Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.1, Feb. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.1.75

스레드 기반 모니터링을 통한 악의적인 행위 주체 추적 및 차단에 관한 연구*

고 보 승,[1†] 최 원 혁,[2‡] 정 다 정[3]
[1,2,3](주) 누 리 랩 (책임 연구원, 대표이사, 주임 연구원)

A Study on the Tracking and Blocking of Malicious Actors through Thread-Based Monitoring*

Boseung Ko,[1†] Wonhyok Choi,[2‡] Dajung Jeong[3]
[1,2,3]Nurilab Inc. (Senior Research Engineer, CEO, Assistant Research Engineer)

요 약

최근 윈도우즈 운영체제 환경에서 악성코드가 고도화됨에 따라 악의적인 행위를 수행하는 주체가 프로세스가 아닌 경우가 많이 발생하고 있다. 운영체제에 기본적으로 탑재된 프로세스 등에 삽입되어 동작하는 악성코드는 DLL/코드 인젝션과 같은 방식으로 스레드 단위로 동작한다. 이 경우 프로세스 단위로 악성 유무를 진단 및 차단하는 것은 시스템 운영에 심각한 문제를 야기할 수도 있다. 본 논문에서는 프로세스 기반 모니터링 정보를 사용하여 프로세스의 악성유무를 판단하고 차단하는 방법이 가지고 있는 문제점을 나열하고 그에 대한 개선된 방안을 제시한다.

ABSTRACT

With the recent advancement of malware, the actors performing malicious tasks are often not processes. Malicious code injected into the process that is installed by default in the operating system works thread by thread in the same way as DLL / code injection. In this case, diagnosing and blocking the process as malicious can cause serious problems with system operation. This white paper lists the problems of how to use process-based monitoring information to identify and block the malicious state of a process and presents an improved solution.

Keywords: Process behavior, Remotethread, DLL/Code injection

I. 서 론

악성코드를 진단하는 가장 간단한 방법은 실행파일에 대한 시그너처를 생성하여 이미 알려진 악성코드 DB와 패턴 매칭을 수행하거나 해시값으로 비교하는 것이다. 하지만 이러한 방법은 악성코드가 빠른 속도로 늘어나고 있는 상황에서는 적절하지 않다. 더불어 악성행위의 주체가 프로세스가 아니라 스레드인 경우에는 해시값을 추출하거나 시그너처를 생성할 대상이 불분명해진다. 그렇기 때문에 보유하고 있는 악성코드 DB를 활용하여 진단하는 방법만으로는 그 한계가 있다.

기존 연구[1]에서는 이러한 악성코드 진단방법의 단점을 보완하고자 프로세스 행위 기반의 진단방법을 제안하고 있다. 빠르고 정확한 진단을 위해서 악성코드 DB로 1차 확인과정을 거친 다음 악성으로 판별되지 않은 프로세스에 대해서는 행위를 지속적으로 모니터링하면서 비정상 행위를 수행하는지 지켜보는

Received(01. 09. 2020), Modified(02. 05. 2020), Accepted(02. 06. 2020)
* 이 논문은 국방과학연구소 핵심기술 과제(사이버전 모의전투 기술)의 지원으로 수행된 연구임(UD190003ED).
† 주저자, bsko@nurilab.com
‡ 교신저자, whchoi@nurilab.com(Corresponding author)

것이다. 하지만 악성코드의 동작이 고도화됨에 따라 악의적인 연산을 수행하는 주체가 프로세스가 아닌 스레드로 동작하는 경우가 늘고 있다. 이런 경우에 스레드는 운영제체에서 중요한 역할을 수행하는 기본 탑재된 시스템 운영 프로세스(svchost.exe, csrss.exe, explorer.exe, 등)를 대상으로 리모트스레드로 생성되는 경우가 많으며 경우에 따라서는 유명한 채팅 프로그램 등의 일반 써드파티 응용 프로그램을 대상으로 하는 경우도 있다.

프로세스 기반 모니터링을 통해서 악성행위를 진단 및 차단하는 방법은 프로세스 자체가 악성코드인 경우에는 유용하다. 하지만 앞서 언급한 것처럼 시스템 운영 프로세스에 리모트스레드로 생성되어 스레드로 동작하는 경우에 이를 프로세스 기준으로 진단하고 차단하게 되면 해당 프로세스의 정상적인 연산도 차단되기 때문에 시스템 운영에 치명적일 수 있다.

본 논문에서는 스레드 기반 모니터링을 통해서 악의적인 연산을 수행하는 스레드를 추적하고 진단 및 차단하는 방법을 제시한다.

II. 프로세스 및 스레드 행위 수집

프로세스와 스레드의 행위를 모니터링 할 수 있는 방법은 다양하다. 커널 레벨에서 필터 드라이버를 사용할 수도 있고 사용자 레벨에서 API 후킹 기법 등을 사용할 수도 있다. 본 논문에서는 커널 레벨에서 파일시스템 미니필터 드라이버 기반으로 프로세스와 스레드의 행위를 모니터링 하는 방법을 사용한다. 각각 장단점이 있지만 커널 레벨에서 모니터링 하는 방법은 모든 프로세스 및 스레드에 전역으로 적용되기 때문에 사용자 레벨에서 특정 프로세스를 지정하여 모니터링 하는 것 보다 문제를 해결하는데 수월하다.

2.1 수집 대상

미니필터 드라이버를 사용하여 파일연산, 레지스트리 연산, 프로세스/스레드 생성 및 종료 연산에 대해서 모니터링을 수행한다.

2.1.1 파일 연산 모니터링

파일시스템 미니필터 드라이버는 필터 매니저와 연계하여 기본적으로 파일 연산을 모니터링 할 수 있는 기능을 제공한다[2]. 본 논문에서는 악의적인 행위 주체 추적 및 차단을 위해 필요한 모니터링 대상을 위주로 수집한다. 파일 연산에서는 Create, Open, Read, Write, Rename, Move, Link, Delete가 그 대상이다. Table 1.은 모니터링 대상인 파일 연산을 표로 정리해서 보여준다.

Table 1. File Actions

Action	Src	Dst
Create	O	
Open	O	
Read	O	
Write	O	
Rename	O	O
Move	O	O
Link	O	O
Delete	O	

Rename, Move, Link 연산의 경우에는 원래 경로(Src)와 변경 후의 경로(Dst)로 구분하여 관리한다. 그 이유는 파일의 경로상의 변경과정을 추적하기 위해서 이다. 일반적인 프로그램의 경우를 포함해서 악성코드의 경우에도 파일을 다룰 때 임시 파일을 생성하고 이름변경 또는 이동연산을 수행하여 최종적으로 파일 경로를 결정하는 연산이 빈번하게 발생한다.

Table 2. Data Structure for File Information

Member	Type	Description
Identifier	int64	Unique Identifier
PID	int	Process Id
TID	int	Thread Id
PPID	int	Parent's PID
PPath	String	Process ImagePath
Time	int64	Event Time
Action	int	File Action
SrcPath	String	Source File Path
DstPath	String	Destination File Path

Table 2.는 미니필터 드라이버에서 수집되는 각각의 실시간 파일 연산 정보를 관리하는 구조이다. 기본적으로 파일 연산 주체인 프로세스/스레드 정보와 파일 연산 대상인 파일 경로 정보로 이루어져 있다. 이러한 낱개의 실시간 정보는 유일한 식별자인

Identifier로 취합되어 추후 프로세스/스레드 연산 주체가 어떠한 파일 연산을 수행했는지 확인하기 편하게 관리된다.

2.1.2 레지스트리 연산 모니터링

레지스트리 연산을 모니터링 하기 위해서 해당 기능을 수행하는 별도의 드라이버를 구현할 수도 있지만 그것은 관리 측면에서 불필요한 비용이 발생하기 때문에 일반적인 경우라면 커널 레벨에서의 실시간 I/O 모니터링 연산은 하나의 드라이버에서 처리하는 것이 효율적이다. 잘 알려진 모니터링 도구인 Process Monitor[3]도 미니필터로 동작하는 하나의 커널 드라이버에서 다양한 모니터링을 수행한다.

레지스트리 연산을 모니터링 하기 위해서 필요한 함수는 CmRegisterCallback(Ex) 함수[4]이다. 이 함수를 미니필터 드라이버에서 호출하여 다양한 레지스트리 연산 정보를 콜백함수로 전달받을 수 있다.

파일 연산과 마찬가지로 추적에 필요한 (Interested) 연산만 추려내어 레지스트리 연산을 모니터링 한다. 레지스트리 연산에서는 CreateKey, SetValue 에 대한 정보만으로도 최소한의 추적이 가능하다. Table 3.은 다양한 레지스트리 연산 중에서 모니터링 대상만 표시한 것이다.

Table 3. Registry Actions

Action	Interested
CreateKey	O
OpenKey	
EnumKey	
EnumValue	
SetValue	O
QueryKey	
QueryValue	
DeleteKey	
DeleteValue	
…	

프로세스/스레드 추적을 위해서 레지스트리 연산을 모니터링 하는 주요 목적은 시스템 재시작시 자동 실행되는 프로그램 목록에 대한 설정 변경을 모니터링 하기 위해서 이다. 이러한 설정 변경에 대한 확인은 CreateKey 와 SetValue 연산을 모니터링 하는 것으로 가능하다. 물론 Table 3.에서 언급한 다양한 레지스트리 연산에 대해서 모두 모니터링 정보를 수집해도 되지만 EnumXxx, QueryXxx 연산은 아주 빈번하게 발생하기 때문에 모니터링 성능을 저하시킬 수도 있다. 그리고 OpenKey, DeleteXxx 연산은 랜섬웨어가 시스템 재시작시 자동실행 되도록 설정하는 연산과는 무관하다.

Table 4. Data Structure for Registry Information

Member	Type	Description
Identifier	int64	Unique Identifier
PID	int	Process Id
TID	int	Thread Id
PPID	int	Parent's PID
PPath	String	Process ImagePath
Time	int64	Event Time
Action	int	Registry Action
RegPath	String	Registry Path

Table 4.는 미니필터 드라이버에서 수집되는 각각의 실시간 레지스트리 연산 정보를 관리하는 구조이다. 실시간 파일 연산 정보와 거의 동일한 구조를 가지고 있다는 것을 알 수 있다.

2.1.3 프로세스/스레드 생성 및 종료 모니터링

마지막으로 프로세스와 스레드의 생성 및 종료 연산을 모니터링 한다. 임의의 프로세스가 어떤 자식 프로세스를 생성하는지 또는 어떤 스레드를 어느 프로세스에 생성하는지 모니터링 한다. 이때 필요한 함수는 PsSetCreateProcessNotifyRoutine(Ex) 함수[5]와, PsSetCreateThreadNotifyRoutine 함수[6]이다. 이 함수들을 호출할 때 등록되는 콜백 함수를 통해서 프로세스/스레드의 생성 및 종료 통지를 모니터링 할 수 있다.

Table 5. Process/Thread Actions

Category	Action
Process	Create
	Terminate
Thread	Create
	Terminate

파일, 레지스트리 연산에 대한 모니터링이 스토리지에 저장되는 파일 자체를 추적하는데 필요한 정보라면 프로세스/스레드의 생성 및 종료에 대한 모니터링 정보는 어떤 행위를 수행한 주체를 추적하기 위해서 필요한 정보라고 할 수 있다.

Table 6. Data Structure for Process/Thread Information

Member	Type	Description
Identifier	int64	Unique Identifier
PID	int	Process Id
TID	int	Thread Id
PPath	String	Process ImagePath
Time	int64	Event Time
Action	int	Process/Thread Action
TPID	int	Target Process Id
TTID	int	Target Thread Id
TPPath	String	Target Process ImagePath

Table 6.은 미니필터 드라이버에서 수집되는 프로세스/스레드의 생성 및 종료 정보를 관리하는 구조이다. 실시간 파일, 레지스트리 연산 정보와 거의 동일한 구조를 가지고 있다는 것을 알 수 있다.

2.2 기준 식별자

미니필터 드라이버를 사용하여 커널 레벨에서 임의의 모든 프로세스/스레드가 수행하는 연산을 모니터링 할 수 있다. 실시간으로 발생하는 모니터링 정보를 누가 수행하고 있는지 구별할 수 있도록 유일한 식별자를 선정하여 그 기준으로 모니터링 정보를 수집하면 해당 식별자 별로 어떠한 연산을 수행했는지 확인할 수 있다. 본 논문에서는 이를 기준 식별자라는 용어로 사용한다.

단순하게 생각하면 Process Id나 Thread Id를 기준 식별자로 사용할 수 있을 것이다. 하지만 이 값들은 유일한 값이 아니다. 예를 들어, 실행중인 메모장의 Process Id가 100이라고 하자. 이 메모장이 종료된 후에는 Process Id 100은 다른 프로세스의 Process Id로 사용될 수 있다. 스레드도 마찬가지다. 행위 주체를 추적하는 입장에서는 종료된 프로세스나 스레드도 관심 대상이기 때문에 유일성을 보장하는 측면에서 Process Id나 Thread Id는 그렇지 못하다.

본 논문에서는 기준 식별자로 시간을 사용한다. 프로세스 기반 행위를 수집하는 경우에 기준 식별자는 행위를 수행하는 프로세스의 생성 시간이다. 스레드의 경우에는 스레드의 생성 시간이다.

Table 2., Table 4., Table 6.에서 보여주는 데이터 구조체에서 Identifier 멤버에 해당하는 것이 바로 기준 식별자이다. Identifier 값은 해당 구조체의 PID(또는 TID)에 해당하는 프로세스(또는 스레드)의 생성시간을 8 바이트 값으로 나타낸 것이다.

파일 연산, 레지스트리 연산, 프로세스/스레드의 생성 및 종료 연산에 대한 모니터링 정보를 프로세스(또는 스레드)의 생성시간을 기준으로 취합하면 Process Id나 Thread Id가 우연히 겹치는 경우라도 유일한 식별자로서 연산을 구분 지을 수 있다.

III. 스레드 추적 및 차단

본 논문에서 다루는 주요 내용은 악성행위의 주체가 스레드인 경우에 대한 것이다. 대개 이런 경우는 시스템 운영 프로세스나 널리 사용되는 써드파티 프로그램을 대상으로 리모트스레드를 생성하는 DLL/Code 인젝션 공격인 경우이다. 지금부터 다루는 내용은 악성코드 프로세스가 명시적으로 실행된 경우 혹은 운영체제의 취약점이나 인터넷 브라우저 등의 취약점에 의해서 악성코드가 실행되는 경우를 통틀어서 이들 악성코드가 임의의 타겟 프로세스에 리모트 스레드를 생성하는 경우에 이를 추적하기 위한 방법을 설명한다.

3.1 스레드 기반 악성코드

서론에서도 잠시 언급했듯이 파일 시그너처를 사용하여 악성코드를 진단하는 방법은 그 한계가 있다. 악성코드의 증가율에 비해서 시그너처 DB의 양을 늘리는 것이 항상 뒤쳐질 수밖에 없기 때문이다. 더불어 악성코드의 동작이 파일리스(fileless) 방식[7]을 사용한다면 이러한 문제는 더욱 심각해진다. 말 그대로 악성코드에 해당하는 파일이 디스크 상에는 존재하지 않기 때문에 파일을 스캔하여 시그너처 DB와 비교할 수가 없는 것이다. 파일리스 악성코드는 여러 가지 방식으로 동작할 수 있다. 본 논문에서

는 한 가지 예로써 파일리스 악성코드가 리모트스레드를 생성하여 악의적인 행위를 하는 방법을 설명하고 스레드 기반 모니터링을 사용하여 대응하는 방법에 대해서 설명한다.

3.1.1 동작방식

파일리스 악성코드에 감염되는 과정을 예를 들어 간단하게 설명하면 다음과 같다. 우선 사용자가 웹메일을 확인하던 중에 스팸 메일에 의해 특정 사이트에 접속했다고 생각해보자. 이 사이트를 방문하면 플래시가 로드된다. 플래시에 대한 보안취약점이 존재하는 상황이라면 플래시는 파워쉘을 사용하여 네트워크 접속을 시도한다. 파워쉘은 C&C (Command and Control) 서버[8]에 접속하여 암호화된 악성 DLL을 메모리 버퍼로 읽어 들이고 타겟 프로세스(예, svchost.exe 혹은 explorer.exe 등)를 지정하여 Reflective DLL 인젝션[9]을 수행한다. 이때 리모트스레드가 타겟 프로세스에 생성되고 악성 행위가 시작된다.

3.1.2 기존 공격과의 차이

앞서 동작방식에서 설명한 내용은 악의적인 행위를 수행하는 리모트스레드의 생성과정을 스팸메일에 의한 악성사이트 접속의 예를 들어 설명하였다. 이처럼 고도화되기 전에는 보다 단순한 방법으로 감염상황을 유도하였다. 예를 들면 스팸메일의 첨부파일 자체가 실행 가능한 악성코드이거나 웹 브라우저를 사용하여 웹서핑 도중에 취약점에 의해서 악성코드가 다운로드되거나 하는 것이다. 이러한 악성코드는 명시적으로 자신만의 프로세스로 실행된다. 자신의 악의적인 동작을 숨기기 위해서 타겟 프로세스에 리모트스레드를 생성하지 않았다.

이처럼 최근 악성코드의 동작은 자신의 존재를 최대한 숨기려 한다. 파일리스 방식으로 물리적인 파일이 존재하지 않는 경우가 증가하고 있고 기존 시스템에 존재하는 프로세스에 리모트스레드를 생성하여 악의적인 행위를 수행하도록 하고 있다.

3.1.3 진단 및 차단

파일리스 동작방식을 사용하는 악성코드는 기존 시그너처 DB를 가지고 진단하지 못한다. 굳이 파일리스 악성코드가 아니더라도 하루가 멀다하고 발견되는 신종, 변종 악성코드를 모두 시그너처 DB에 반영하는 것은 거의 불가능하다. 그리고 악성코드가 자체 프로세스로 동작하는 경우라면 해당 프로세스를 종료시키는 방식으로 차단 및 치료하면 된다. 하지만 기존에 동작중인 정상의 프로세스에 리모트스레드를 생성하여 악의적인 행위를 수행하는 경우라면 프로세스를 종료시키는 방식으로는 잘 차단했다고 볼 수 없다. 만약 csrss.exe에 리모트스레드를 생성하여 악의적인 연산을 수행중이라고 가정해보자. 악의적이라고 판단되었을 때 csrss.exe 프로세스를 강제로 종료시킨다면 시스템은 블루스크린를 발생시킬 것이다. 블루스크린의 이유는 Bug Check 0xF4: CRITICAL_OBJECT_TERMINATION[10] 이라는 이름으로 설명하고 있다.

리모트스레드에 의한 악의적인 공격을 차단하기 위해서 프로세스 단위로 치료를 수행하게 되면 이처럼 심각한 문제를 야기하기 때문에 이러한 경우라면 스레드 단위로 치료를 수행해야 한다. 그리고 시그너처 DB를 사용해서 스레드를 진단할 수는 없기 때문에 스레드 기반 모니터링 결과를 가지고 행위 진단 방식을 사용해야 한다.

일반적으로 행위 기반 악성 코드 분석은 프로세스의 행위를 대상으로 한다. Detection and Classification of Malicious Processes Using System Call Analysis 논문[11]에서 소개하는 행위 분석도 프로세스를 대상으로 하고 있다. 본 논문에서 소개하는 스레드 기반 모니터링 또한 전반적인 모니터링 정책은 프로세스를 대상으로 했을 때와 동일하게 가져갈 수 있다. 다만 모니터링 데이터를 수집할 때 기준 식별자를 무엇으로 두느냐에 따라서 프로세스 단위 행위 수집이 되거나 스레드 단위 행위 수집이 되거나 하는 것이다. 여기서는 기준 식별자별로 아주 간단한 행위 진단과 스레드를 추적하기 위한 정보만 수집하도록 한다.

그리고 본 논문에서는 모니터링 해야 하는 자원이 어떤 것이어야 하는지에 대한 것 보다는 악의적인 행위를 수행하는 주체 스레드를 추적하는 쪽에 초점을 맞추고 있다. 즉, 모니터링 데이터에서 특징을 추출하고 분류하여 악성 유무를 판단하는 것은 진단에 대한 것이다. 그리고 악의적의 행위를 수행한 주체가 프로세스인지 스레드인지, 스레드라면 해당 스레드의 생성 주체가 누구인지 밝혀내는 것은 추적 및 차단에 대한 것이다.

3.2 스레드 생성 테이블

PsSetCreateThreadNotifyRoutine 함수를 호출할 때 입력 매개변수로 사용된 콜백함수는 임의의 스레드가 생성될 때 호출된다. 콜백함수의 매개변수는 Fig. 1과 같다.

```
PCREATE_THREAD_NOTIFY_ROUTINE PcreateThreadNotifyRoutine;

void PcreateThreadNotifyRoutine(
  HANDLE ProcessId,
  HANDLE ThreadId,
  BOOLEAN Create
)
{...}
```

Fig. 1. PCREATE_THREAD_NOTIFY_ROUTINE callback function

ProcessId는 스레드가 생성되는 프로세스의 PID이다. ThreadId는 이제 막 생성된 스레드의 TID이다. Create 은 TRUE/FALSE 값을 가지며 각각 생성/종료를 의미한다. 콜백함수의 입력 매개변수인 ProcessId, ThreadId, Create 정보와 함께 스레드의 생성 및 종료 주체에 대한 정보를 확인할 수 있다. PsGetCurrentProcessId 함수[12]는 현재 연산을 수행하는 프로세스의 PID를 반환한다. PsGetCurrentThreadId 함수[13]는 현재 연산을 수행하는 스레드의 TID를 반환한다.

스레드의 생성에 대한 콜백함수가 호출될 때 PsGetCurrentProcessId 함수를 호출하여 현재 PID를 구하고 PsGetCurrentThreadId 함수를 호출하여 현재 TID를 구한다. 그리고 콜백함수의 입력 매개변수의 정보를 조합하면 Table 7. 과 같은 테이블을 구성할 수 있다.

Table 7. Thread Creation Table

#	Src PID	Src TID	Dst PID	Dst TID
1				
2				
…				

Src는 스레드를 생성한 행위 주체자를 의미한다. 여기서는 PsGetCurrentXxxId 함수를 사용하여 구한 PID, TID가 이에 해당한다. Dst는 스레드가 생성되는 곳의 당사자를 의미한다. 이 정보는 스레드 생성 통지 시점에 호출되는 콜백함수의 매개변수인 ProcessId, ThreadId로부터 구할 수 있다.

3.3 리모트스레드 판별

리모트스레드가 생성되는지 확인하는 가장 손쉬운 방법은 CreateRemoteThread 함수[14]를 후킹하여 모니터링하는 것이다. 하지만 이 방법은 사용자 레벨에서 API 후킹 등의 방법으로 가능하다. 커널 레벨에서 CreateRemoteThread 함수에 대응하는 커널 함수는 존재하지 않는다.

본 논문에서는 커널 레벨에서 미니필터 드라이버를 사용하여 파일 연산, 레지스트리 연산, 프로세스/스레드의 생성 및 종료 연산을 모니터링 한다. 여기서 스레드의 생성 통지 연산 모니터링을 사용하여 리모트스레드의 생성 여부를 확인할 수 있다.

스레드의 생성 통지 시점에 스레드 생성 테이블을 구성할 수 있다. Src PID는 스레드를 생성하는 주체 프로세스의 PID 이다. Dst PID는 스레드가 생성되는 대상 프로세스의 PID이다. 스레드 생성 통지 시점에 Src PID와 Dst PID가 다르면 이것은 리모트스레드의 생성이다. 그리고 이때 Dst TID는 리모트스레드의 ThreadId이다.

주의해야 할 점은 프로세스의 생성은 항상 리모트스레드의 생성을 동반한다는 것이다. 예를 들어서 탐색기에서 메모장을 실행하는 경우에 최초 발생하는 스레드 생성 통지에서 Src PID는 탐색기에 대한 것이고 Dst PID는 메모장에 대한 것이다. 본 논문에서 말하는 리모트스레드는 이처럼 일반적인 프로세스 생성과정 중에 프로세스의 최초 스레드를 생성하는 것은 배제한다. 여기서 말하는 리모트스레드는 기존에 동작중인 프로세스에 DLL/Code 인젝션 등의 방법으로 리모트스레드를 생성하는 경우에 한정한다.

결국 본 논문에서 말하는 리모트스레드의 생성을 확인하는 방법은 스레드 생성 통지 시점에 Src PID와 Dst PID가 다르고 Dst PID에 해당하는 프로세스의 스레드 개수가 2개 이상인 경우이다.

Fig. 2. 는 리모트스레드 자신이 직접 악의적인 행위를 수행하는 경우이다.

Fig. 3. 은 리모트스레드에 의해서 여러 단계로 생성된 스레드가 악의적인 행위를 수행하는 경우이다. 이처럼 다양한 형태로 리모트스레드는 동작할 수 있다.

기존 스레드 생성 테이블을 보강하여 리모트스레드

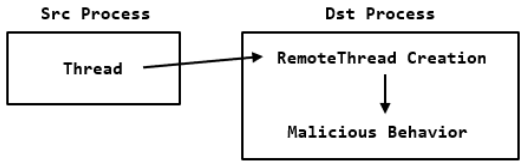


Fig. 2. The case of malicious behavior of RemoteThread

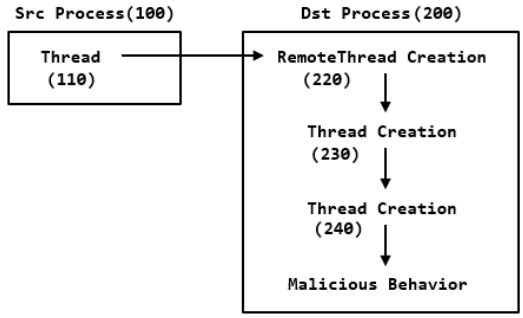


Fig. 3. Malicious behavior case caused by the RemoteThread's derived thread

여부를 나타내는 항목을 추가하면 보다 자세한 스레드 생성 흐름을 확인할 수 있다. Table 8. 은 Fig. 3.에 대한 스레드 생성 흐름을 나타내는 강화된 스레드 생성 테이블을 보여준다.

Table 8. Advanced Thread Creation Table

#	Src PID	Src TID	Remote Thread	Dst PID	Dst TID
1	100	110	O	200	220
2	200	220	X	200	230
3	200	230	X	200	240

3.4 프로세스 vs. 스레드 차단

단순히 프로세스 식별자를 기준으로 하여 행위를 수집하고 악성 유무를 판별하게 되면 최종적으로 수행해야 하는 절차인 차단에 대해서도 프로세스를 대상으로 하게 된다. 이런 경우에 문제가 되는 부분은 차단할 프로세스가 시스템 운영 프로세스인 경우이다. 악성코드가 svchost.exe 프로세스를 공격대상으로 하여 악의적인 리모트스레드를 생성하였고 이후 파생 스레드가 생성되어 악의적인 행위를 했다고 가정해보자. 스레드 생성 테이블도 존재하지 않고 리모트스레드의 생성 여부에 대해서도 아무런 정보가 없는 경우이다. 프로세스에 대한 행위를 취합하여 행위기반 탐지 등의 기법을 사용하여 악의적인 행위라고 진단한 경우에 svchost.exe는 악성행위 프로세스로 간주되어 이후의 연산이 차단되거나 프로세스 자체가 강제 종료될 것이다. 여기서 발생하는 문제점을 나열해보면 우선 첫 번째로 프로세스 자체가 종료되었다는 것이다. 때문에 해당 프로세스의 정상적인 연산을 수행하는 스레드도 자동적으로 종료되어 버린다. 그리고 두 번째는 svchost.exe 와 연계작업을 수행중인 다른 프로세스의 연산이 비정상 동작을 하게 된다. svchost.exe는 말 그대로 서비스를 호스팅 해주는 역할을 한다. 플러그 앤 플레이, RPC 관리, 세션 관리, 네트워크 연결 관리, 윈도우즈 업데이트 관리 등등 상당히 많은 서비스를 다루기 때문에 강제 종료된 svchost.exe가 담당하는 서비스와 관련된 작업을 수행하는 임의의 다른 프로세스는 오작동의 여지가 있다. 마지막으로 앞서 csrss.exe 프로세스의 강제 종료에서 언급했던 블루스크린 문제가 발생할 수도 있다.

본 논문에서 다루고자 하는 주요 내용은 바로 이러한 상황에서 보다 효과적으로 차단 혹은 치료를 하는 방법에 대한 것이다. svchost.exe 프로세스에 리모트스레드를 생성하여 악의적인 행위를 수행하는 동일한 상황에서 스레드 생성 테이블이 존재하고 리모트스레드의 생성이라고 판명된 스레드에 대해서는 스레드의 행위를 취합하도록 하자. 그리고 파생 스레드의 경우를 감안하여 리모트스레드에 의해서 생성된 파생 스레드에 대해서도 행위를 취합하도록 하자. 즉, 프로세스 기반 행위도 취합하고 리모트스레드나 그 파생 스레드라고 확인된 경우에 대해서는 스레드 기반 행위도 취합하는 것이다. 행위기반 탐지방법으로 악의적이라고 진단된다면 svchost.exe에 리모트스레드 및 그 파생 스레드만을 차단 및 종료하여 기존 svchost.exe의 연산에는 영향을 주지 않고 치료할 수 있다.

앞서 프로세스의 생성시간, 스레드의 생성시간을 기준 식별자로 사용하여 파일, 레지스트리, 프로세스/스레드의 생성 등의 연산을 수집하였다. 기본적으로 프로세스는 프로세스의 생성시간을 기준 식별자로 하여 행위를 수집한다. 그리고 스레드 생성 테이블의 정보를 바탕으로 리모트스레드이거나 그 파생 스레드의 경우에는 스레드의 생성시간을 기준 식별자로 하여 행위를 수집한다. 행위기반 탐지방법으로 진단되

면 해당 스레드(또는 프로세스)를 종료한 후 해당 스레드(또는 프로세스)가 수행한 행위에 대해서만 치료하면 된다. 그 행위 안에는 파일 연산으로 확인할 수 있는 숙주 파일의 생성 및 이름변경 정보를 포함할 수 있고 레지스트리 연산으로 확인할 수 있는 시스템 재시작시 숙주 파일의 실행을 담당하는 레지스트리 설정 값 등의 정보도 확인할 수 있다.

스레드 생성 테이블을 사용하여 리모트스레드와 그 파생 스레드를 구별할 수 있고 이러한 경우에 해당 스레드에 대해서는 스레드 생성시간을 기준 식별자로 하여 파일, 레지스트리 등의 연산을 선별하여 모니터링 할 수 있다. 행위기반 탐지기법으로 스레드의 행위가 악의적이라고 판단되는 경우에는 해당 스레드가 수행한 파일, 레지스트리 연산을 확인하여 숙주 파일과 악의적인 레지스트리 설정을 제거할 수도 있다.

DLL/Code 인젝션 등의 방법으로 리모트스레드를 이용한 공격인 경우라면 프로세스 기반 모니터링이 아닌 스레드 기반 모니터링 방법을 사용하는 것이 보다 안전하게 치료하는데 효과적이다.

3.5 스레드 기반 악성코드 탐지 알고리즘

프로세스를 타겟팅하여 리모트스레드를 생성한 다음 악의적인 행위를 수행하는 경우에 스레드 단위로 모니터링 데이터를 수집하는 내용을 그림으로 표현하면 Fig. 4.와 같다.

이러한 경우에 모니터링 데이터는 Malicious Thread의 스레드 생성 시간을 기준 식별자로 사용하여 수집하게 된다. Fig. 5.는 기준 식별자를 사용하여 모니터링 데이터를 취합한 자료구조이다. 해시 테이블 알고리즘[15]을 사용해서 ProcessMap, ThreadMap을 관리한다. 이때 기준 식별자인 생성시간을 Key로 사용하여 Bucket을 정하고 해당 Bucket에서부터 각 프로세스/스레드 엔트리(P/T)를 찾아갈 수 있는 연결리스트를 따라가면 Key와 일치하는 생성시간을 가지는 프로세스/스레드의 모니터링 정보를 확인할 수 있다.

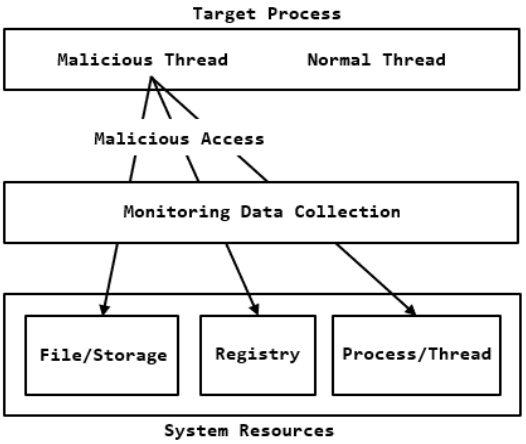


Fig. 4. Thread based monitoring

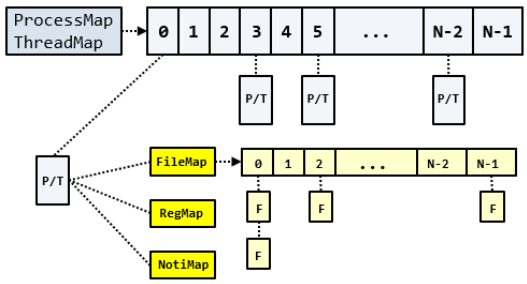


Fig. 5. Monitoring data structure

각 프로세스/스레드 엔트리는 또다시 파일, 레지스트리, 프로세스/스레드의 생성 통지에 대한 모니터링 데이터를 관리하는 해시 테이블로 구성되어 있다. FileMap으로 관리하는 파일 연산 정보는 Table 2.에서 설명한 자료구조의 엔트리들로 채워지고 RegMap, NotiMap은 각각 Table 4., Table 6.에서 설명한 자료구조의 엔트리들로 채워진다. 이제 기준 식별자 별로 수행한 연산을 찾아갈 수 있고 그 연산들이 의미하는 것을 행위 엔진으로 매핑할 수 있다.

이제 스레드 생성 테이블을 가지고 리모트스레드의 생성 체인을 확인할 수 있고 기준 식별자로 취합한 모니터링 데이터를 가지고 다양한 프로세스와 스레드가 수행한 연산을 분석할 수 있다. Fig. 6.은

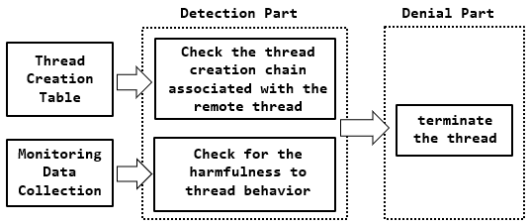


Fig. 6. Overview of thread behavior based detection and denial

이러한 정보를 바탕으로 하여 리모트스레드와 그 파생 스레드에 대한 유해성을 확인하여 치료하는 내용을 나타내고 있다.

IV. 프로토타입 테스트

본 논문에서 제시하는 내용을 검증하기 위해서 세 개의 랜섬웨어 샘플을 사용하였다. 이 샘플들은 최근 이슈가 되고 있는 랜섬웨어 중에서 리모트스레드 생성 방식으로 동작하는 것을 추려낸 것이다. 이 샘플들에 대해서 정상적으로 그리고 안정적으로 차단하는지 프로토타입을 구현하여 확인하였다.

4.1 테스트 환경

랜섬웨어를 사용하여 검증하는 과정은 상당히 위험하기 때문에 안전한 테스트를 위해서 가상환경을 사용하였다. VMware 가상환경의 Win7 32비트 운영체제 환경에서 테스트하였으며 프로토타입에 사용된 모듈은 커널 레벨의 파일시스템 미니필터 드라이버와 이와 통신하는 응용 어플리케이션으로 구성하였다.

테스트를 위해서 특정 확장자를 파일이름의 뒤에 붙이는 경우를 악의적인 행위로 간주하여 랜섬웨어로 진단하도록 하였다. 랜섬웨어로 진단된 경우 스레드 생성 테이블을 사용하여 악성 행위의 주체와 리모트스레드의 생성 주체를 추적하여 프로세스 및 스레드의 종료와 숙주 파일을 삭제하는 연산을 수행하였다.

4.2 랜섬웨어 샘플

CryptoLocker[16]와 GandCrab[17], 그리고 마지막으로 SymmyWare[18]는 유명한 랜섬웨어이다. 이 랜섬웨어 샘플을 테스트하여 스레드 기반 모니터링으로 악의적인 연산을 수행하는 주체를 추적하고 안정적으로 차단할 수 있는지 확인한다. Table 9. 는 실제 사용한 랜섬웨어 샘플의 SHA256 해시 값을 나타낸다.

테스트에 사용한 샘플의 SHA256 값은 최초 실행할 파일의 이름으로 사용하였다.

Table 9. Ransomware samples

Name	SHA256
CryptoLocker	05e40c76ca10b44346290338eec2b771d0d257e6198df47adb92248b21c84e5f
GandCrab	416607c797962f487db469554905fe710fc7afbb15139aa5ce3fdb29c71cdca8
SymmyWare	da50730580bd7fe14fca5c3547eb54882b6f79b42cd474530b9b07dd5de4f1ac

4.3 랜섬웨어 테스트 결과

먼저 CryptoLocker 랜섬웨어 샘플을 실행하면 랜덤한 이름을 가지는 동일한 실행파일을 Temp 경로에 생성한다. 테스트에서는 “umiwbmj.exe” 라는 이름으로 숙주파일을 생성하였다. Fig. 7.은 미니필터 드라이버에서 수집된 정보를 응용 프로그램으로 출력한 것이다. Action 컬럼에 보이는 “CW” 는 랜섬웨어 프로세스가 umiwbmj.exe 파일을 Create, Write 했음을 의미한다. Fig. 7.은 치료가 완료된 상태를 보여주기 때문에 Status 컬럼에서 숙주파일이 삭제된 것으로 표시되었다.

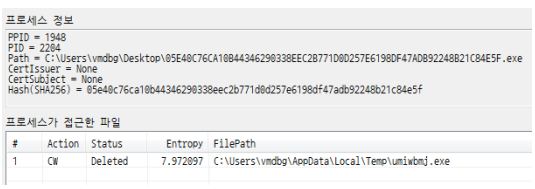


Fig. 7. Create the same executable

umiwbmj.exe 프로세스는 explorer.exe 프로세스에 리모트스레드를 생성하였다. 이 정보는 미니필터 드라이버에서 관리하는 스레드 생성 테이블에서

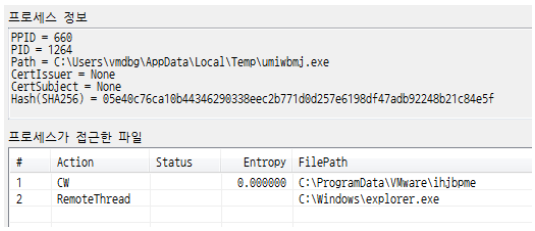
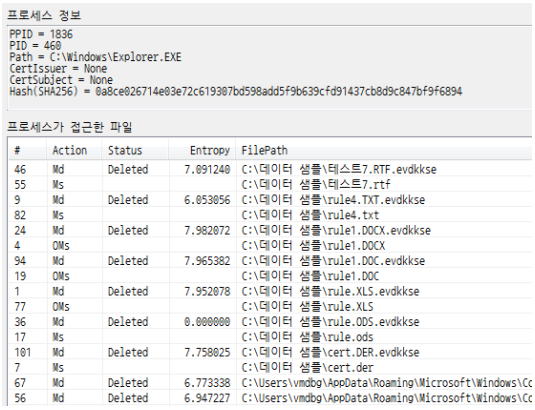


Fig. 8. umiwbmj.exe created a remote thread in explorer.exe

확인한 내용으로 Fig. 8.은 그 내용을 보여준다.

explorer.exe 에 생성된 리모트스레드에 의해서 악의적인 연산이 수행되었다. Fig. 9.는 리모트스레드의 생성시간을 기준 식별자로 사용하여 수집된 파일의 행위를 보여준다. 랜섬웨어의 암호화 타겟으로 놓아둔 문서 파일들이 이동되었고(Ms->Md) 최종적으로 이동된 파일의 이름에 "evdkkse" 라는 랜덤한 확장자를 붙이는 연산을 확인할 수 있다.



프로세스 정보

PPID = 1836
PID = 468
Path = C:\Windows\Explorer.EXE
CertIssuer = None
CertSubject = None
Hash(SHA256) = 0a8ce026714e03e72c619307bd598add5f9b639cfd91437cb8d9c847bf9f6894

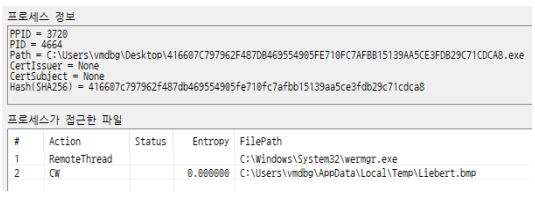
프로세스가 접근한 파일

#	Action	Status	Entropy	FilePath
46	Md	Deleted	7.091240	C:\데이터 샘플\테스트7.RTF.evdkkse
55	Ms			C:\데이터 샘플\테스트7.rtf
9	Md	Deleted	6.053056	C:\데이터 샘플\rule4.TXT.evdkkse
82	Ms			C:\데이터 샘플\rule4.txt
24	Md	Deleted	7.982072	C:\데이터 샘플\rule1.DOCX.evdkkse
4	OMs			C:\데이터 샘플\rule1.DOCX
94	Md	Deleted	7.965382	C:\데이터 샘플\rule1.DOC.evdkkse
19	OMs			C:\데이터 샘플\rule1.DOC
1	Md	Deleted	7.952078	C:\데이터 샘플\rule.XLS.evdkkse
77	OMs			C:\데이터 샘플\rule.XLS
36	Md	Deleted	0.000000	C:\데이터 샘플\rule.ODS.evdkkse
17	Ms			C:\데이터 샘플\rule.ods
101	Md	Deleted	7.758025	C:\데이터 샘플\cert.DER.evdkkse
7	Ms			C:\데이터 샘플\cert.der
67	Md	Deleted	6.773338	C:\Users\vmdbg\AppData\Roaming\Microsoft\Windows\Cc
56	Md	Deleted	6.947227	C:\Users\vmdbg\AppData\Roaming\Microsoft\Windows\Cc

Fig. 9. Malicious behavior is performed by the remote thread in explorer.exe

앞서 설명했듯이 테스트를 위해서 미니필터 드라이버는 파일 연산을 모니터링 하는 과정에서 파일이름에 특정 확장자를 붙이는 경우 악의적인 행위로 간주하도록 하였다.

결과적으로 Fig. 9. → Fig. 7. 의 과정으로 진단되고 치료되었다. 스레드 생성 테이블을 사용하여 리모트스레드의 연산을 수집하였고 그 연산으로부터 악의적인 행위를 판가름하여 악성으로 진단하였다. 그 결과 숙주파일인 umiwbmj.exe 존재를 확인할 수 있었고 이 파일을 생성한 주체인 랜섬웨어 실행파일을 확인할 수 있었다. 여기서 또 한 가지 중요한 것은 explorer.exe 프로세스 자체를 종료하지 않고 내부의 악의적인 스레드만 선별하여 종료하였다는 사실이다.



프로세스 정보

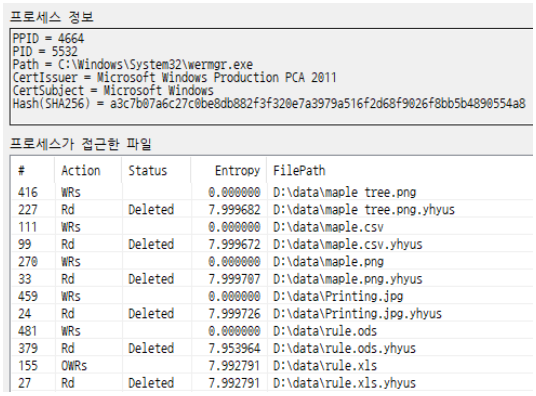
PPID = 3720
PID = 4664
Path = C:\Users\vmdbg\Desktop\416607C797962F487D8469554905FE710FC7AFBB15139AA5CE3FDB29C71CDCA8.exe
CertIssuer = None
CertSubject = None
Hash(SHA256) = 416607c797962f487db469554905fe710fc7afbb15139aa5ce3fdb29c71cdca8

프로세스가 접근한 파일

#	Action	Status	Entropy	FilePath
1	RemoteThread			C:\Windows\System32\wermgr.exe
2	CW		0.000000	C:\Users\vmdbg\AppData\Local\Temp\Liebert.bmp

Fig. 10. gandcrab created a remote thread in wermgr.exe

두 번째로 테스트한 샘플은 GandCrab 랜섬웨어이다. Fig. 10. 은 GandCrab 랜섬웨어가 실행된 후에 wermgr.exe 프로세스에 리모트스레드를 생성한 사실을 보여준다.



프로세스 정보

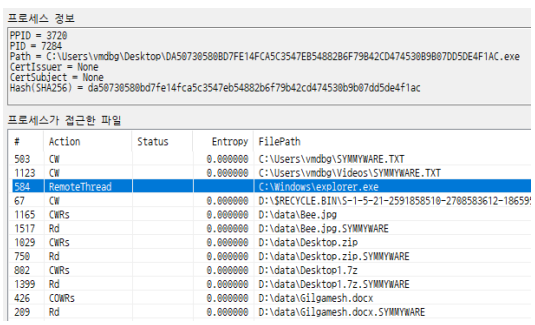
PPID = 4664
PID = 5532
Path = C:\Windows\System32\wermgr.exe
CertIssuer = Microsoft Windows Production PCA 2011
CertSubject = Microsoft Windows
Hash(SHA256) = a3c7b07a6c27c0be8db882f3f320e7a3979a516f2d68f9026f8bb5b4890554a8

프로세스가 접근한 파일

#	Action	Status	Entropy	FilePath
416	WRs		0.000000	D:\data\maple tree.png
227	Rd	Deleted	7.999682	D:\data\maple tree.png.yhyus
111	WRs		0.000000	D:\data\maple.csv
99	Rd	Deleted	7.999672	D:\data\maple.csv.yhyus
270	WRs		0.000000	D:\data\maple.png
33	Rd	Deleted	7.999707	D:\data\maple.png.yhyus
459	WRs		0.000000	D:\data\Printing.jpg
24	Rd	Deleted	7.999726	D:\data\Printing.jpg.yhyus
481	WRs		0.000000	D:\data\rule.ods
379	Rd	Deleted	7.953964	D:\data\rule.ods.yhyus
155	OWRs		7.992791	D:\data\rule.xls
27	Rd	Deleted	7.992791	D:\data\rule.xls.yhyus

Fig. 11. Malicious behavior is performed by the remote thread in wermgr.exe

Fig. 11. 은 wermgr.exe에 생성된 리모트스레드에 의해서 악의적인 행위가 어떻게 발생했는지 보여준다. 랜섬웨어의 암호화 타겟으로 놓아준 문서 파일들을 암호화(W)하고 이름변경하였고(Rs->Rd) 최종적으로 파일의 이름에 랜덤한 확장자인 "yhyus"를 붙였다. 앞서 설명한 CryptoLocker와 유사하게 동작했다는 것을 알 수 있다. 치료과정도 동일하다. 프로세스를 대상으로 하지 않고 wermgr.exe의 리모트스레드만을 종료하였기 때문에 wermgr.exe는 정상적으로 동작한다.



프로세스 정보

PPID = 3720
PID = 7284
Path = C:\Users\vmdbg\Desktop\DA50730580BD7FE14FCA5C3547EB54882B6F79B42CD474530B9B07DD5DE4F1AC.exe
CertIssuer = None
CertSubject = None
Hash(SHA256) = da50730580bd7fe14fca5c3547eb54882b6f79b42cd474530b9b07dd5de4f1ac

프로세스가 접근한 파일

#	Action	Status	Entropy	FilePath
503	CW		0.000000	C:\Users\vmdbg\SYMMYWARE.TXT
1123	CW		0.000000	C:\Users\vmdbg\Videos\SYMMYWARE.TXT
584	RemoteThread			C:\Windows\explorer.exe
67	CW		0.000000	D:\$RECYCLE.BIN\S-1-5-21-2591858510-2708583612-18659
1165	CWRs		0.000000	D:\data\Bee.jpg
1517	Rd		0.000000	D:\data\Bee.jpg.SYMMYWARE
1029	CWRs		0.000000	D:\data\Desktop.zip
750	Rd		0.000000	D:\data\Desktop.zip.SYMMYWARE
882	CWRs		0.000000	D:\data\Desktop1.7z
1399	Rd		0.000000	D:\data\Desktop1.7z.SYMMYWARE
426	COWRs		0.000000	D:\data\Gilgamesh.docx
289	Rd		0.000000	D:\data\Gilgamesh.docx.SYMMYWARE

Fig. 12. SymmyWare created a remote thread in explorer.exe

마지막으로 Fig. 12.는 SymmyWare 랜섬웨어가 explorer.exe 프로세스에 리모트스레드를 생성하였다는 사실을 보여준다. 그리고 SymmyWare는 리모트스레드에 의해서 랜섬웨어 동작을 수행함과 동시에 인젝션을 수행한 프로세스도 랜섬웨어 동작을 수행했다는 것을 알 수 있다. 디렉터리마다 "SYMMYWARE.TXT" 라는 이름의 랜섬노트를 생성했고 그림 파일, 압축 파일, 문서 파일의 확장자 뒤에 "SYMMYWARE" 붙임으로써 파일이 암호화 되었다는 표시를 했다. 앞서 CryptoLocker, GandCrab과 같은 방식으로 기존 파일에 특정 확장자를 붙이는 경우로 판단하여 악의적인 행위로 간주하여 해당 프로세스를 종료하였고 explorer.exe 프로세스에 생성된 리모트스레드도 선별적으로 종료하였기 때문에 explorer.exe는 정상적으로 동작한다.

V. 결 론

본 연구를 통하여 프로세스 기반 모니터링으로 해결하기 난해한 문제에 대해서 설명하였고 그 문제에 대한 해결책을 제시하였다. 이처럼 스레드 기반 모니터링을 사용하면 세밀한 모니터링이 가능하기 때문에 이를 보완하고 발전시키면 행위 기반 엔진으로 검사할 수 있는 대상을 보다 구체적으로 명시할 수 있을 것으로 기대한다.

이와 관련하여 제작된 프로토타입은 하나의 커널 드라이버와 하나의 응용 프로그램으로 구성된 아주 단순한 구조를 가지고 있고 내부적으로 하드코딩된 부분도 많다. 때문에 다양한 샘플을 테스트하기에 다소 불편함이 있다. 앞으로의 연구는 프로토타입을 보다 유연하게 구성하여 테스트 자동화 방식으로 진행할 계획이다.

마지막으로 본 논문에서 다루지 않은 네트워크 모니터링이나 커널 드라이버에서는 수집할 수 없는 자원들에 대해서도 추가적으로 연구하여 보다 강화된 모니터링 수집 기법 및 행위기반 진단 기법을 연구할 계획이다.

References

[1] Myungcheol Lee, Daesung Moon and Ikkyun Kim, "Real-time Abnormal Behavior Detection System based on Fast Data," *Journal of The Korea Institute of information Security & Cryptology*, 25(5), pp. 1027-1041, Oct. 2015.

[2] Microsoft Docs, "Filter Manager and Minifilter Driver Architecture" https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-and-minifilter-driver - architecture/, Jan. 7 2020.

[3] Microsoft Docs, "Process Monitor" https://docs.microsoft.com/en-us/sysinternals/downloads/procmon/, Jan. 7 2020.

[4] Microsoft Docs, "CmRegisterCallbackEx" https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-cmregistercallbackex/, Jan. 7 2020.

[5] Microsoft Docs, "PsSetCreateProcessNotifyRoutineEx" https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-pssetcreateprocessnotifyroutineex/, Jan. 7 2020.

[6] Microsoft Docs, "PsSetCreateThreadNotifyRoutine" https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-pssetcreatethreadnotifyroutine/, Jan. 7 2020.

[7] Wikipedia, "Fileless malware" https://en.wikipedia.org/wiki/Fileless_malware, Jan. 7 2020.

[8] Trendmicro, "Command and Control [C&C] Server" https://www.trendmicro.com/vinfo/us/security/definition/command-and-control-server, Jan. 7 2020.

[9] Red Teaming Experiments, "Reflective DLL Injection" https://ired.team/offensive-security/code-injection-process-injection/reflective-dll-injection, Jan. 7 2020.

[10] Microsoft Docs, "CRITICAL_OBJECT_TERMINATION" https://docs.microsoft.com/en-us/windows-hardware/drivers/

debugger/bug-check-0xf4--critical-object-termination, Jan. 7 2020.
〔11〕 Raymond J. Canzanese, Jr, "Detection and classification of malicious processes using system call analysis," Doctor of Philosophy, Drexel University, May 2015.
〔12〕 Microsoft Docs, "PsGetCurrentProcessId" https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-psgetcurrentprocessid/, Jan. 7 2020.
〔13〕 Microsoft Docs, "PsGetCurrentThreadId" https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/nf-ntddk-psgetcurrentthreadid/, Jan. 7 2020.
〔14〕 Microsoft Docs, "CreateRemoteThread" https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread/, Jan. 7 2020.
〔15〕 Wikipedia, "Hash table" https://en.wikipedia.org/wiki/Hash_table, Jan. 7 2020.
〔16〕 Wikipedia, "CryptoLocker" https://en.wikipedia.org/wiki/CryptoLocker, Jan. 7 2020.
〔17〕 Malwarebytes, "GandCrab" https://www.malwarebytes.com/gandcrab/, Jan. 7 2020.
〔18〕 PCrisk, "SymmyWare" https://www.pcrisk.com/removal-guides/13980-symmyware-ransomware, Jan. 7 2020.

〈저자소개〉

고 보 승 (Boseung Ko) 정회원
2004년 2월: 제주대학교 전자공학과 학사
〈관심분야〉 정보보호, 해킹, 실시간 빅데이터 분산처리

최 원 혁 (Wonhyok Choi) 정회원
1997년 2월: 경일대학교 컴퓨터공학과 학사
2001년 2월: 동국대학교 정보보호학과 석사
2015년 2월: 동국대학교 정보통신공학과 컴퓨터전공 박사과정 수료
〈관심분야〉 디지털 포렌식, 악성코드 분석, 해킹, 정보통신, 정보보호

정 다 정 (Dajung Jeong) 정회원
2011년 2월: 호원대학교 사이버수사경찰학부 학사
2018년 2월: 성균관대학교 정보보호학과 석사
〈관심분야〉 디지털 포렌식, 데이터 복원, 정보통신, 정보보호