情 報 保 護 學 會 誌
第18卷 第3號, 2008. 6

다형성 바이러스의 소개(Win32/Kashu)

최 동 균*, 양 하 영*

요 약

본 논문에서는 최근 국내에서 발견된 바이러스 중 다형성 바이러스인 Win32/Kashu에 대한 특징을 설명하고자 한다. 이를 통해 다형성 바이러스의 일반적인 특징을 알 수 있으며, 나아가 안티바이러스 제품에서 어떻게 치료가 이루어지는지를 알 수 있다. 다형성 바이러스는 일반적인 바이러스와 달리 감염 시 자신의 코드가 매번 다른 형태를 나타내는 것으로 진단과 치료가 어려운 것이 특징이다. 최근에는 이러한 복잡한 형태의 바이러스가 점점 증가하고 있는 추세이며, 이러한 현상은 사용자 및 안티바이러스 업체 모두에게 큰 위협으로 다가올 수 있다.

Ⅰ. 서 론

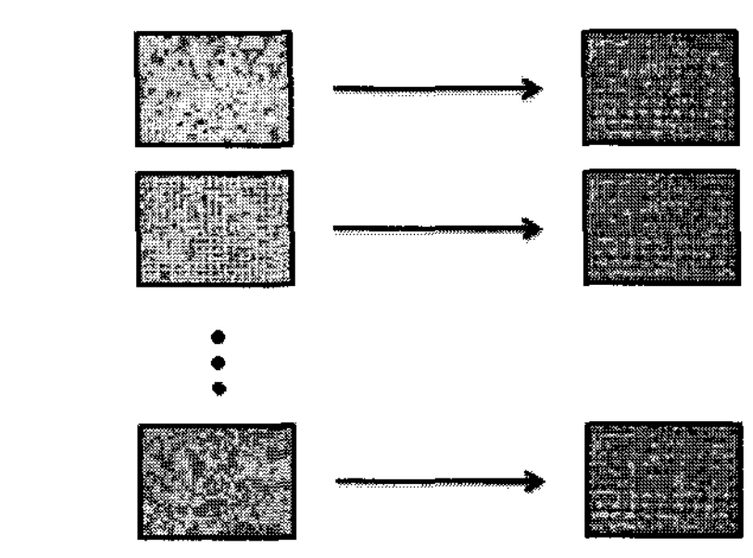
악성코드는 제작자가 의도적으로 사용자에게 피해를 주고자 만든 모든 악의적 목적을 가진 프로그램을 의미한다. 이러한 악성코드는 수행하는 기능 및 형태에 따라 트로이목마(Trojan), 웜(Worm), 바이러스(Virus)등으로 구분할 수 있다. 이 중 바이러스는 사용자 시스템의 정상파일을 수정하여 자신의 코드를 삽입하는 것으로 감염형태 및 기법에 따라 다양하게 분류가 가능하다. 바이러스 제작자는 자신이 제작한 바이러스가 진단되는 것을 어렵게 하기위해 여러 가지 다양한 기법을 사용하고 있으며. 이러한 기법들은 점점 더 진화하고 있다. 이러한 목적을 위해 가장 일반적으로 사용되는 방법은 자신의 코드를 암호화하는 것으로 이는 바이러스 이외의 다른 악성코드에도 널리 사용되고 있다. 이러한 암호화된 형태의 바이러스도 암호화 바이러스(Encrypted Virus)와 다형성 바이러스(Polymorphic Virus), 메타몰픽 바이러스(Metamorphic Virus)등으로 구분되며, 이 중 다형성 바이러스는 단순한 형태의 암호화 바이러스와 달리 수백만 가지 이상의 변형을 생성해 낼 수 있는 조작엔진(Mutation Engine)을 갖고 있는 복잡한 형태를 나타낸다. 최근 국내에서 발견된 Win32/Kashu 바이러스는 다형성 바이러스이면서 정상파일의 일부분을 덮어쓰는 후위형 바이러스이다.

본 논문에서는 Win32/Kashu 바이러스의 감염형태 및 복호화 방식에 대해 살펴보고 어떻게 치료가 이루어지는지를 살펴보고자 한다.

Ⅱ. 감염형태

2.1 다형성 바이러스

다형성 바이러스(Polymorphic Virus)는 하나의 바이러스 본체(Body)가 암호화되어 수백만 가지 이상의 다양한 형태로 자신을 변형시킬 수 있는 바이러스이다. 또한, 다형성 바이러스는 암호화기법 외에 쓰레기 명령어(Garbage Instruction)의 삽입을 통해 실제로 자신이 사용하는 유효한 코드를 식별하기 어렵게 한다. 아래의 [그림 1]은 하나의 다형성 바이러스에 의해 생성된 서



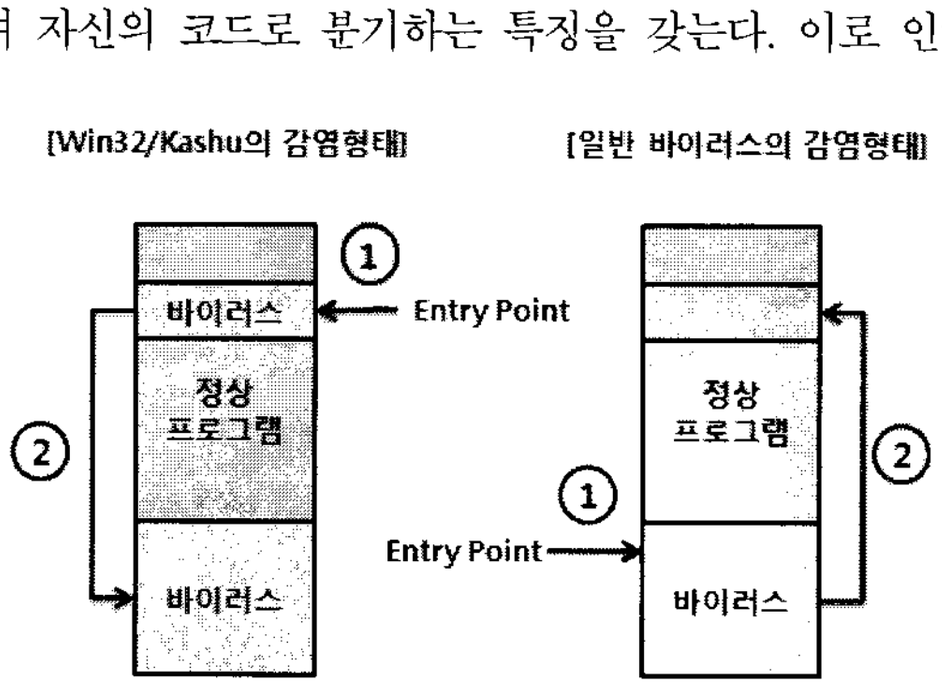
〔그림 1〕 복호화 후 다형성바이러스 본체

* 안철수연구소 ASEC팀 주임연구원 (cdk@ahnlab.com)
** 안철수연구소 ASEC팀 주임연구원 (hyyang@ahnlab.com

로 다른 형태의 개체들이 복호화 작업 이후에는 동일한 하나의 바이러스 본체를 갖는 것을 나타낸다. Win32/Kashu 바이러스도 매 감염 시 서로 다른 형태의 개체들을 생성할 수 있으며, 쓰레기 명령어의 삽입으로 복호화에 필요한 유효 명령어를 찾기 어렵게 한다.

2.2 EPO 바이러스

일반적인 바이러스는 감염 시 자신의 코드부분을 먼저 수행하도록 하기위해 원본파일의 시작주소를 나타내는 엔트리 포인트(Entry Point) 값을 자신의 코드 중 일부로 변경하는 형태를 갖는다. 이를 통해 바이러스에 감염된 파일은 프로그램의 시작에 대한 제어를 바이러스에 넘기고 원본 파일은 바이러스 코드가 모두 수행된 이후에 동작하게 된다. 이러한 엔트리 포인트 주소 값의 수정을 통한 일반적인 바이러스의 감염기법과 달리 EPO(Entry-Point Obscuring)바이러스는 정상파일의 엔트리 포인트 주소 값을 변경하지 않은 채 바이러스 코드로 분기할 수 있도록 하는 기법이다. 가장 일반적으로 사용되는 EPO기법은 정상 프로그램의 명령어 중 특정 명령어(Call명령, ExitProcess() API호출 명령 등)를 수정하여 정상 프로그램의 동작 도중 혹은 종료시점에 바이러스 시작주소로 분기하도록 하는 것이다. Win32/Kashu 바이러스는 [그림 2]와 같이 EPO바이러스로 정상파일의 엔트리 포인트 주소 값을 수정하지 않고, 자신의 코드로 분기하도록 하는 특징을 갖는다. 특이한 점은 EPO기법 중 정상 프로그램의 특정 명령어를 수정하는 방식이 아닌 엔트리 포인트 주소 부분의 코드 중 일부분을 덮어쓰는 형태를 취하는 점이다. 후위에 감염된 바이러스 본체부분으로 분기하기 위해 덮어쓴 코드는 감염 파일마다 가변적이며, 마지막에 JMP명령을 사용하여 자신의 코드로 분기하는 특징을 갖는다. 이로 인해 Win32/Kashu 바이러스에 감염된 파일을 치료하기 위해서는 덮어쓴 정상 프로그램의 일부 코드 부분을 복구해주는 작업이 필요하다.



〔그림 2〕 EPO 기법을 이용한 감염형태

Win32/Kashu 바이러스에 의해 손상된 원본코드는 후위에 존재하는 바이러스 코드부분에 암호화된 형태로 존재하며, 자신이 덮어쓴 원본코드의 크기와 위치정보가 함께 저장되어 있다. 아래의 [표 1]과 [표 2]는 각각 Win32/Kashu 바이러스에 감염 전, 후의 "notepad.exe" 파일의 시작부분의 코드를 나타낸다.

〔표 1〕 감염 전 "notepad.exe" 파일의 엔트리 포인트 시작부분 코드

0100739D 6A70	PUSH 70
0100739F 68 98180001	PUSH 01001898
010073A4 E8 BF010000	CALL 01007568
010073A9 33DB	XOR EBX, EBX
...(중간 생략)	
0100747D 8D45DC	LEA EAX, [EBP-24]
01007480 50	PUSH EAX
01007481 FF35 AC9A0001	PUSH [1009AAC]
01007487 8D45 D4	LEA EAX, [EBP-2C]
0100748A 50	PUSH EAX
0100748B 8D45 D0	LEA EAX, [EBP-30]
0100748E 50	PUSH EAX
...(이하 생략)	

[표 1]에서 정상적인 "notepad.exe" 파일의 엔트리 포인트 주소 값은 0x0100739D임을 알 수 있으며, [표 2]에서 엔트리 포인트 시작 부분(0x0100739D)부터 0x01007488까지의 코드가 바이러스에 의해 덮어써진 것을 확인 할 수 있다.

[표 2]의 바이러스 코드 중 (A)위치의 "JMP ECX" 명령어에 의해 분기하는 곳은 [그림 2]에서 정상 프로그램의 후위에 존재하는 바이러스 본체 부분이다. 앞에서 언급한 것과 같이 정상 프로그램의 시작부터 덮어쓰는 코드의 크기는 감염 파일마다 가변적이며, 코드도 가변적임을 알 수 있다. [표 2]의 바이러스 코드 중 "JMP ECX" 명령어에 의해 분기하는 위치의 주소 값이 저장된 ECX 레지스터의 값은 다양한 연산과정을 거쳐 얻어진 값이며, 해당 ECX 값을 얻기 위해 필요한 명령어 이외의 모든 명령어는 쓰레기 명령어이다. 바이러스 본체로 분기하여 수행하는 작업은 암호화된 바이러스 본체

〔표 2〕 감염 후 "notepad.exe 파일의 엔트리 포인트 시작부분 코드

```
0100739D 60              PUSHAD
0100739E 33EE            XOR EBP, ESI
010073A0 0FADD8          SHRD EAX, EBX, CL
010073A3 0FAFFE          IMUL EDI, ESI
010073A6 0FADFD          SHRD EBP, EDI, CL
010073A9 6A00            PUSH 0
...(중간 생략)
0100747C D1D6            RCL ESI, 1
0100747E 29D5            SUB EBP, EDX
01007480 87F0            XCHG EAX, ESI
01007482 85DA            TEST EDX, EBX
01007484 FFC6            INC ESI
01007486 FFE1            JMP ECX ;Go Virus (A)
01007488 45              INC EBP
01007489 D450            AAM 50
0100748B 8D45D0          LEA EAX, [EBP-30]
0100748E 50              PUSH EAX
...(이하 생략)
```

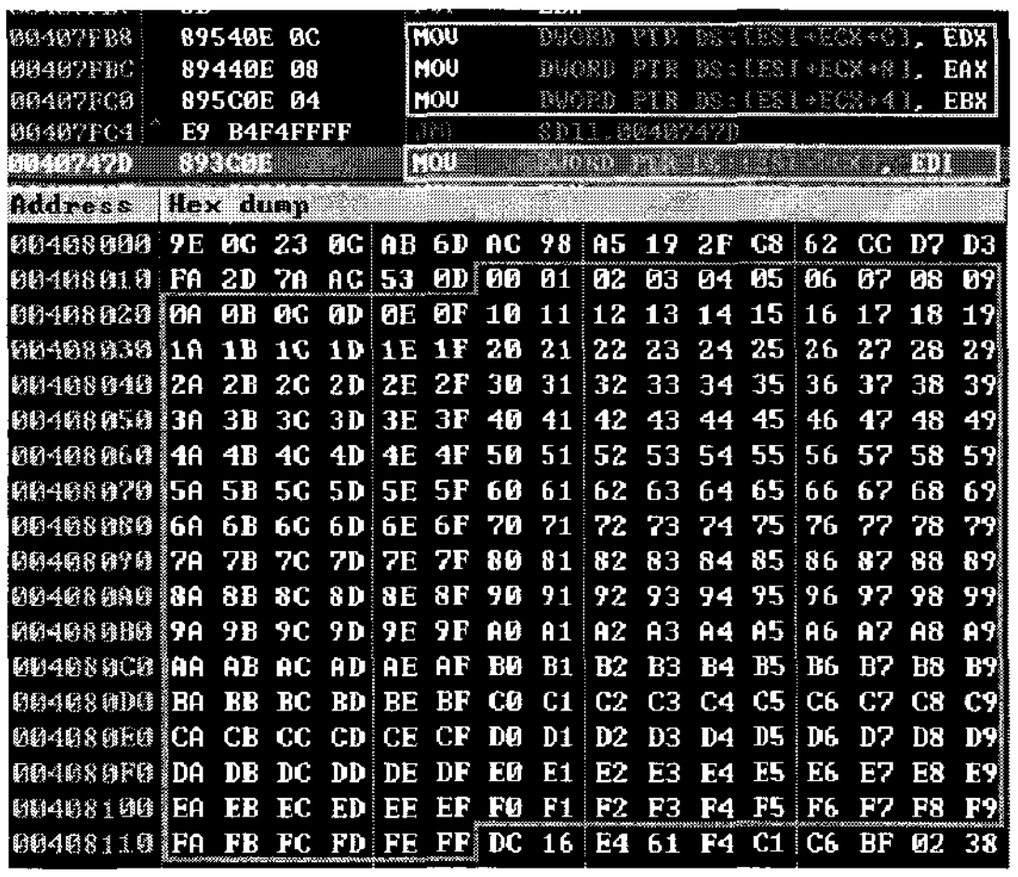
를 복호화하는 작업이며, 복호화를 통해 생성된 코드에서 수행하는 작업은 *.EXE 파일을 검색하여 감염시키는 바이러스 기능뿐 아니라 다양한 악의적인 기능들을 포함하고 있다.

Ⅲ. 유효코드 복호화

3.1 바이트 코드 치환 (1차 디코딩)

Win32/Kashu 바이러스 코드에 진입하면 먼저 다형성 코드로 암호화된 자신을 복호화 하는 과정으로부터 시작한다. 정상파일의 감염시점에 겹겹이 다중 인코딩 되었기 때문에 역으로 인코딩이 걸린 횟수 만큼 디코딩이 필요하다. 바이러스는 자신의 코드를 풀어내기 위해 먼저 디코딩 테이블을 생성하며 아래의 [그림 3]은 디코딩 테이블 생성을 위해 16진수 0xFF부터 0x00까지 역순으로 버퍼에 덮어쓰는 코드와 버퍼를 발췌한 정보이다.

완성된 디코딩 테이블 영역은 별도의 오프셋 테이블을 참조하여 바이트 단위 치환 작업이 진행된다. 치환 작업이 완료되면 인코딩 상태의 유효코드 들이 조합되며, 최종 디코딩 작업만을 남겨놓게 된다. 아래 [표 3]은



〔그림 3〕 디코딩 테이블 버퍼

바이트 단위의 치환 작업을 발췌한 코드이며 C언어 타입으로 변환한 코드는 [표 4]에 기술되어 있다. 코드를 살펴보면 순차적인 흐름을 타지 않고 많은 분기가 발생하는데, 이는 분석을 방해하고 백신 프로그램의 진단을 어렵게 하기 위한 의도로 볼 수 있다.

자신의 코드를 인코딩 하여 실제 유효코드를 은닉한 바이러스는 일반 바이러스에 비해 상대적으로 분석 및

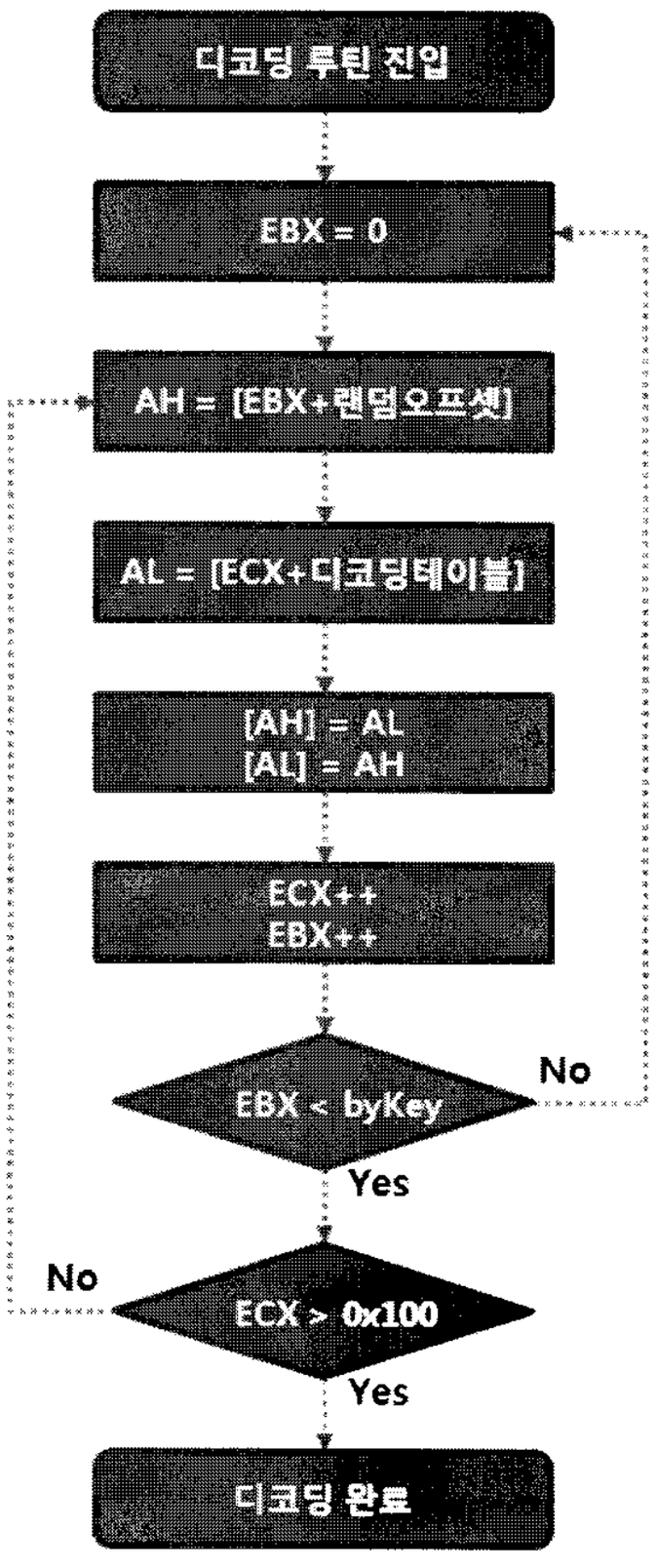
〔표 3〕 바이트 코드치환

```
00407E03 02141E              ADD  DL, [ESI+EBX]
00407E06 E9 D4FFFFFF         JMP  00407DDF

00407DDF 02D0                ADD  DL, AL
00407DE1 8AA415 16100000     MOV  AH, [EBP+EDX+1016]
00407DE8 E9 E1F2FFFF         JMP  004070CE

004070CE 43                  INC  EBX
004070CF 888415 16100000     MOV  [EBP+EDX+1016], AL
004070D6 E9 CD020000         JMP  004073A8

004073A8 88A40D 16100000     MOV  [EBP+ECX+1016], AH
004073AF 89C0                MOV  EAX, EAX
004073B1 3BDF                CMP  EBX, EDI
004073B3 0F82 C1FDFFFF       JB   0040717A
004073B9 E9 26020000         JMP  004075E4

0040717A FEC1                INC  CL
0040717C 0F85 C0080000       JNZ  00407A42
00407182 E9 960A0000         JMP  00407C1D

00407A42 8A840D 16100000     MOV  AL, [EBP+ECX+1016]
00407A49 E9 B5030000         JMP  00407E03
```

〔표 4〕 바이트 코드치환 (C Language)

```
bDL = 0;
for( i = 0; i < 0x100; i++)
{
        bAL = pBuf[i+0x16];
        bDL += pBuf[i%byKey];
        bDL &= 0xFF;
        bDL += bAL;
        bDL &= 0xFF;
        bAH = pBuf[bDL+0x16];
        pBuf[bDL+0x16] = (ahn_uint8)bAL;
        pBuf[i+0x16] = (ahn_uint8)bAH;
}
```

진단에 많은 시간이 소요 되는데, 이는 백신 프로그램에 진단정책이 반영되는 것을 최대한 지연시켜 자신의 생존시간을 늘려주는 효과를 가져오게 된다. [그림 4]는 바이트 코드 치환 작업을 순서도로 나타낸 것이다.



〔그림 4〕 바이트 코드치환 (순서도)

3.2 유효코드 복원 (2차 디코딩)

1차 디코딩이 완료된 이후 바이트 코드 치환 작업이 다시 한번 진행되며 계산된 오프셋에서 XOR Key를 획득하여 디코딩 작업이 이어진다. XOR 디코딩이 완료되면 최종 바이러스 유효코드와 함께 감염 시 백업된 정상파일 영역도 복원된다. 아래 [표 5]는 XOR 디코딩 코드를 발췌한 정보이며 C언어 타입으로 변환한 코드는 [표 6]에 기술되어 있다. 코드를 살펴보면 한 세트의 루프마다 0x00407D5A 주소에서 XOR Key를 얻어오는 오프셋이 재계산 되어 값이 매번 달라지는 것을 볼 수 있다.

최종 디코딩이 완료되어 유효코드가 노출되면 바이러스 제작자가 본래 의도한 목적코드(본체코드)로 진입하게 된다. 방어를 위해 겹겹이 쳐둔 보호 루틴이 모두 제거 되었으므로 보다 직관적인 코드를 볼 수 있게 된다. 복호화 이전의 코드인 [그림 3]의 디코딩 테이블 버퍼가 디코딩이 완료되어 유효코드가 노출되면 [그림 5]와 같아지며 [표 5]의 0x00407D6E 주소에서 RETN 분

〔표 5〕 XOR 디코딩 코드

```
0040789A FEC2                   INC     DL
0040789C 029C15 16100000        ADD     BL, [EBP+EDX+1016]
004078A3 89DB                   MOV     EBX, EBX
004078A5 E9 B5030000            JMP     00407C5F

00407C5F 8A8415 16100000        MOV     AL, [EBP+EDX+1016]
00407C66 E9 D5000000            JMP     00407D40

00407D40 8AAC1D 16100000        MOV     CH, [EBP+EBX+1016]
00407D47 88841D 16100000        MOV     [EBP+EBX+1016], AL
00407D4E 88AC15 16100000        MOV     [EBP+EDX+1016], CH
00407D55 87D2                   XCHG    EDX, EDX
00407D57 00E8                   ADD     AL, CH
00407D59 47                     INC     EDI
00407D5A 8A8405 16100000        MOV     AL, [EBP+EAX+1016]
00407D61 3007                   XOR     [EDI], AL
00407D63 FEC9                   DEC     CL
00407D65 4E                     DEC     ESI
00407D66 0F85 2EFBFFFF          JNZ     SDII.0040789A
00407D6C 58                     POP     EAX
00407D6D 58                     POP     EAX
00407D6E C3                     RETN    ; Go 본체코드
```

〔표 6〕 XOR 디코딩 코드 (C Language)

```
bDL = 0;
for( i = 0; i < DF_DOCODE_SIZE; i++ )
{
        bDL++;
        bDL &= 0xFF;
        bBL += pBuf[bDL+0x16];
        bBL &= 0xFF;
        bAL = pBuf[bDL+0x16];
        bCH = pBuf[bBL+0x16];
        pBuf[bBL+0x16] = bAL;
        pBuf[bDL+0x16] = bCH;
        bAL += bCH;
        bAL &= 0xFF;
        bAL = pBuf[bAL+0x16];
        pBuf[i+0x116] ^= bAL;
}
```

Address	Hex dump															
00408000	9E	0C	23	0C	AB	6D	AC	98	A5	19	2F	C8	62	CC	D7	D3
00408010	FA	2D	7A	AC	53	0D	E0	C3	6E	7C	BE	19	68	84	26	23
00408020	DD	83	8C	63	1E	5C	92	F2	34	D4	0D	8A	BB	7A	44	91
00408030	85	7F	0F	98	C1	79	78	E3	5A	A4	0E	53	C2	99	8B	0A
00408040	62	04	AC	B6	60	3E	35	30	C6	65	88	D7	DB	01	6C	87
00408050	37	59	73	B0	1B	18	CA	6F	47	0B	82	43	46	9F	61	66
00408060	2B	CC	E8	D9	75	2F	5B	9B	F4	1A	09	BD	4B	C4	29	41
00408070	B3	5E	F5	F6	E2	40	4C	00	14	F1	BA	6D	4E	F3	E7	81
00408080	4A	57	A9	3B	51	11	70	25	A1	B1	67	2C	D6	17	DF	10
00408090	A2	CE	BF	BC	50	4F	07	71	3F	48	80	97	EF	A3	03	A7
004080A0	D2	74	89	B4	54	4D	DE	FB	D3	EC	EA	F7	52	93	1F	0C
004080B0	06	AF	F9	F8	76	72	C9	3D	ED	7E	9E	D5	A8	FD	FA	6B
004080C0	05	2A	B5	2D	B2	5D	16	CD	E9	31	33	AB	13	39	6A	9D
004080D0	2E	9C	8D	E1	56	FC	9A	5F	58	A6	02	64	E6	94	38	8F
004080E0	AD	B9	7B	B8	96	1C	AE	55	D1	D8	8E	F0	3C	32	C8	AA
004080F0	A0	21	49	08	C0	36	CB	EB	A5	90	7D	DC	42	12	28	1D
00408100	FE	C5	FF	C7	DA	CF	77	E4	15	20	22	95	D0	45	3A	B7
00408110	27	86	69	EE	24	E5	E8	00	00	00	00	5D	81	ED	05	10

〔그림 5〕 디코딩이 완료된 유효코드

기하는 위치(Go 본체코드)는 [그림 5]의 0x00408116 주소가 된다.

Ⅳ. 바이러스 코드 진입

4.1 바이러스 파일의 구조

아래의 [그림 6]은 디코딩이 완료된 후 바이러스 파일의 구조를 나타낸 것이다. '본체코드'는 스레드 생성 및 해당 코드에서 필요로 하는 Windows API 함수들을 준비한다. 정상 PE 파일의 경우 Import Directory에 필요한 라이브러리 와 함수정보가 있지만, 태생적으로 Import Directory 사용에 제약을 받는 바이러스는 메모

- 스레드 생성
- 원본코드 실행 (정상파일 실행)
- 원본코드 복원
- PE 파일감염
- EP 지점의 원본코드

〔그림 6〕 바이러스 파일의 구조

리상의 kernel32.dll 주소를 찾고 Export Directory 리스트 중 LoadLibrary() , GetProcAddress() 포인터를 획득하는 것이 일반적인 방법이다. '스레드 코드'는 후위에 보관해 둔 원본코드를 복원하고 다른 PE 파일의 감염을 수행한다.

4.2 원본코드 복원 및 실행

감염 전 정상 프로그램의 엔트리 포인트에서 잘라냈던 원본코드의 복원이 완료되면 바이러스 코드는 복원된 원본의 엔트리 포인트 주소로 분기하고 자연스럽게 정상 프로그램이 실행되게 된다. [표 7]은 원본코드를 복원하는 스레드의 일부분을 발췌한 것이며 0x004085F6 주소에서 자신이 덮어쓴 원본코드의 크기를 확인하여 ECX 레지스터에 저장하고 0x004085FC 주소에서 원본코드의 백업 위치가 계산되어 ESI 레지스터에 저장된다. 코드 복원이 완료되면 0x0040860F 주소에서 상태 플래그가 설정되는 것을 볼 수 있다.

〔표 7〕 원본코드의 복원

```
004085F6 8B8D D0174000    MOV ECX,[EBP+4017D0]
004085FC 8DB5 D4174000    LEA ESI,[EBP+4017D4]
00408602 8BBD AE114000    MOV EDI,[EBP+4011AE]
00408608 F3:A4            REP MOVS [EDI],[ESI];복원
0040860A B8 01000000      MOV EAX,1 ; 상태 플래그
0040860F 8985 CC144000    MOV [EBP+4014CC],EAX
```

[표 8]은 스레드 동작을 감시하는 본체 코드를 발췌한 것이며 일정 시간 간격동안 SLEEP() 함수 루프를 돌면서 0x004082B9 주소의 복원성공 플래그 세팅을 확인하는 것을 볼 수 있다. 만일 스레드에 의해 복원성공 플래그가 세팅되면 0x004082C8 주소에서 엔트리 포인트 주소로 분기한다.

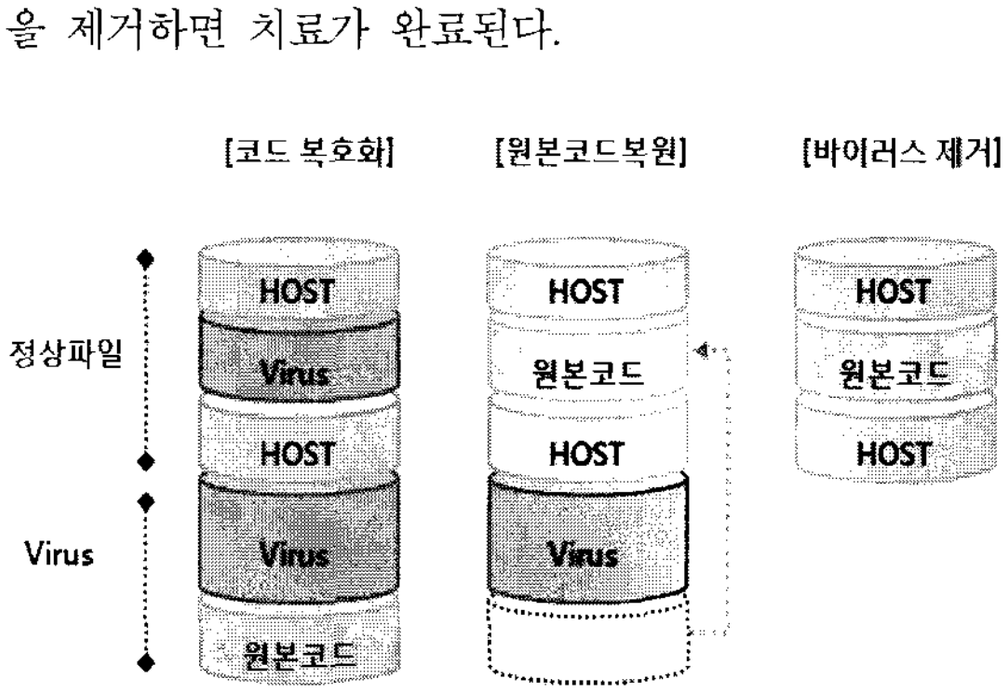
〔표 8〕 Entry Point 분기코드

```
0040829E 80BD CF174000 00    CMP   [EBP+4017CF], 0
004082A5 75 0A               JNZ   004082B1
004082A7 6A FF               PUSH  -1
004082A9 FF95 53144000       CALL  [SLEEP]
004082AF EB ED               JMP   0040829E
004082B1 6A 0C               PUSH  0C
004082B3 FF95 53144000       CALL  [SLEEP]
004082B9 83BD CC144000 01    CMP   [EBP+4014CC], 1
004082C0 75 DC               JNZ   0040829E
004082C2 61                  POPAD
004082C3 B8 D2224000         MOV   EAX, D2224000
004082C8 FFE0                JMP   EAX ; Entry Point
```

V. 바이러스 치료

5.1 EPO 바이러스의 치료

일반적인 후위형 바이러스의 치료는 정상 프로그램의 후위에 덧붙인 바이러스를 잘라내고 변경된 엔트리 포인트(Entry Point) 값을 원래의 값으로 수정해 주는 것으로 마무리 된다. 하지만 EPO 기법을 사용한 Win32/Kashu는 정상 프로그램 영역의 일부분이 바이러스 내부에 백업되어 있기 때문에 원본 코드 복원이 생략된 일반적인 치료법은 정상 프로그램을 손상시키는 결과를 가져온다. [그림 7]은 Win32/Kashu 치료과정을 나타낸 것으로 치료를 위해 먼저 백업된 정상 프로그램 영역이 노출될 때까지 인코딩 된 바이러스 코드의 복호화 과정이 진행되고 잘려진 정상 프로그램 영역을 원래의 위치에 복원해 주는 원본코드 복원 작업이 이어진다. 마지막으로 정상프로그램 후위에 삽입된 바이러스 영역을 제거하면 치료가 완료된다.



〔그림 7〕 Win32/Kashu 치료과정

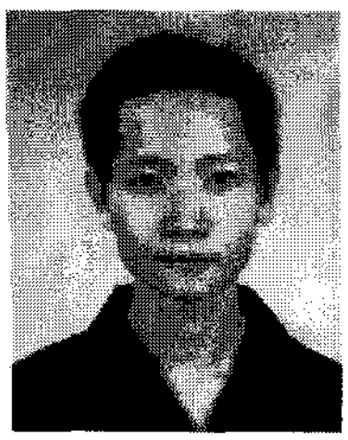
VI. 결 론

본 논문에서는 다형성 바이러스의 특징과 함께 다중 인코딩 기법이 적용된 사례를 설명하였다. 바이러스의 생존 시간은 발견을 기점으로 백신 프로그램에 진단정책이 반영되기 까지의 소요시간에 비례하므로 분석을 어렵게 하여 진단 되는 것을 지연 시키기 위해 매우 다양한 기법들이 사용된다. 이에 대응하는 백신 프로그램 또한 보다 효과적인 진단과 치료를 위해 진화를 거듭하고 있다. 하지만 '발견 후 진단'의 순환 고리를 갖는 바이러스의 특징으로 말미암아 항상 백신 프로그램은 그 뒤를 좇는 것이 현실이다. 보다 안전한 시스템 환경을 위해 최신 보안패치 적용과 같은 사용자의 적극적인 방역 의지가 우선 된다면 악성코드에 노출되는 확률은 최소화 될 수 있을 것이다.

참고문헌

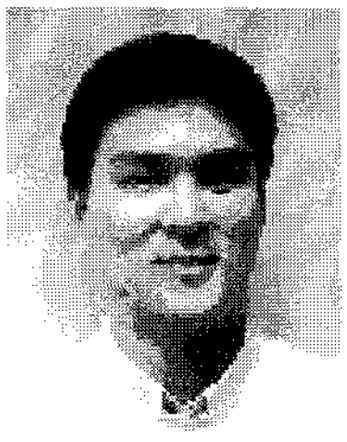
[1] Peter Szor, "The Art of Computer Virus Research and Defense" 2005
[2] David A. Solomon "Inside Microsoft Windows 2000, Third Edtion" 2000
[3] Kris Kaspersky "Hacker Dissembling Uncovered" 2003
[4] Kris Kaspersky "Hacker Debugging Uncovered" 2005
[5] Mihai Christodorescu, "Malware Detection"
[6] John Aycock, "Computer Viruses and Malware"
[7] www.ahnlab.com
[8] www.virusbtn.com/index
[9] www.microsoft.com/korea/technet/security

〈著者紹介〉



최 동 균 (Dong-Kyun Choi)
정회원

2001년 2월 : 대림대학 금속과 졸업
2002년 7월~현재 : 안철수연구소 ASEC팀 주임연구원, CISSP
<관심분야> 정보보안, 네트워크보안



양 하 영 (Yang Ha Young)
정회원

2003년 2월 : 충남대학교 컴퓨터과학과 졸업
2005년 2월 : 충남대학교 컴퓨터과학과 석사졸업
2005년 1월~현재 : 안철수연구소 ASEC팀 주임연구원
<관심분야> 바이러스, 보안 취약점