유저 모드 기반의 은닉된 네이티브 API 호출 탐지 기법 연구

최심현
고려대학교 컴퓨터정보통신대학원 소프트웨어보안학과
e-mail : cshvx2000@korea.ac.kr

A Study on the Method for detecting Stealth Native API calls in User-mode

Choe Sim Hyeon
Dept. of Software Security, Korea University

요 약

본 연구에서는 API 호출을 은닉할 수 있는 새로운 유형의 유저모드 기반 루트킷으로 Cuckoo Sandbox 를 회피하는 기법과 이를 탐지하기 위한 연구를 한다. Cuckoo Sandbox 의 행위 분석을 회피하기 위해 잠재적으로 출현 가능한 은닉된 코드 이미지 기반의 신종 루트킷 원리를 연구하고 탐지하기 위한 방안을 함께 연구한다. 네이티브 API 호출 코드 영역을 프로세스 공간에 직접 적재하여 네이티브 API 를 호출하는 기법은 Cuckoo Sandbox 에서 여전히 잠재적으로 행위 분석 회피가 가능하다. 본 연구에서는 은닉된 외부주소 호출 코드 영역의 탐지를 위해 프로세스의 가상메모리 공간에서 실행 가능한 페이지 영역을 탐색 후 코사인 유사도 분석으로 이미지 탐지 실험을 하였으며, 코드 영역이 맵핑된 정렬 단위의 4 가지 실험 조건에서 평균 83.5% 유사도 탐지 결과를 확인하였다.

Keywords : User mode, Native API, Rootkit, Dynamic Analysis, Detection, Memory Mapping

1. 서론

최근 안티 바이러스 솔루션의 트렌드 중 하나는 샌드 박스를 기반으로 에뮬레이션을 통한 동적 행위 분석의 결과로, 악성 코드의 주요 행위를 수집하여 유저에게 분석 정보를 제공해주고 다시 학습을 위한 데이터로 재가공하여 기계 학습의 분류를 통해 알려지지 않은 악성코드를 탐지를 목적으로 하고 있다. 악성코드는 Cuckoo Sandbox 의 가상 환경을 우회하기 위해 가상 환경임을 확인할 수 있는 버전, 서비스 리스트 혹은 고유한 파일 및 VMX 포트를 이용하거나 휴먼 인터랙션 기반의 타임 트리거 및 휴먼 이벤트를 통해 이를 감지하여 잠복 후 향후 동작하도록 하는 우회 기법을 활용하고 있다[1]. 이에 대한 대응으로, 안티 바이러스 솔루션은 가상 환경 기반의 악성코드 우회를 저지하기 위해 가상 환경 인식을 위한 특징 정보를 리얼 환경처럼 조작하거나 리얼 환경에서 행위 분석하도록 하여 이러한 우회를 무력화 할 수 있다. 그럼에도 불구하고 여전히 유저모드에서 가상 환경과 리얼 환경의 행위 분석에 관계없이 악성코드는 악성 행위와 직접 연관된 API 호출을 은닉하기 위해 유저 프로세스 공간에 NTDLL.DLL 에서 Text 섹션 일부의 외부함수 주소 호출 코드(이하 스텁코드)만 적재하여 직접 네이티브 API 를 호출함으로써 행위 은닉 기법을 활용할 수 있다. 이는 후킹 기반의 행위 분석 모니터링을 회피하여 왜곡된 분석 정보로 잠재적으로 유저에게 혼란을 야기하기 할 수 있으며 이러한 분석 데이터를 기반으로 한 기계 학습의 오탐율이 증가하는 원인이 될 수 있다. 본 논문에서는 유저 모드에서 이러한 은닉 기법을 탐지할 수 있는 방법론에 대해 연구한다.

2. 선행연구

기본적으로 악성코드는 행위 분석을 회피하기 위해 Cuckoo Sandbox 의 User-mode API unhooking 을 시도할 수 있다. 이와 같은 방법을 무력화하기 위한 기법이 “Prevalent Threats Targeting Cuckoo Sandbox Detection and Our Mitigation”에서 소개되어 있다[2]. 주요 기법은 특정 DLL 에서 Cuckoo Sandbox 가 후킹한 API Function 영역의 메모리 변조를 제한하도록 하고 ASM 명령어 쓰기 관련 동작 수행 시 ACCESS VIOLATION 처리를 위한 예외 처리 핸들러를 등록하여 무력화하도록 할 수 있다. 하지만 후킹된 API 함수를 변조하지 않고 프로세스 가상 메모리 공간에서 임의의 주소에 DLL 이미지를 직접 맵핑하여 API 호출로 다시 회피할 수 있다. 이처럼 맵핑된 DLL 검출하기 위한 기존 연구를

살펴보면, “Design & Implement of Detection Algorithm for loaded modules – rootkit-based on Windows System (2016)”[3]에서 물리적 이미지, 논리적 이미지, PE File Format 로 나누어 세 가지 모델을 제시하여 탐지 기법을 언급하였다.

모델 A 에서는 프로세스 내부에 은닉된 이미지 중 물리적 이미지는 존재하지만 PEB 구조체의 논리적 연결이 존재하지 않는 경우에 탐지하는 기법으로, 맵핑된 동적 모듈의 이름을 PEB 구조체에서 존재 여부를 확인하지만 이미지가 맵핑된 순간의 시점에서 검출하지 못하면 악성코드는 맵핑된 섹션을 별도로 할당한 가상메모리에 복제 후 삭제하므로 탐지가 불가능하며, 모델 B 에서는 맵핑된 속성 정보 MEM_IMAGE 와 PE 헤더로 검출하지만, 별도로 할당된 가상메모리 영역에 PE 헤더를 포함하지 않는 외부함수 주소 호출 코드만 존재하는 경우 탐지가 불가능하고, 모델 C 에서는 PE 헤더가 존재하지 않는 Text 섹션에서 DllMain 을 추적하는 기법이지만 외부함수 주소 호출 코드만 존재하는 경우 이 역시 탐지가 불가능하다. 본 논문에서는 이와 같이 프로세스 가상 메모리 영역에서 임의의 주소에 맵핑된 스텁코드를 탐지하기 위한 방안을 제시한다.

3. 기존 Cuckoo Sandbox 동적 분석 우회 기법

Cuckoo Sandbox 는 기본적으로 Guest Machine 의 Windows 에서 실행중인 프로세스의 가상 메모리에 맵핑된 Kernel32.dll, Kernelbase.dll, Ntdll.dll 서브시스템 모듈을 각각 INLINE 후킹하여 행위 분석 모니터링을 한다[Fig. 1].

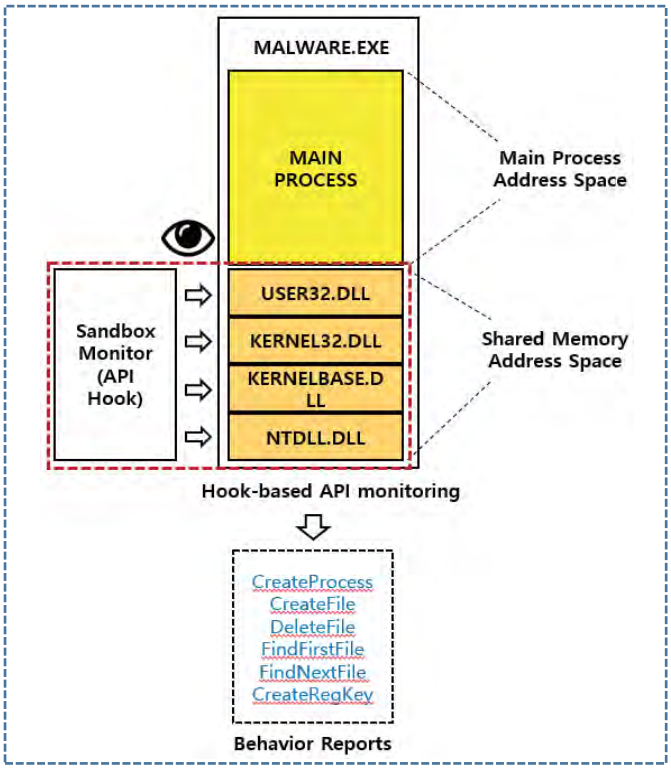

Fig. 1. Cuckoo Sandbox 모니터링 기법

이를 우회하기 위한 기본 아이디어는 서브시스템 모듈을 프로세스 임의의 가상 주소 공간에 DLL 을 적재하여 호출하여 우회 가능하다[4]. 여기서 적재할 DLL 의 대상은 Kernel32.dll 나 Kernelbase.dll 이 가능하나 모든 임포트 함수를 재귀적으로 로드하기 위해서는 더 많은 코드가 필요하다[5]. 이러한 이유로 어떤 임포트 함수를 포함하지 않고 System Call 을 랩핑한 외부함수 호출만 가진 Ntdll.dll 은 의존적이지 않고 독립적으로 호출 가능하므로 적재에 적합하다. Ntdll.dll 을 시스템 폴더 경로를 통해서 직접 적재할 수 있으나 이 방법은 재배치 작업을 수동으로 해야 하므로 복잡해진다. 따라서, 캐쉬되어 있는 KnownDll Section 에서 이미 재배치된 Ntdll.dll 을 임의의 가상 주소 공간에 맵핑하여 네이티브 API 를 호출하여 우회할 수 있다[5][6][Fig. 2]. 초기 네이티브 메모리맵 파일 API 의 획득을 은닉하기 위해 LoadLibrary 와 GetProcAddress 를 사용하지 않고 프로세스 PEB 구조체의 _PEB_LDR_DATA 에서 이미 적재되어 있는 Ntdll.dll 의 Base Address 를 획득한 후 이를 참조하여 GetProcAddress 를 직접 구현하는 기법으로 네이티브 API 두가지 NtOpenSection 과 NtMapViewOfSection 을 통해 KnownDll 섹션을 맵핑하여 Ntdll.dll 이미지 전체를 임의의 유저 주소 공간에 맵핑한다.

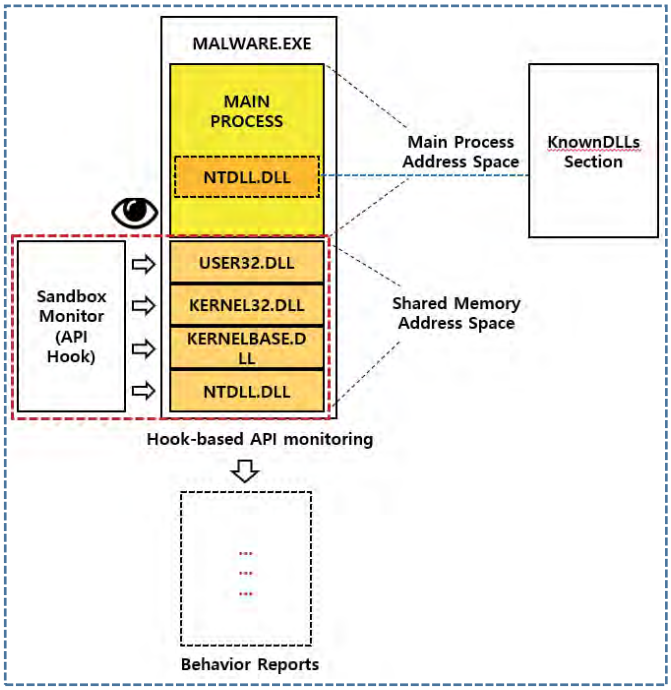

Fig. 2. Cuckoo Sandbox 분석 우회 기법

상기 탐지 기법은 프로세스가 실행중인 상태에서 주소공간을 VirtualQueryEx 로 조사하면 MEM_IMAGE 속성값을 확인한 영역에서 GetMappedFileName 을 통해 맵핑된 Ntdll.dll 의 경로 문자열을 획득하여 프로세스 PEB 구조체 내에 있는 PEB_LDR_DATA 구조체 포인터를 참조 후 _LIST_ENTRY 검색을 통해 해당 가상 메모리 기준 주소에 맵핑된 이미지를 조사하여 논리적 이미지 연결이 유효하지 않은 경우 동적으로 적

재된 이미지로 탐지 가능하다[3].

4. 새로운 동적 분석 우회 기법 연구

기존 Cuckoo Sandbox 동적 분석 우회 기법이 탐지 가능한 핵심 조건은 첫 번째로 맵핑된 Ntdll.dll 이미지의 PE 헤더 정보를 가지고 있으므로 맵핑된 기준 주소에서 'MZ' 시그니처의 확인이 가능하며, 두 번째는 맵핑된 영역의 메모리 속성과 맵핑된 DLL 이미지 이름을 조사할 수 있다. 여기서 잠재적으로 공격자가 취할 수 있는 방법은 Ntdll.dll 의 이미지 전체를 맵핑하는 것이 아니라 PE 헤더를 제외한 스텁코드만 맵핑한 후, 새로운 가상메모리 공간을 할당하여 맵핑한 코드 영역을 복제하는 기법이다[Fig. 3].

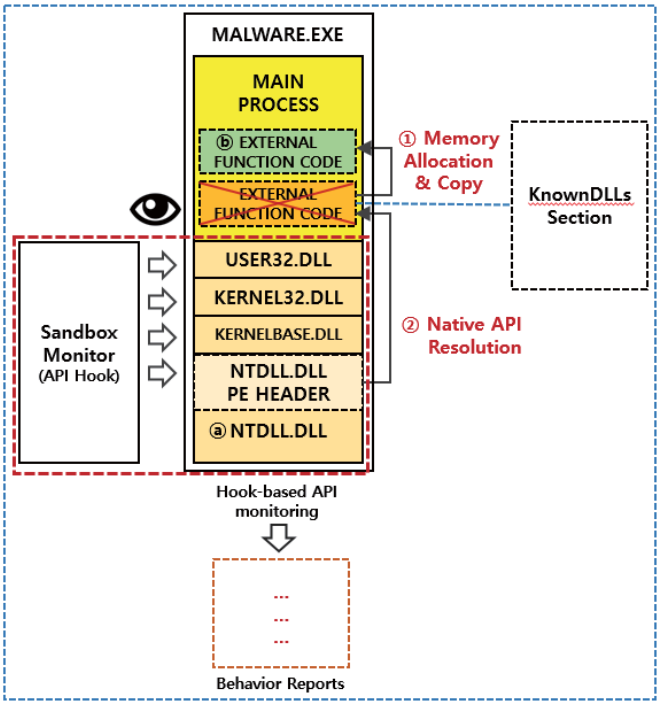

Fig. 3. 새로운 동적 분석 우회 기법

KnownDlls 섹션의 Ntdll.dll 이미지에서 맵핑할 스텁코드 영역의 시작 위치의 오프셋을 구하기 위해 의 IMAGE_OPTIONAL_HEADER 의 Image Base 를 기준으로 IMAGE_EXPORT_DIRECTORY 의 함수들의 기준 주소를 구한 후 Ordinals 의 모든 인덱스를 참조하여 스텁코드 영역의 가장 낮은 외부함수 시작 주소와 가장 높은 주소를 구할 수 있다. 이 주소값들과 정렬된 기본 맵핑 크기, 페이지 단위로 스텁코드 영역의 정렬된 시작 오프셋과 크기를 구하여 NTDLL.DLL 의 일부 코드 영역만 맵핑한다. 프로세스 공간에 새로운 가상 메모리 공간을 할당하여 맵핑한 스텁 코드를 복사한 후 기존 맵핑한 영역은 맵핑을 해제한다. 두 번째로, Native API Resolution 을 통해 System Call 을 랩핑한 스텁코드에서 각 네이티브 API 의 실제 함수 주소를 얻을 수 있다. 기존의 주소 공간에 맵핑된 NTDLL.DLL 의 PE 헤더의 Image Base 와 IMAGE_EXPORT_DIRECTORY 를 참조하여 NTDLL.DLL 의 외부함수의 시작 오프셋ⓐ을 구한 후 스텁코드 영역ⓑ의 주소를 기준으로 실제 호출할 네이티브 API 함수의 주소를 획득한다. 전체 과정을 요약하면 악성코드의 네이티브 API 호출 은닉 알고리즘은 다음 [Fig. 4]와 같다.

① 프로세스 PEB 구조체에서 NTDLL.DLL의 기준 주소 획득
② 네이티브 API Resolution – Ntdll.dll(KnownDlls)
③ NtOpenSection – Ntdll.dll
④ NtMapViewOfSection – Ntdll.dll
⑤ **NtAllocateVirtualMemory – PAGE_EXECUTE_READWRITE**
⑥ **CopyMemory – 할당된 영역에 스텁 코드 복사**
⑦ **NtUnmapViewOfSection – Mapped Address**
⑧ 네이티브 API Resolution – NtCreateFile, NtClose
⑨ **NtCreateFile 호출 (은닉 대상)**
⑩ **NtClose 호출 (은닉 대상)**

Fig. 4. 네이티브 API 호출 은닉 알고리즘

[Fig. 4] 과정에서 ①~⑧까지가 네이티브 API 호출을 은닉하기 위한 절차가 되며 ⑤~⑩은 악성코드가 호출을 은닉할 실제 네이티브 API 대상이 된다. 이와 같은 시나리오를 가진 샘플을 컴파일하여 Cuckoo Sandbox 에서 행위 분석 실험 결과는 [Fig. 5]에서 보여 주고 있다. Cuckcoo Sandbox 의 Process Behavior 탭에서 보면 해당 샘플의 분석 결과는 초기 KnownDlls 섹션에서 Ntdll.dll 맵핑을 위한 NtOpenSection 과 NtMapViewOfSection 만 표시되며, 그 이후의 NtAllocateVirtualMemory, CopyMemory, NtUnmapViewOfSection, NtCreateFile, NtClose 는 의도한대로 은닉되어 회피가 가능함을 입증하였다 [Fig.5].

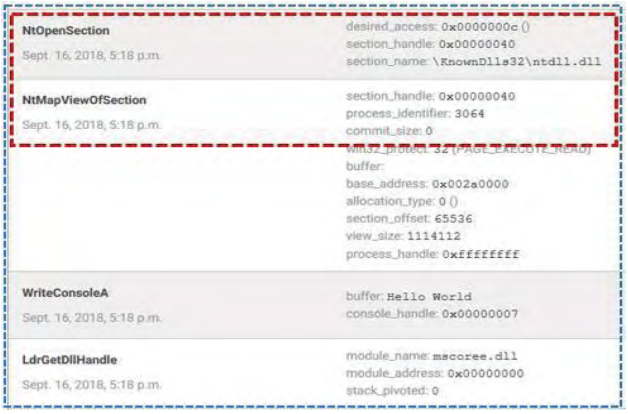

Fig. 5. 동적 분석 실험 결과

5. 새로운 동적 분석 우회 탐지 기법 연구

본 연구에서는 상기 언급한 프로세스의 가상메모리 공간에 은닉된 NTDLL.DLL 의 스텁코드를 탐지하기 위해 다음 알고리즘을 제안한다[Table 1].

Table 1. 동적 분석 우회 탐지 알고리즘

① Ntdll.dll에서 특정 스텁코드를 일부 읽어들여 버퍼에 저장

② OpenProcess – 은닉된 스텁코드를 조사할 대상 프로세스

③ 유저 프로세스 영역의 최소/최대 주소값 구함

④ NtQueryVirtualmemory – 지정된 가상메모리 주소 질의

⑤ 가상메모리 속성조사 – PAGE_EXECUTE_READWRITE & MEM_PRIVATE

⑥ ReadProcessMemory – 기준 주소(N)에서 N+0번째와 N+1번째 페이지를 읽어 들임

⑦ N+0, N+1번째 페이지의 코드와 스텁코드 버퍼를 PAGE_SIZE 단위로 Cosine Similarity 수행 (4096차원)

⑧ N+0, N+1의 유사도 평균을 계산해 Criteria가 0.80이상이면 은닉된 스텁코드로 탐지

⑨ 유저 프로세스의 모든 영역을 순회할 때까지 ④~⑧ 반복

⑩ 탐색 종료

제안한 탐지 알고리즘은 프로세스 주소 공간에서 PAGE_EXECUTE_READWRITE 와 MEM_PRIVATE 속성을 가진 페이지 영역을 탐색하여 그 기준 주소에서 N, N+1 번째 페이지를 읽은 후 바이너리 코드의 코사인 유사도 분석을 수행하게 되는데, 여기서 페이지 버퍼를 벡터 A 로 정의하고 스텁코드 패턴을 벡터 B 로 정의하여 페이지 크기 단위인 4096 차원으로 벡터의 내적을 계산하여 유사도 평균값이 0.80 이상인 경우, 은닉된 스텁코드로 탐지하도록 설계하였다. 스텁코드 패턴은 규칙성을 가진 System Call 을 호출하는 바이너리 코드 영역으로 선정하였다.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \quad (1)$$

6. 연구 결과

실험 시나리오는 공격자가 NTDLL.DLL 전체 이미지에서 가상주소 공간에 임의의 오프셋으로부터 맵핑할 수 있다는 것을 가정하고 실험하였다. 탐지 유사도 평가 조건은 4 가지 바이트 정렬 단위(1B, 4B, 16B, 256B)와 임의의 주소 영역을 기준으로 6 가지의 정렬 단위로 증가하는 오프셋(N~N+5)에서 페이지 단위로 코사인 유사도 분석 정확도를 실험하였다. 16 바이트나 256 바이트처럼 큰 자릿수 단위로 쉬프트된 정렬단위에서는 평균 96.8%의 높은 유사도를 보였으나 반면, 1 바이트나 4 바이트처럼 작은 자릿수 단위로 쉬프트된 경우, 비교적 낮은 평균 70.2%의 유사도를 확인하였다.

Table 2. 스텁코드 코사인 유사도 평가 결과

	N	N+1	N+2	N+3	N+4	N+5
1B	0.976	0.806	0.649	0.564	0.632	0.724
4B	0.976	0.632	0.457	0.634	0.809	0.564
16B	0.976	0.973	0.974	0.973	0.973	0.974
256B	0.976	0.987	0.967	0.958	0.949	0.938

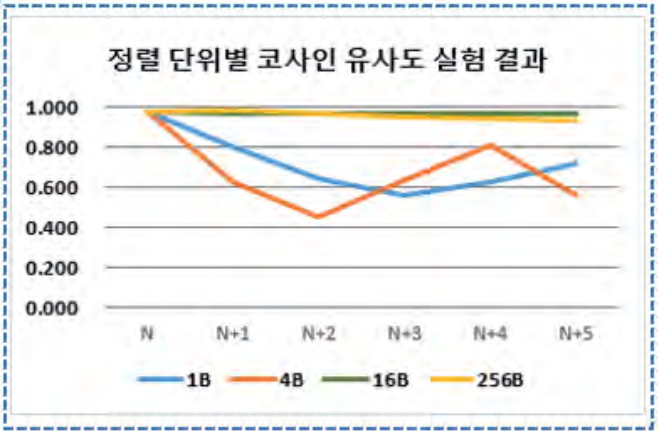

Fig. 6. 스텁코드 코사인 유사도 평가 결과

7. 결론

유저모드 관점에서 제시한 새로운 유형의 은닉된 스텁코드를 탐지하기 위해, 본 연구에서 제안한 프로세스의 가상메모리 영역 탐색 및 바이너리 이미지의 코사인 유사도 분석을 통한 탐지 기법을 현업에서 적용하여 스텁코드 이미지를 탐지하기 위한 보조 도구로 활용할 수 있을 것으로 기대한다.

참고문헌

[1] Dilshan Keragala, Detecting Malware and Sandbox Evasion Techniques, SANS Institute, Malware Evasion Techniques, p3-8, January 16, 2016

[2] Floser Bacurio and Wayne Low, Prevalent Threats Targeting Cuckoo Sandbox Detection and Our Mitigation, January 03, 2018

[3] Hwang, Won Il, Design & Implement of Detection Algorithm for loaded modules-rootkit base on Windows System, Hanyang University, p23-30, 2016.8

[4] Monirul I. Sharif, ROBUST AND EFFICIENT MALWARE ANALYSIS AND HOST-BASED MONITORING, Georgia Institute of Technology, DLLs loaded at arbitrary virtual addresses, p38, December 2010

[5] Scorch Security, Bypassing user-mode hooks the sneaky way, https://scorchsecurity.wordpress.com/2016/08/26/bypassing-user-mode-the-sneaky-way/, 2016.8

[6] Larry Osterman, What are Known DLLs anyway?, https://blogs.msdn.microsoft.com/larryosterman/2004/07/19/what-are-known-dlls-anyway/, Microsoft, July 19, 2004