을 직접 실행
Injection	DLL Injection	실행 중인 다른 프로세스의 공간에 강제로 DLL을 Injection 하는 방식
	Reflective DLL Injection	프로세스의 메모리에 임의의 DLL에 대한 데이터를 삽입한 후 직접 매핑 (Mapping) 하여 실행시키는 방식
Hijacking	DLL Hijacking	합법적인 실행파일의 정상 DLL인것 처럼 실행하는 방식
	DLL side-loading	Search Order Hijacking과 DLL Proxy를 이용하여 원하는 DLL를 실행하는 방식

[3-1 직접 실행]

첫 번째는 직접 실행함으로써 공격하는 방식이다. 저번 포렌식때 내가 선택한 조사파일이 rundll32.exe라는 파일이었는데, 이 파일이 바로 첫번째 방식에 해당한다.

이 rundll32.exe 파일을 활용해서 공격할 수 있는 방식은 4가지가 존재한다.

1. 정상적인 함수를 사용해 보안 단계를 우회하기

2. 정상적인 DLL 혹은 import 함수를 악성 행위를 수행하도록 이용하기

3. 공격자가 악성 DLL을 사용자 시스템 내에서 다운로드하고 rundll32.exe 실행하기

4. 기존에 있던 정상적인 DLL을 다른 악의적 DLL로 교체하기

1번과 2번 방식은 조금 비슷해보여 정리해볼 필요가 있다 생각이 들었다.

1번은 단어 '우회'에 집중해야 한다. 1번과 2번 둘 다 정상 함수를 사용한다는 공통점이 있지만, 1번은 그 정상함수를 이용해 악성 스크립트나 코드를 가져오려는 것이다.

스크립트, 코드로 우회하여 실행을 시킨다면 반면 2번은, 정상 함수 기능을 활용해 악성 코드를 바로 가져오는 것이다. 좀 더 직관적인 예시를 들자면, 정상 DLL에서 바로 악성 URL을 가져와 명령하는, 우회가 아닌 직접 실행이 목적인 방식이다.

다시 돌아와 이 직접 실행이라는 방식을 한 번 정리해보자면,

이 rundll32.exe 파일은 운영체제 윈도우의 필수 구성파일로, 시스템을 훼손하지 않는 이상은 이 파일을 차단/비활성화 할 수 없다. 즉 위에서 나온 4가지 방식대로 공격자는 이 필수파일을 악용하여 정상작업인 척 credential access (자격 증명 도용)실행 및 우회 수

단으로 이 파일을 사용한다는 것이다.

실행파일로 악성코드를 돌릴 때에 비해 DLL로 악성코드를 돌리게 되어 감지가 매우 어려운 방식이다.

이는 매우 간단한 DLL 공격 방식으로, DLL 경로와 악성 코드를 돌릴 DLL 진입점만 알고 있으면 가능한 방식이라고 한다

[3-2 HIJACKING]

두 번째 공격방식은 hijacking 방식이다.

윈도우 운영 체제가 특정 프로그램을 실행할 때 필요한 **DLL 파일을 찾지 못하거나** 잘못된 경로에서 로드하여 **악성 DLL**이 실행되는 공격 기법이야.

이 방식은 실행파일로 악성코드를 실행하는대신, 정상적 어플리케이션을 거쳐 악성코드를 실행하도록 하는 방법이다. 즉 악성코드 감지에 대한 우회가 가능하다는 것이다.

하이재킹 방식은 방어회피, 지속성, 권한 상승을 공격의 주 목표로 한다.

하이재킹 방법은 4가지로 나뉜다고 한다.

1. Search order hijacking

: 가장 잘 알려진 DLL Hijacking 예시이다. 공격자는 윈도우 운영 체제에서 실행을 위한 DLL 검색 시 사용하는 검색 방식을 악용하는 방식이다. 정상적 검색 프로세스에서 악성코드를 실행하며 악성코드를 숨기게 된다.

*여기서 '검색'이라는 키워드가 나왔는데, 이 검색은 우리가 직접하는 게 아니라 윈도우 운영체제가 프로그램 작동을 위해 필요한 DLL 파일을 로드하는 과정을 칭한다.

2. Relative path DLL hijacking

1번 Search Order Hijacking의 변종이다.

1번 방식이 검색 프로세스에서 숨기는 거라면, 이 방식은 공격자가 쓰기 권한이 있는 폴더에 악성 DLL을 숨기는 방식이다.

정상 파일과 악성 DLL을 함께 붙여 정상적인 실행 파일의 이름을 변경하여 생성하는 경우라 말할 수 있다.

예를 들어, 메모장 앱 notepad.exe을 보자. 이건 쓰기 권한이 있는 파일로, 메모장이 불러오는 DLL 정보를 미리 확보한다. 그리고 notepad.exe가 위치한 경로가 아닌 전혀 다른 경로에 Dll을 옮긴 이후, 메모장 파일명을 바꾸고 악성 DLL을 같은 폴더에 넣어 실행하는 방식이다. 이렇게 악성 DLL을 숨길 수 있다.

3. Phantom DLL hijacking

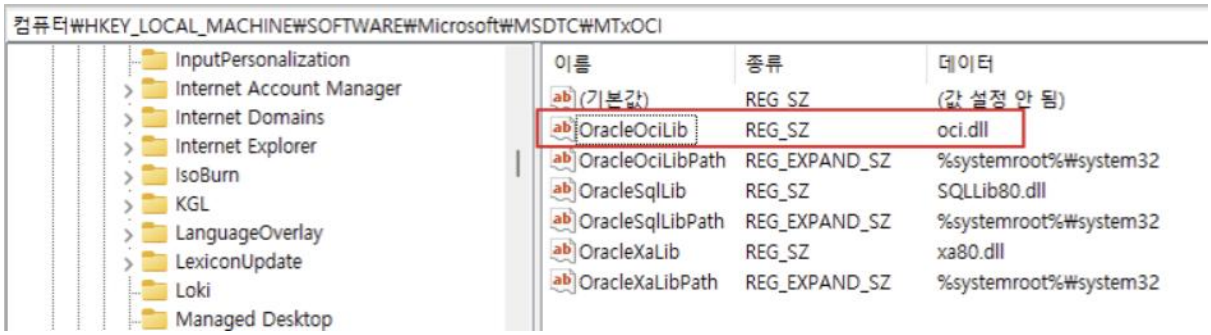
윈도우 운영 체제는 존재하지 않는 DLL 파일을 의외로 많이 참조한다고 한다. 그 이유는 다양한 버전과의 호환성 때문이다. 예시로 kernel32.dll라는 파일은 현재 사용하지는 않고 다른 DLL로 같은 기능을 수행하지만, 여전히 참조할 때는 과거 이름인 kernel32.dll을 사용하고 있다. 과거 버전을 사용하면 과거 DLL을 요구할 수 있기에 만약을 대비해 참조하는 것이다.

DLL 공격에서는 이 점을 이용하여 공격자가 이러한 누락된(존재하지 않는(~사용하지 않는)) DLL 중 하나를 지정하여 악성 DLL을 작성한다. 운영 체제에서는 해당 파일을 참조하는 코드를 실행할 때 이 악성 DLL이 로드된다.

4. DLL redirection

공격자는 윈도우에서 이미 정해져있던 DLL 검색 순서를 활용하는 방식이다. 윈도우 운영 체제가 DLL 파일을 검색하는 위치를 변경한다.

이 파트는 내가 이해가 어려워 블로그에 나온 예시를 함께 들어 정리해보고자 한다.



위 사진 파일을 보면 oci.dll이라는 파일이 보인다. 이 파일은 OracleOcilLib의 키 값이다.

원래는 C:₩windows₩system32₩oci.dll의 경로를 따라 oci.dll 파일을 사용해야한다.

하지만 여기서 oci.dll을 공격자가 원하는 악성 DLL로 변경한 이후, 다음경로 "HKEY_LOCAL_MACHINE₩SOFTWARE₩Microsoft₩MSDTC₩MTxOCI₩OracleOciLib"에 있는 레지스터리에 접근하여 OracleOciLib 키를 우리가 앞에서 변경한 악성 DLL의 이름으로 변경한다. 이제 변경 이후 MSDTC 서비스를 재시작하면 악성 DLL 실행된다.

*MSDTC란?

윈도우 운영체제에서 분산 트랜잭션을 관리하는 서비스를 뜻함.

여러 개의 트랜잭션이 오류 없이 잘 작동하도록 도와준다.

*트랜잭션: 데이터베이스/시스템에서의 일련의 작업들을 하나로 묶어 처리하는 것

[3-3 INJECTION]

마지막 세 번째 방식은 내가 가장 궁금했던 인젝션 방식이다. 인젝션이란 단어는 보안 공부를 하며 가끔 들어본 용어같아 무슨 기법인지 호기심이 생겼었다.

DLL INJECTION 방법은 실행 중인 다른 프로세스의 공간에 강제로 특정 DLL을 Injection(=삽입)하는 방법이다.

공격방법은 3가지가 존재한다.

1) 원격 스레드 생성(CreateRemoteThread() API)

2) 레지스트리 이용 (AppInit_DLLs 값)

3) 메시지 후킹 (SetWindowsHookEx() API)

그 중에서도 1번 방식을 살펴보자.

다른 프로세스가 LoadLibrary()라는 API를 호출하도록 유도하여 공격자가 원하는 DLL을 로딩하게 만드는 게 원리이다.

비교적 간단한 방법으로도 DLL을 Injection 시킬 수 있기 때문에 악성코드에서 많이 사용되었다. 다만 CreateRemoteThread함수를 사용하기에, 대상 프로세스의 Injection 된 DLL을 실행할 때 들어가는 lpStartAddress라는 함수의 인자 때문에 쉽게 탐지된다는 단점을 가진다. lpStartAddress 인자란 스레드가 시작되는 순간의 함수 API 주소를 말한다. 이때 지정하는 함수가 바로 LoadLibrary(혹은 GetProcAddress)가 된다.

만약 보안 시스템이 부분을 탐지하게 되면, lpStartAddress인자가 LoadLibrary를 가리키는 것을 수상하게 여기며 DLL Injection을 막을 수 있게 된다.

왜냐하면 앞에서 설명했듯, 대상 프로세스에서 LoadLibrary()를 사용해 악성 DLL을 가져오기 때문이다.

이 DLL INJECTION 기법은 로딩 대상이 다른 프로세스가 아닌 자기 자신인 일반적 DLL 로딩과 구분된다.

이렇게 알려진 DLL INJECTION을 활용하여 메시지 후킹, API 후킹, 악성 코드에 활용 가능하다.

결론

이렇게 이번 SSR에서는 EVTX 포렌식 과제를 공부하면서 배운 DLL에 대해 깊이 공부하였다.

내가 정확하게 알지 못했던 DLL의 정의를 확실하게 배우고, 이걸 어떻게 보안 취약점을 공격하는 방법으로 활용되고 있는지 배웠다.

다음에는 내가 특히나 관심이 있던 DLL INJECTION 방식을 직접 실습해보며 INJECTION에 대한 이해를 넓히고 싶다.

참고

1. 곧대희. (2018, October 9). *DLL이란? (Dynamic Link Library)*. 곧대희의 블로그. https://goddaehee.tistory.com/185

2. jabdabg. (2024, May 20). *DLL이란? 동적 라이브러리의 개념과 실전 적용법*. 네이버 블로그. https://m.blog.naver.com/PostView.naver?blogId=jabdabg&logNo=223558291315

3. 이글루코퍼레이션. (2023, March 2). *DLL Hijacking PART 1: Search Order Hijacking - Security & Intelligence*. 이글루코퍼레이션. https://www.igloo.co.kr/security-information/dll-hijacking-part-1-search-order-hijacking/

4. 샤브샤브. (2023, December 3). *DLL Injection & 코드 인젝션*. Code ShabuShabu. https://code-shabushabu.tistory.com/16

5. ChatGPT. (2024, May 30). *GPT SSR - 모듈식 아키텍처 개념*. https://chatgpt.com/c/68259b6a-a1c0-8011-b030-55a32575c033