METASPLOIT FORENSIC

Meterpreter Artifact

MaJ3stY
http://maj3sty.tistory.com
2013-01-20

1. 서론

해당 문서에서 살펴 볼 Metasploit Framework는 침투테스트용 해킹 프레임워크로 전세계 모의 침투 테스터들과 해커들에게 많은 사랑을 받고 있는 오픈 소스 도구이다. 현재 루비 스크립트 언어로 제작되고 있으며, 신규 취약점이 발표 될 때 마다 빠른 스크립트 포팅 지원으로 최신 공격을 쉽게 할 수 있는 장점이 있는 반면, 모든 해킹 도구들이 그렇듯이 해당 도구 또한 악의적인 목적으로 사용될 시 그 파급력은 사용 범위와 사용목적 등에 따라 달라진다. 공격을 위한 간단한 취약한 웹 서버 페이지 제작부터 백도어 등 여러 가지 해킹에 필요한 부수적인 도구를 손 쉽게 만들 수 있는 장점이 있어 실제 해킹에서도 많이 사용되고 있는 것이 바로 Metasploit 도구인 만큼 해당 문서에서는 포렌식 관점에서 Metasploit 도구가 피해자 컴퓨터에 어떤 흔적을 남기는지 살펴 볼 예정이다.

2. How Meterpreter work

Metasploit 도구에서 공격에 가장 많은 직접적 기능을 제공하는 것은 Meterpreter라고 하는 인터프리터 도구이다. Meterpreter는 피해자 컴퓨터와 공격자 컴퓨터를 연결 시켜주는 기능을 제공하고, 그 연결을 통해 여러 가지 행위를 할 수 있도록 여러 기능들을 제공한다.[1] Meterpreter는 제작 초기부터 몇 가지 디자인 목적을 가지고 있다. 이 이유는 완벽에 가까운 공격을 수행하기 위함인데, 그 목표는 다음과 같다.

- *Meterpreter resides entirely in memory and writes nothing to disk.*
- *No new processes are created as Meterpreter injects itself into the compromised process and can migrate to other running processes easily.*[2]
- *By default, Meterpreter uses encrypted communications.*
- *All of these provide limited forensic evidence and impact on the victim machine.*

목표를 보면, 디스크가 아닌 메모리에서 동작하며, 개별 프로세스를 생성하지 않고, 기본 통신으로 암호화 통신을 채택하고 있다. 마지막으로 포렌식 증거로서의 제한을 목표로 하고 있다.
마지막 목표는 보는 관점에서 그 해석이 조금씩 달라질 수 있는 부분인데, 이 목표에서 언급하는 포렌식 증거는 Meterpreter 자체의 흔적이 아닌 Meterpreter에서 지원하는 안티 포렌식 기능을 의미한다. Meterpreter에서 지원하는 안티 포렌식 기능은 목표에서 언급하는 것만큼 강력하지 못하다. 그러므로 Meterpreter가 남기는 흔적만 찾아낸다면, 충분히 포렌식 관점에서 사건을 재구성하고, 증명 할 수 있다.
그렇다면, Meterpreter는 어떻게 동작할까? 이번 절에서는 간단하게 Meterpreter 동작 과정에

---

[1] http://www.offensive-security.com/metasploit-unleashed/Meterpreter_Basics
http://scadahacker.com/library/Documents/Cheat_Sheets/Hacking - Meterpreter Cheat Sheet.pdf

[2] Migrate 명령에서는 Reflective DLL Injeciton 기법이 사용 된다.

대해서 알아보고자 한다. 이 문서에서 언급하는 Meterpreter의 동작 환경은 다음과 같다.

[공격자]
OS : Backtrack5 R3 32bit
payload : exploit/multi/handler

[피해자]
OS : Windows 7 32bit
*Payload : Msfvenom(x86/shikata_ga_nai*5, reverse_tcp)*

2.1. TLV Packet

Meterpreter는 기본적으로 암호화 통신을 한다고 하였다. 이는 Meterpreter 소개 글에도 나와 있는 내용으로 TLS/1.0을 이용 해 명령 패킷들을 암호화하여 주고 받는다. 이때 명령을 주고 받을 때 사용되는 프로토콜은 TLV(Type-Length-Value) 프로토콜이다. TLV 프로토콜은 굉장히 간단한 형태를 가지고 있는데, Meterpreter에서는 해당 프로토콜 패킷을 통해 명령지시와 명령 결과를 주고 받으며 기능을 수행한다. 다음 그림은 TLV 패킷의 Format을 나타낸 그림이다.

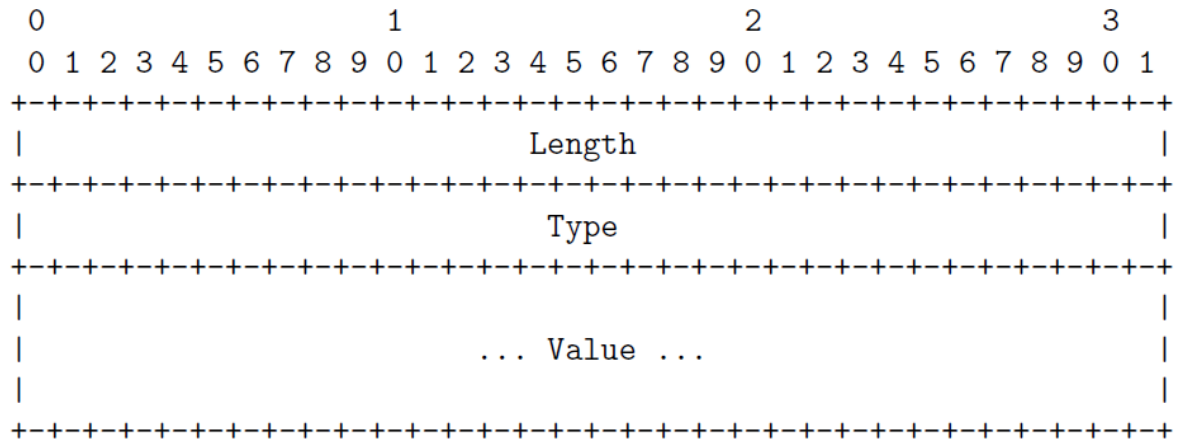


그림 1 TLV Format

Length : TLV 패킷의 길이는 나타내는 필드로 32bit 필드이며, Length, Type, Value 필드의 길이를 모두 포함한다.

Type : TLV 패킷의 형식을 나타내는 필드로 32bit이다.

Value : TLV 패킷이 전송하는 값들을 나타내는 필드로 32bit이다.

2.2. Meterpreter Core

Meterpreter의 핵심 기능은 과연 어떻게 구성되어 있을까? 의외로 구성은 간단하다. 두 개의 dll 파일이 Meterpreter의 핵심기능을 담당하고 있으며, 부수적으로 몇 개의 dll 파일이 추가기능을 구현하도록 구성되어 있다. 다음은 Meterpreter를 구성하고 있는 핵심 dll 설명이다.

- *Metsrv.dll : Meterpreter가 통신하기 위한 암호화 통신과 채널을 생성하고 연결시켜주는 역할을 하는 라이브러리 파일이다.*

- *ext_server_stdapi.dll : Meterpreter의 기능을 구현하는 라이브러리 파일이다.*

기본적으로 두 파일은 Meterpreter가 동작할 때 꼭 필요한 파일로, 피해자 컴퓨터 프로세스에서 필히 사용하고 있는 라이브러리 파일이다. 해당 파일들의 내용을 한번 살펴보자. 다음은 Metsrv.dll 파일을 IDA로 열었을 때 보이는 여러 문자열들이다. 문자열들 중 TLS 프로토콜과 관련된 여러 문자들을 볼 수 있다.

그림 2 TLS 경고 프로토콜 문자열

또 ext_server_stdapi.dll의 경우 Meterpreter에서 지원하는 기능들의 함수들을 쉽게 찾아 볼 수 있다.

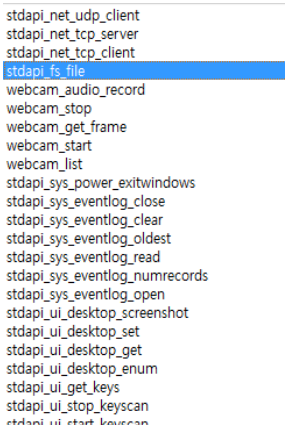


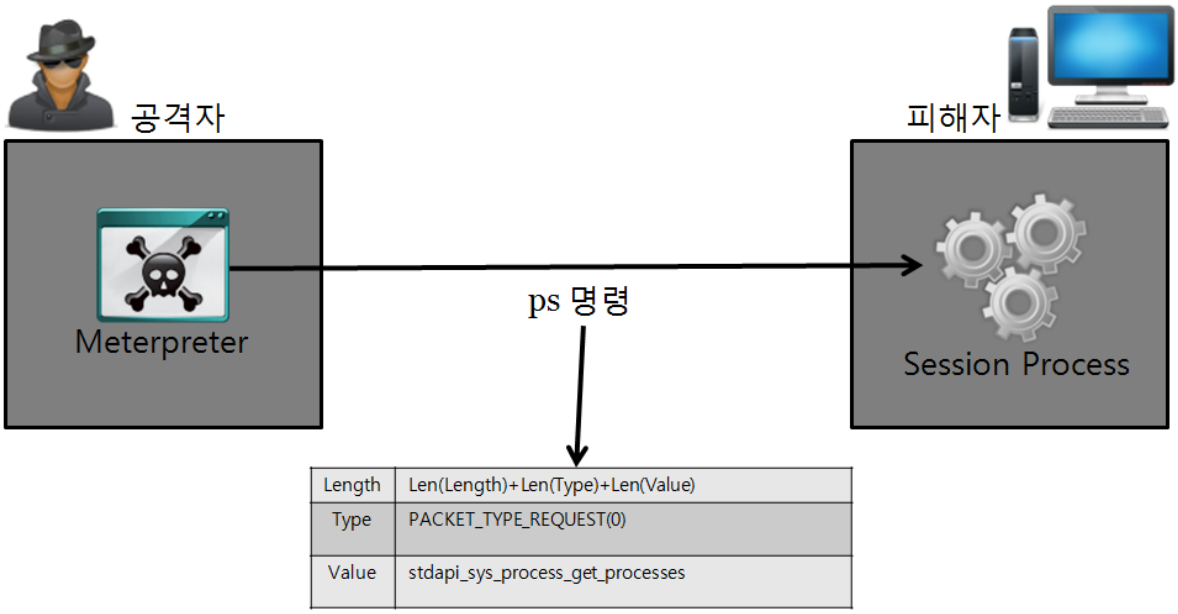
그림 3 Meterpreter 기능들의 함수

함수들의 이름만 보아도 무슨 동작을 하는지 대부분 파악이 될 정도로 함수들의 이름이 직관적

이다. 사실 여기까지만 알았더라도 어느 정도 Meterpreter가 어떻게 동작하는지 감이 올 것이다.

2.3. Meterpreter Work

이제부터 동작 과정에 대해서 살펴보자. 여기서의 동작과정은 공격자가 피해자 컴퓨터에 있는 Meterpreter 세션 프로세스에게 ps 명령을 내렸을 때 동작하는 과정이다. 다음은 세션이 성립되어 있는 상태에서 ps 명령이 전송되는 모습이다.



Length	Len(Length)+Len(Type)+Len(Value)
Type	PACKET_TYPE_REQUEST(0)
Value	stdapi_sys_process_get_processes

그림 4 ps 명령 전송

패킷의 구조는 앞서 설명 했던 TLV 구조로 패킷 길이와 패킷의 형태, 값이 전송 되는데 패킷의 형태는 명령을 요청하는 것이므로 meterpreter/packet.rb에 정의되어 있는 PAKCET_TYPE_REQUEST 이다. (0)은 값을 나타낸다. 그리고 Value에는 ext_server_stdapi.dll 에서 실행 할 함수를 나타내고 있는데, ps 명령은 Process 목록을 얻어 오는 명령이므로 이 명령에 해당 하는 stdapi_sys_process_get_process() 함수를 호출할 목적으로 value에 해당 함수 호출 값을 넣어 전송 한다. 그 다음 과정을 살펴보자.

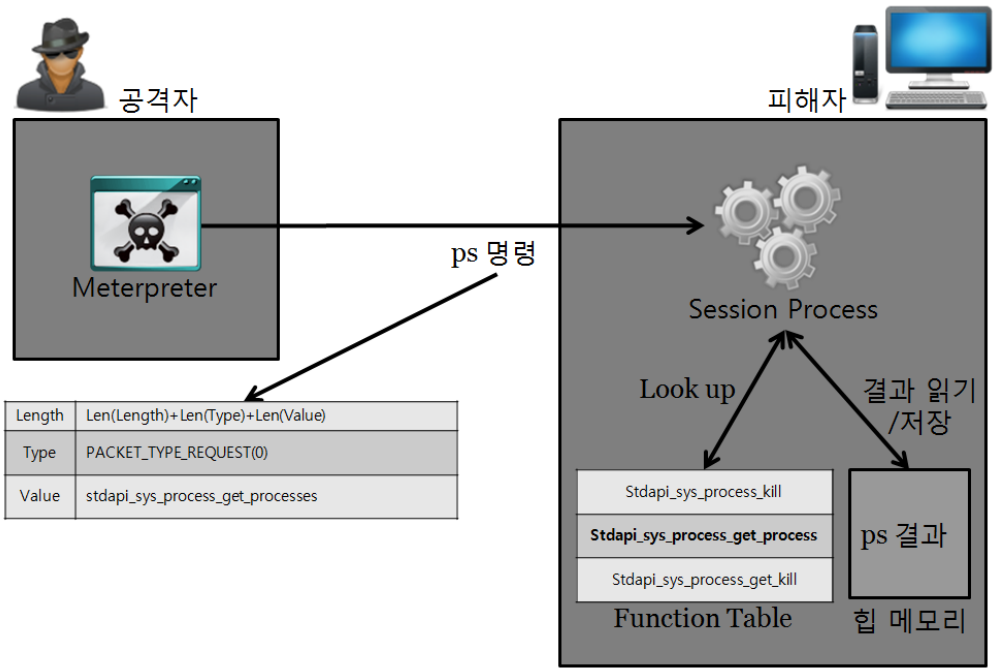


그림 5 ps 명령 처리

Meterpreter와 연결 된 세션 프로세스는 TLV 명령 패킷을 해석 해 그 요청에 맞는 동작을 하게 된다. 위 과정에서는 ps 명령 전송으로 stdapi_sys_process_get_process() 함수가 전달 되었으므로 세션 프로세스는 ext_server_stdapi.dll에 있는 Function Table에서 해당 함수를 찾아 호출 할 것이다. 함수가 실행되는 과정에서 세션 프로세스는 메모리에 힙 영역을 설정 한 후 함수 결과를 잠시 저장해 둔다. 함수 실행이 모두 끝이 나면 다음과 같은 과정이 진행 된다.

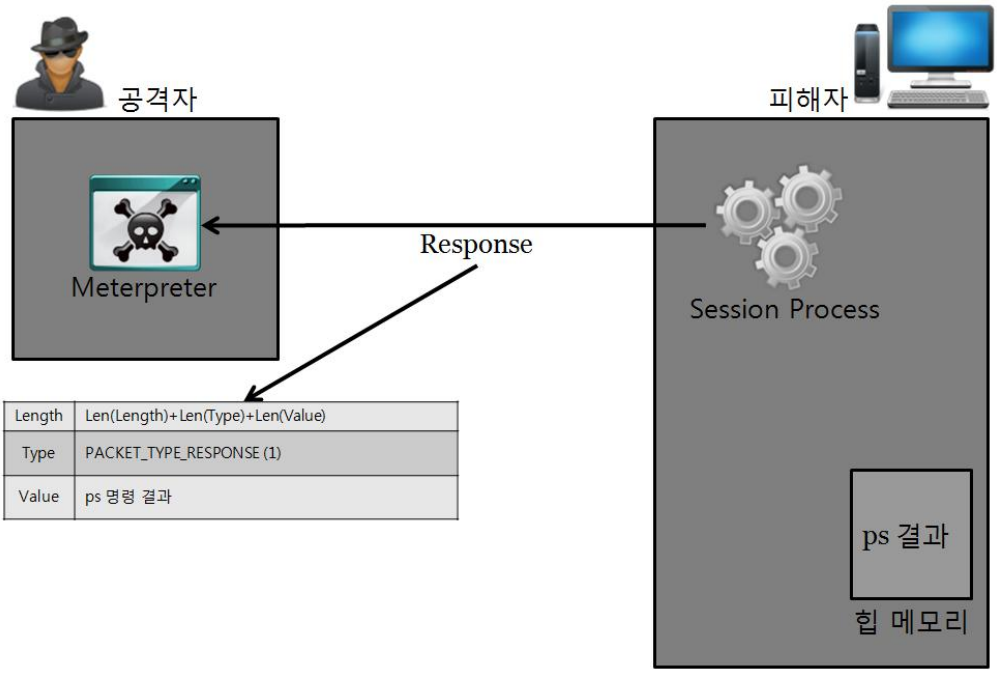


그림 6 명령 결과 전송

세션 프로세스는 Response를 준비하는 과정에서 packet_create_response() 함수를 호출한다. 해당 함수가 호출 되면 패킷이 생성되는데 패킷의 Value는 세션프로세스가 소유하고 있는 힙 영역에서 가져오게 된다. 그 다음 패킷 생성이 모두 완료 되면 세션 프로세스는 packet_transmit() 함수를 호출 해 패킷을 전송하게 된다. 이때 힙 메모리 영역에는 명령의 결과와 Value값이 고스란히 저장되어 있다.

여기서 한가지 주의해야 할 점은 그림에서 Value를 표현 해 놓은 것을 보면 간단히" stdapi_sys_process_get_process" 또는 "ps 명령 결과"로 표현 했지만, 사실 Value에는 여러 가지 TLV 패킷이 그룹화 되어 들어가 있다. 그러므로 Value 값이 단일 값이라는 오해는 하지 않아야 한다.

3. Artifact

그럼 이제 메모리에서 한번 흔적을 찾아보도록 하자. 위 과정 설명에서 우리가 제일 눈 여겨 봐야 할 부분은 힙 메모리가 할당 되고 명령의 결과가 힙 메모리 영역에 저장되며, 그 값은 지워지지 않는다는 부분이다. 메모리를 조금만 살펴보면 다음과 같이 Function Table 영역을 발견 할 수 있다.

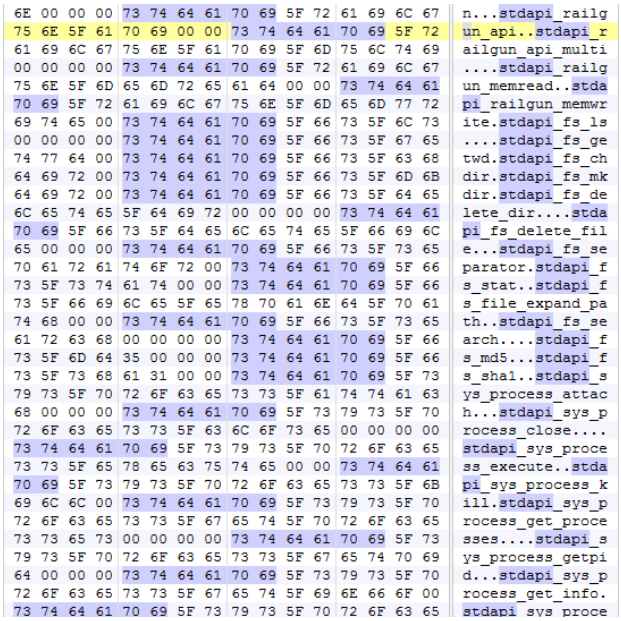


그림 7 Function Table

Function Table 다음으로는 Response 패킷이 저장되어 있는 힙 메모리 영역이 존재한다.

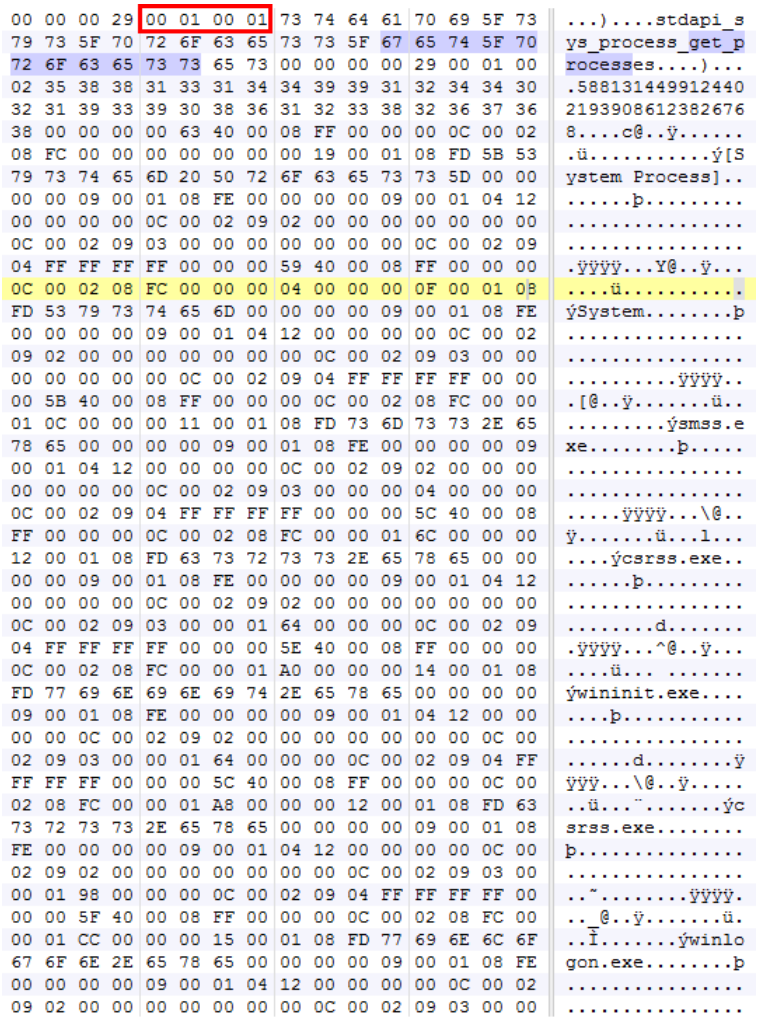


그림 8 Response 패킷

[그림 8]을 보면 호출 함수 명이 보이고 그 다음부터는 명령의 결과들이 저장되어 있는 것을 볼 수 있다. 실제 Meterpreter의 명령 결과와 비교해보면 동일하다는 것을 알 수 있다.

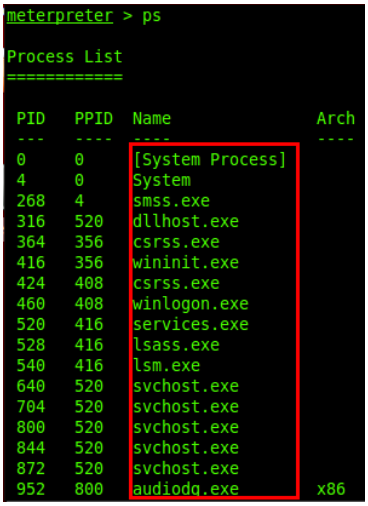


그림 9 Meterpreter 명령 결과

[그림 8]에 빨간 박스로 강조되어 있는 부분이 TLV 구조 중 Type에 해당 되는 부분이다. 값을 보면 0x00010001(TLV_TYPE_METHOD)인데, 이 값은 meterpreter/packet.rb에 저장되어 있다.

```
TLV_TYPE_ANY             = TLV_META_TYPE_NONE   |   0
TLV_TYPE_METHOD          = TLV_META_TYPE_STRING |   1
TLV_TYPE_REQUEST_ID      = TLV_META_TYPE_STRING |   2
TLV_TYPE_EXCEPTION       = TLV_META_TYPE_GROUP  |   3
TLV_TYPE_RESULT          = TLV_META_TYPE_UINT   |   4
```

그림 10 TLV TYPE

TLV_TYPE_METHOD = TLV_META_TYPE_STRING | 1 에서 TLV_META_TYPE_STRING은 TLV_META_TYPE_STRING = (1 << 16)로 계산하면 65536이다. 65536 | 1을 하게 되면 65537이 되고 이를 hex값으로 변환하면 0x00010001이 된다.

TLV Type 필드 바로 앞 4byte는 TLV 패킷의 길이는 나타내는 Length 필드로 0x00000029 값을 보여주고 있다.

앞서 말했듯이 이 TLV 패킷만이 Response 패킷의 Value가 아니다. 그 다음으로 공격자가 보낸 REQUEST 패킷의 ID를 나타내는 TLV 패킷(TLV_TYPE_REQUEST_ID, 0x00010002)이 존재한다. 나머지 Value는 앞서 봤던 명령 결과에 해당 된다.

4. 결론

사실 앞에서 설명한 것 외에 Metasploit을 개발한 Rapid 7에서 자신만만하게 포렌식 증거로서의 제한을 언급한 이유는 메모리의 휘발성 때문이다. Meterpreter는 오로지 메모리에서만 상주하고 메모리에서만 실행 되기 때문에 침해사고가 일어난 시점으로부터 1시간여만 지나도 메모리 크기가 작은 시스템에서는 비 할당 메모리 영역이 금방 다른 프로세스 메모리 영역으로 할당 되기 때문에 데이터가 덮어 씌어져 그 흔적을 찾아보기가 힘들다. 현실적으로 침해사고가 일어난 후 1시간 이내에 대응을 하기란 사실 힘든 일이다.

이런 이유로 Meterpreter의 흔적을 메모리에서 찾기란 굉장히 어려운데, 방법은 아예 없는 것은 아니다. 첫 번째 가능성이 있는 방법으로는 먼저 포렌식 준비도가 선행되어야 하고 포렌식 준비도에 메모리보존과 관련된 지침이 명시되어야 한다. 그 다음, 포렌식을 수행하는 자는 포렌식 준비도에 의해 보존 된 메모리 파일 또는 pagefile.sys, crash dump 등의 OS가 덤프 하여 둔 메모리 덤프파일을 분석해 보아야 한다. crash dump 파일은 메모리 덤프 중 가장 순수한 덤프파일이고, 공격 도중 crash가 일어났을 경우 OS가 자동으로 메모리를 덤프 파일로 생성해 주기 때문에 Meterpreter가 연결되어 있는 상태의 메모리 내용을 담고 있을 가능성이 가장 많다. pagefile.sys 또한 page의 swap이 일어나면서 Meterpreter의 흔적도 같이 swap 될 가능성이 있기 때문에 Meterpreter의 흔적을 찾는 목적이 있다면 꼭 분석을 수행해 봐야 한다.