2018. 05. 04

Analysis Report

MS Office 수식 편집기 취약점 분석

CVE-2017-11882 / CVE-2018-0802

안랩 시큐리티대응센터(ASEC) 분석연구팀

AhnLab

경기도 성남시 분당구 판교역로 220 (우) 13493 | 대표전화 : 031-722-8000 | 팩스 : 031-722-8901 | www.ahnlab.com
© AhnLab, Inc. All rights reserved.

목차

AhnLab © AhnLab, Inc. All rights reserved.

개요

CVE-2017-11882, CVE-2018-0802은 마이크로소프트 오피스(Microsoft Office)에 포함된 수식 편집기(Equation Editor) 관련 취약점으로, 각각 2017년 11월과 2018년 1월에 공개됐다. 마이크로소프트(이하 MS)에서는 CVE-2017-11882 취약점 해결을 위한 보안 패치를 발표했으나, 결과적으로 CVE-2018-0802 취약점이 존재하게 되었다.

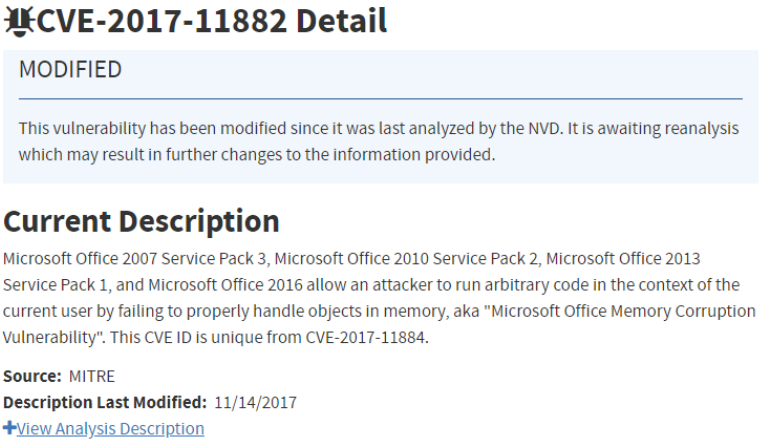
한편, CVE-2017-11882 취약점이 공개된 지난 2017년 11월 이후, 관련 샘플이 비록 소량이지만 지속적으로 나타나고 있다. 해외에서는 지난 2017년 연말 지능형 위협 공격(Advanced Persistent Threat, APT)에 해당 취약점이 이용된 사례가 확인된 바 있다. 국내에서는 기업을 대상으로 스팸 메일을 통한 관련 샘플 유포가 확인되었으며, 지난 2018년 3월에는 외형적으로 기존 샘플들과 다소의 차이를 보이는 변종도 확인된 바 있다.

본 보고서에서는 주요 샘플을 이용해 이들 취약점의 발생 원인과 함께 유사성 및 차이점에 대해 상세히 살펴본다.

CVE-2017-11882/CVE-2018-0802 요약

CVE-2017-11882, CVE-2018-0802 취약점은 마이크로소프트 오피스(Microsoft Office, 이하 MS오피스) 컴포넌트인 수식 편집기(Equation Editor, 이하 EQNEDT32.EXE)의 취약점으로, EQNEDT32.EXE가 메모리상에서 특정 오브젝트를 적절하게 처리하지 못하는 과정에서 발생한다. MS오피스 메모리 손상 취약점(Microsoft Office Memory Corruption Vulnerability)으로 알려져 있다[1][2].

 © AhnLab, Inc. All rights reserved.



CVE-2017-11882 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

Microsoft Office 2007 Service Pack 3, Microsoft Office 2010 Service Pack 2, Microsoft Office 2013 Service Pack 1, and Microsoft Office 2016 allow an attacker to run arbitrary code in the context of the current user by failing to properly handle objects in memory, aka "Microsoft Office Memory Corruption Vulnerability". This CVE ID is unique from CVE-2017-11884.

Source: MITRE
Description Last Modified: 11/14/2017
View Analysis Description

[그림 1] CVE-2017-11882 취약점 공지(*출처: National Vulnerability Database)

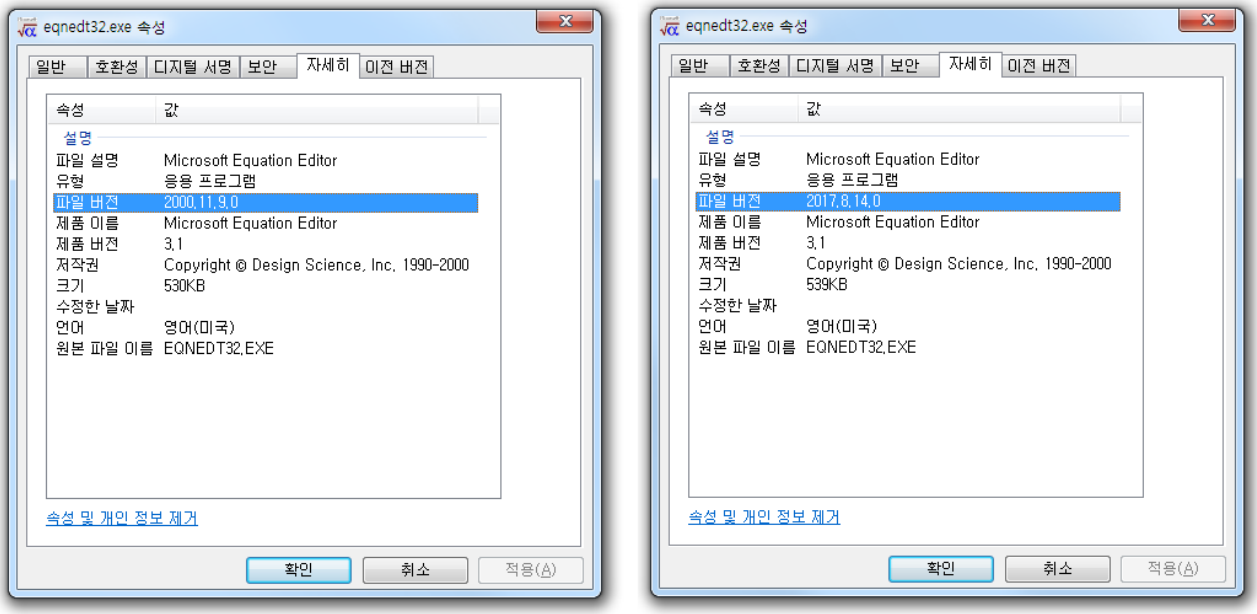
공격자가 해당 취약점을 악용할 경우, 현재 사용자의 관리자 권한으로 로그인해 임의의 코드를 실행할 수 있다. 이들 취약점에 영향 받는 제품군은 MS오피스 2007, 2010, 2013, 2016 등으로, 현재 시중에서 이용되고 있는 대다수의 MS오피스 제품군이 해당된다. 즉, 수많은 MS오피스 사용자들이 영향 받을 수 있으며, 이에 따라 이들 취약점의 심각도 또한 "높음(HIGH)"으로 분류되어 있다.

주목할 점은 [표 1]에서 볼 수 있는 것처럼 CVE-2017-11882 취약점에 대한 보안 업데이트가 발표된 지 10여일만에 CVE-2018-0802가 보고되었다는 사실이다. 이들 취약점은 동일한 프로그램에서 동일한 원리(스택 오버플로우)에 의해 발현되는데, 당초 배포된 보안 업데이트에서 잠재적으로 취약한 모든 부분에 대해 조치가 이루어지지 못했기 때문에 이러한 현상이 발생했다.

일시	주요 내용
2017년 8월 3일	CVE-2017-11882 취약점 보고 (보고자: Embedi [3])
2017년 11월 14일	CVE-2017-11882 취약점 공식 발표 및 MS 보안 업데이트 배포 [4] - 업데이트된 EQNEDT32.EXE 버전: 2017.08.14.0
2017년 11월 24일 경	CVE-2018-0802 취약점 보고(보고자: Qihoo 360, 0patch, Check Point Software Technologies 등 다수 분석가 [5])
2018년 1월 9일	CVE-2018-0802 취약점 공식 발표 및 MS 보안 업데이트 배포 [5] - 보안 업데이트 배포 후 EQNEDT32.EXE가 MS오피스 패키지에서 제거됨

[표 1] CVE-2017-11882/CVE-2018-0802 취약점 타임라인

 © AhnLab, Inc. All rights reserved.



[그림 1] CVE-2017-11882 취약 버전(왼쪽)과 CVE-2018-0802 취약 버전(오른쪽) EQNEDT32.EXE

이와 관련해 제로패치(0patch)는 보안 업데이트가 배포되면 업데이트된 내용에 대해 상세한 분석을 수행하는 경우가 많아지는 추세로, CVE-2018-0802 취약점 또한 이러한 과정에서 발견된 것이라 할 수 있다[6].

CVE-2017-11882/CVE-2018-0802 분석

1. 취약점 발생 원인

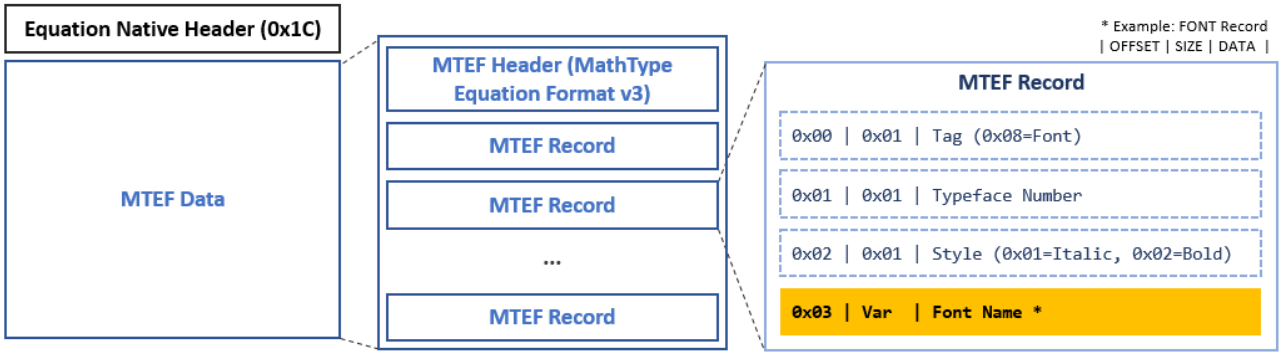
MS오피스의 수식 편집기를 이용하여 수식을 삽입하면 해당 수식은 임베디드(embedded) OLE 형태로 아래와 같은 스트림에 저장된다.

- ₩x01Ole
- ₩x01CompObj
- ₩x03ObjInfo
- Equation Native - POC 예제에서 사용됨
- ₩x01Ole10Native - 이후 변형 악성 샘플에서 발견됨

 © AhnLab, Inc. All rights reserved.

이러한 수식이 포함된 문서를 열 때 수식 OLE 객체를 처리하기 위해 EQNEDT32.EXE가 실행된다. 수식 처리 과정에서 내부적으로 IPersistStorage::Load() 메소드가 호출되며, 각 데이터 스트림의 구조에 따라 필요한 데이터를 차례로 파싱(parsing)한다.

Equation Native 스트림을 예로 들면, 먼저 Equation Native 헤더가 파싱되고 차례로 MTEF 데이터가 파싱된다. [그림 3]과 같이 MTEF 데이터 스트림은 내부에 다수의 레코드를 포함할 수 있다. 레코드의 시작 부분 데이터가 0x08이면 폰트(FONT) 레코드로서 처리되는데, 폰트 레코드 내부의 폰트명(Font Name) 데이터가 CVE-2017-11882 및 CVE-2018-0802 취약점과 관련 있는 부분이다.

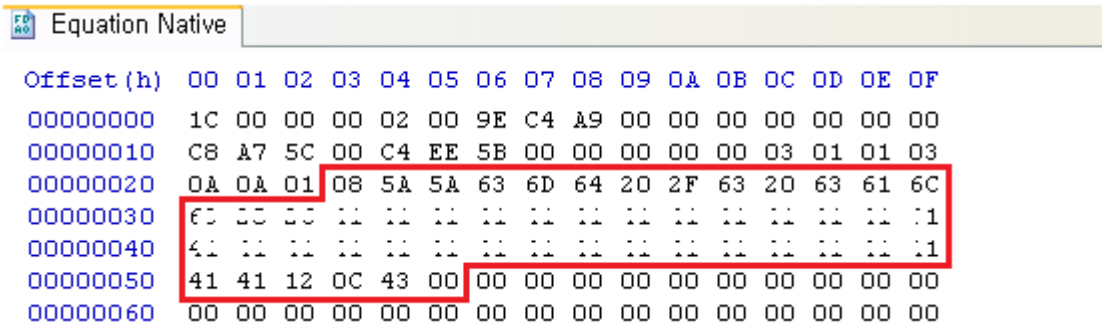


[그림 2] Equation Native 스트림 내부 구조

EQNEDT32.EXE에서 폰트 레코드의 폰트명을 처리하는 과정에서 strcpy()가 사용되며, 문자열 복사 대상 버퍼는 32바이트로 선언되어 있다. 그러나 복사 과정에서 원본•대상 버퍼에 대한 경계 체크가 이루어지지 않는다. 따라서 복사 원본 버퍼(폰트명)에 32 바이트를 초과하는 데이터가 존재하더라도 대상 버퍼로 그대로 복사되는 스택 오버플로우가 발생하게 된다.

예를 들어 [그림 4]의 예시와 같이 복사되는 버퍼의 내용(폰트명)을 적절히 조작하면 해당 함수의 리턴(Return) 주소를 덮어써 공격자가 원하는 지점으로 분기하도록 할 수 있다. [그림 4]의 예시에서는 리턴 주소를 0x430C12로 덮어썼으며, 이후부터 해당 주소의 코드를 실행하게 된다.

 © AhnLab, Inc. All rights reserved.


Equation Native

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	1C	00	00	00	02	00	9E	C4	A9	00	00	00	00	00	00	00
00000010	C8	A7	5C	00	C4	EE	5B	00	00	00	00	00	03	01	01	03
00000020	0A	0A	01	08	5A	5A	63	6D	64	20	2F	63	20	63	61	6C
00000030	63	20	20	20	20	20	20	20	20	20	20	20	20	20	20	31
00000040	41	20	20	20	20	20	20	20	20	20	20	20	20	20	20	31
00000050	41	41	12	0C	43	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

[그림 3] Equation Native 스트림 내부의 악성 데이터 예시

2. CVE-2017-11882와 CVE-2018-0802의 연관성 및 차이점

CVE-2017-11882 취약점이 처음 발견됐을 당시, 아래와 같은 EQNEDT32.EXE 취약한 부분 2가지가 확인되었다.

- SUB_41160F() 내부의 strcpy() - CVE-2017-11882 취약점 관련
- SUB_421774() 내부의 SUB_421E39() 에서의 strcpy() - CVE-2018-0802 취약점 관련

이들 모두 strcpy() 사용 시 버퍼 경계의 체크가 없어 스택 오버플로우 취약성이 존재하지만 코드의 흐름상 SUB_41160F()에서 먼저 취약점이 발생함에 따라 실질적으로 SUB_421E39()까지는 제어가 올 수 없는 구조였다.

지난 2017년 11월 배포된 MS 보안 업데이트를 통해 SUB_41160F() 함수에는 strcpy() 전•후 버퍼 크기를 확인하는 코드가 추가되었지만, SUB_421E39()의 strcpy()에 대해서는 조치가 취해지지 않았다. 더구나 업데이트 이후 기존에 문제가 되었던 SUB_41160F()를 자연스럽게 통과하여 SUB_421E39()까지 제어가 넘어올 수 있게 되면서 기존 취약점 유발 데이터를 조금만 조작하면 두 번째 취약점까지 악용하는 것이 가능해졌다. 이것이 CVE-2018-0802 취약점이다.

 © AhnLab, Inc. All rights reserved.

	2017.08.14.0 이전 버전	2017.08.14.0 버전

SUB_41160F()	LOWORD(v13) = -1;
	LOWORD(v8) = -1;
	v9 = strlen(a1);
	strcpy(&v12, a1);
	_strupr(&v12);
	v11 = sub_420FA0();
	LOWORD(v10) = 0;
	while (v11 > (signed __int16)v10)
	{
	if (sub_420FBB(v10, &v5))
	{
	strcpy(&v4, &v5);
	if (v6 == 1)
	_strupr(&v4);
	v7 = strstr(&v4, a1);
	if (v7 || (v7 = strstr(&v4, &v12)) != 0)
	{
	if (!a2 || !strstr(&v4, a2))
	{
	if ((signed __int16)strlen(&v5) == v9)
	{
	strcpy((char *)a3, &v5);
	return 1;
	}
	if (v7 == &v4)
	LOWORD(v8) = v10;
	else
	LOWORD(v13) = v10;
	}
	}
	}
	LOWORD(v10) = v10 + 1;
	}
	if ((v8 & 0x8000u) != 0)
	{
	if ((v13 & 0x8000u) != 0)
	{
	result = 0;
	}
	* 길이 체크 없는 strcpy()가 사용됨
		LOWORD(v17) = -1;
		LOWORD(v12) = -1;
		v13 = strlen(a1);
		v3 = strlen(a1) + 1;
		if (v3 >= 0x21)
		v3 = 32;
		qmemcpy(v16, a1, v3);
		v16[v3] = 0;
		_strupr(v16);
		v15 = sub_420FA0();
		LOWORD(v14) = 0;
		while (v15 > (signed __int16)v14)
		{
		if (sub_420FBB(v14, &v10))
		{
		v4 = strlen(&v10) + 1;
		if (v4 >= 0x21)
		v4 = 32;
		qmemcpy(v9, &v10, v4);
		v9[v4] = 0;
		_strupr(v9);
		v11 = strstr(v9, a1);
		if (v11 || (v11 = strstr(v9, v16)) != 0)
		{
		if (!a2 || !strstr(v9, a2))
		{
		if ((signed __int16)strlen(&v10) == v13)
		{
		v5 = strlen(&v10) + 1;
		if (v5 >= 0x21)
		v5 = 32;
		qmemcpy((void *)a3, &v10, v5);
		*(_BYTE *)(a3 + v5) = 0;
		return 1;
		}
		if (v11 == v9)
		LOWORD(v12) = v14;
		else
		LOWORD(v17) = v14;
		* strlen() 후 qmemcpy()가 사용됨

SUB_421E39()	LPARAM result; // eax
	
	strcpy((char *)(lParam + 28), lpLogfont);
	*(_BYTE *)(lParam + 23) = 1;
	EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
	*(_DWORD *)(lParam + 4) = 0;
	*(_DWORD *)(lParam + 8) = 0;
	*(_DWORD *)(lParam + 12) = 0;
	if (a2 & 1)
	*(_DWORD *)(lParam + 16) = 700;
	else
	*(_DWORD *)(lParam + 16) = 400;
	if (a2 & 2)
	*(_BYTE *)(lParam + 20) = 1;
	else
	*(_BYTE *)(lParam + 20) = 0;
	*(_BYTE *)(lParam + 21) = 0;
	*(_BYTE *)(lParam + 22) = 0;
	*(_BYTE *)(lParam + 24) = 0;
	*(_BYTE *)(lParam + 25) = 0;
	*(_BYTE *)(lParam + 26) = 0;
	result = lParam;
	*(_BYTE *)(lParam + 27) = 0;
	return result;
	* 길이 체크 없는 strcpy()가 사용됨
		LPARAM result; // eax
		
		strcpy((char *)(lParam + 28), lpLogfont);
		*(_BYTE *)(lParam + 23) = 1;
		EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
		*(_DWORD *)(lParam + 4) = 0;
		*(_DWORD *)(lParam + 8) = 0;
		*(_DWORD *)(lParam + 12) = 0;
		if (a2 & 1)
		*(_DWORD *)(lParam + 16) = 700;
		else
		*(_DWORD *)(lParam + 16) = 400;
		if (a2 & 2)
		*(_BYTE *)(lParam + 20) = 1;
		else
		*(_BYTE *)(lParam + 20) = 0;
		*(_BYTE *)(lParam + 21) = 0;
		*(_BYTE *)(lParam + 22) = 0;
		*(_BYTE *)(lParam + 24) = 0;
		*(_BYTE *)(lParam + 25) = 0;
		*(_BYTE *)(lParam + 26) = 0;
		result = lParam;
		*(_BYTE *)(lParam + 27) = 0;
		return result;
		* 길이 체크 없는 strcpy()가 그대로 존재함
		(CVE-2018-0802 취약점 원인)

[표 2] MS 보안 업데이트 전후의 EQNEDT32.EXE 비교

CVE-2017-11882와 CVE-2018-0802는 서로 연관된 취약점이지만 익스플로잇(exploit)과 관련해서는 다음과

 © AhnLab, Inc. All rights reserved.

같은 차이를 보인다.

- CVE-2017-11882 취약점 악용 익스플로잇
  - 2017년 11월 MS 보안 업데이트가 적용되기 이전의 EQNEDT32.EXE에서만 동작한다. (2017.08.14.0 이전 버전)
  - 해당 업데이트 이후에도 SUB_421E39()은 여전히 취약하기 때문에 해당 함수 내부에서 스택 오버플로우는 발생하지만, 실제 익스플로잇은 제대로 발현되지 않는다. 리턴 주소를 덮어쓰기 위해 복사해야하는 데이터의 크기가 달라져야 하기 때문이다.
- CVE-2018-0802 취약점 악용 익스플로잇
  - 2017년 11월 MS 보안 업데이트가 적용된 EQNEDT32.EXE에서만 동작한다. (2017.08.14.0 버전)
  - 해당 업데이트 이전에는 SUB_421E39()이전에 SUB_41160F()에서 먼저 스택 오버플로우가 발생하며, 이때 리턴 주소를 정확히 덮어쓰지 못하기 때문에 익스플로잇이 성공하지 못한다.

이들 두 취약점의 동작 환경을 정리하면 다음과 같다.

취약점 ID	2017.08.14.0 이전 버전	2017.08.14.0 버전
CVE-2017-11882	익스플로잇 성공	익스플로잇 실패
CVE-2018-0802	익스플로잇 실패	익스플로잇 성공

[표 3] EQNEDT32.EXE 버전별 취약점 동작 환경

AhnLab © AhnLab, Inc. All rights reserved.

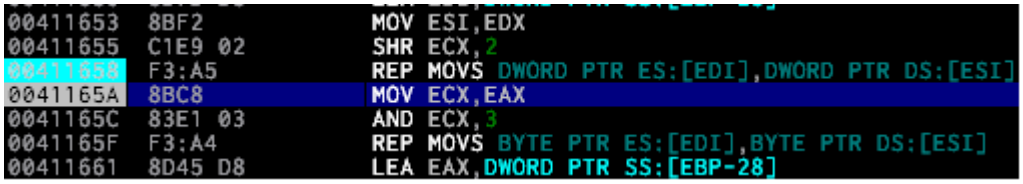
CVE-2017-11882/CVE-2018-0802 익스플로잇 상세 분석

CVE-2017-11882와 CVE-2018-0802는 연관된 취약점임에도 불구하고 익스플로잇 발현 환경과 과정에서 차이가 존재한다. 2017년 11월 MS 보안 업데이트 적용 전후의 EQNEDT32.EXE 버전에 따른 이들 두 취약점의 익스플로잇 과정을 상세히 살펴본다.

1. CVE-2017-11882

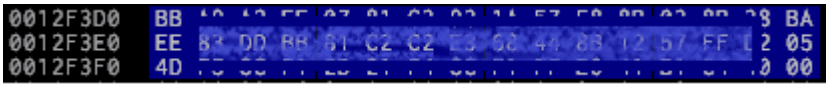
1.1) EQNEDT32.EXE 버전 2000.11.09.0 환경(보안 업데이트 적용 전)

EQNEDT32.EXE의 SUB_41160F() 내의 취약한 strcpy()는 [그림 5]와 같다.



[그림 4] EQNEDT32.EXE 내부의 취약한 strcpy()

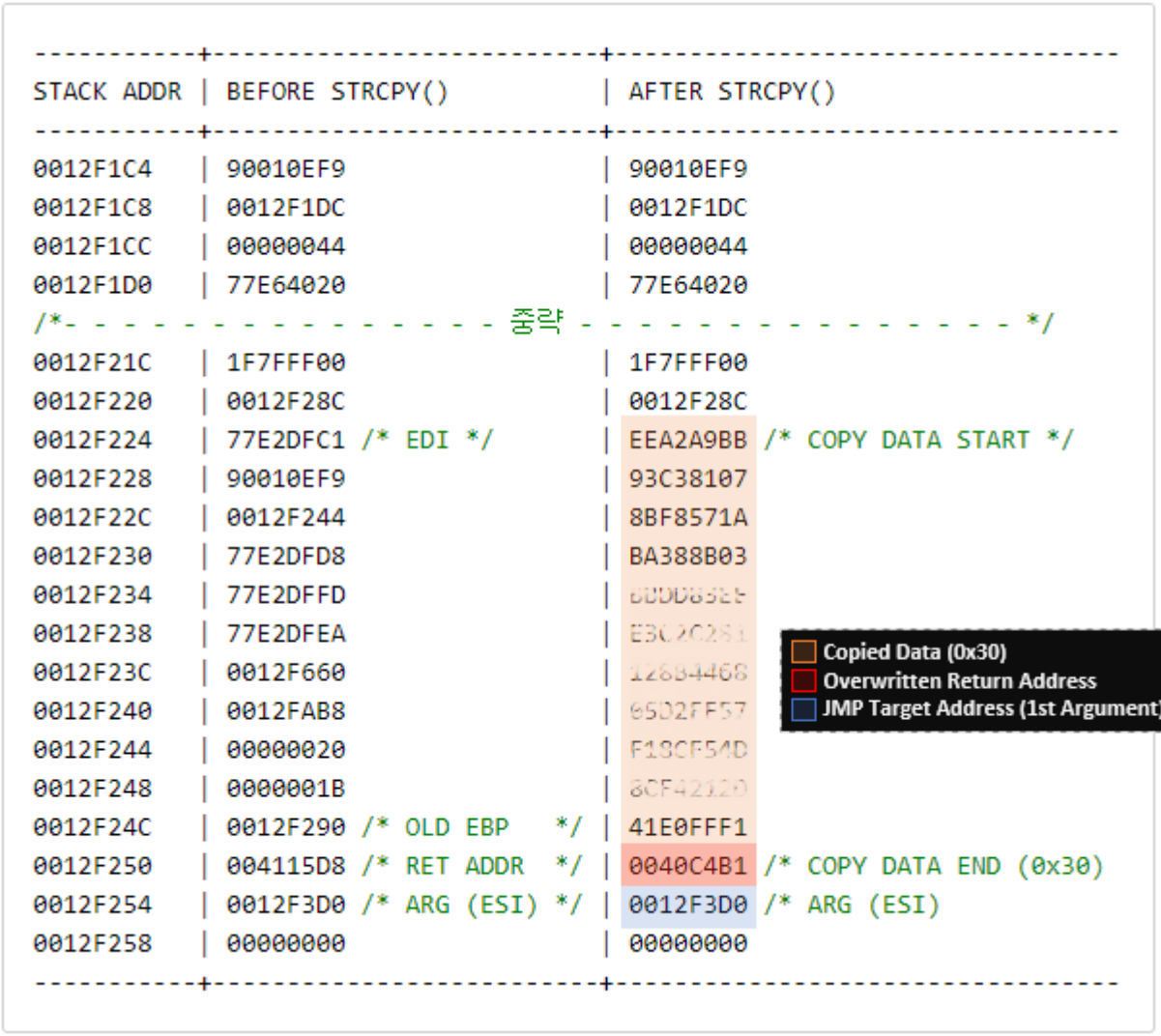
MS오피스로 악성 문서를 불러오면 우선 정상적으로 폰트가 몇 차례 파싱되며, 이후 악성 폰트 레코드 데이터가 복사된다. 복사되는 데이터는 총 0x30이며 마지막은 null(0x00)로 끝난다. 해당 데이터는 이후 쉘코드(shellcode)서 동작한다.



[그림 5] CVE-2017-11882 취약점을 악용하는 폰트명 데이터

strcpy() 호출 전후의 스택의 모습은 [그림 7]과 같다.

 © AhnLab, Inc. All rights reserved.



[그림 6] CVE-2017-11882 취약점 익스플로잇 성공 시 스택

원래는 0x20 크기의 데이터가 복사 되어야 하지만, 경계 체크 루틴이 존재하지 않아 [그림 7]과 같이 공격자가 의도한 대로 0x30 크기의 데이터가 모두 복사 되었으며, SUB_41160F() 호출 후 복귀할 리턴 주소가 덮어써진 것을 확인할 수 있다(@12F250 = 0x4115D8 → 0x40C4B1).

또 하나 주목할 부분은 리턴 주소 바로 다음의 값이다. 이는 SUB_41160F() 호출 당시 첫 번째 인자로 전달된 값이며, strcpy()에서 원본 버퍼의 주소(ESI)에 해당한다. 즉, 복사 대상이었던 0x30 크기의 쉘코드가 존재하는 주소다(익스플로잇 성공 시 분기할 주소, @12F254 = 0x12F3D0)

 © AhnLab, Inc. All rights reserved.

SUB_41160F() 함수 종료 후 [그림 8]과 같이 0x40C4B1 주소로 리턴하는 것을 확인할 수 있다.

[그림 7] 쉘코드 분기를 위해 사용되는 RETN 명령어

[그림 8]의 메모리 상에 보이는 @40C4B1은 EQNEDT32.EXE 내부 SUB_40C46C() 함수의 종료 부분의 RETN 명령을 가리키고 있다. 실제로는 EIP에 직접 값을 대입할 수 없지만 이 RETN 명령으로 인해 이론적으로 POP EIP와 같은 효과가 나타난다. 현재 ESP 위치(@12F254)에 저장된 값(0x12F3D0)으로 EIP 값이 대체된다.

RETN 실행 후 [그림 9]와 같이 EIP가 0x12F3D0로 설정되며 코드 분기가 이동한 것을 확인할 수 있다. 이 주소 영역은 앞서 설명한 바와 같이 strcpy() 복사 원본이었던 0x30 크기의 쉘코드 시작 부분에 해당한다.

[그림 8] CVE-2017-11882 익스플로잇 성공 후 쉘코드

이후 Equation Native 스트림 내부에 포함된 추가 쉘코드로 분기가 이루어지며, 이에 따른 악성 행위가 수행된다.

1.2) EQNEDT32.EXE 버전 2017.08.14.0 환경(보안 업데이트 적용 후)

2017년 11월 보안 업데이트가 적용된 EQNEDT32.EXE에서는 SUB_41160F()에 버퍼 경계 체크가 추가되었기 때문에 CVE-2017-11882 취약점이 동작하지는 않는다. 그러나 앞서 언급한 바와 같이 여전히 SUB_421774() 내부의 SUB_421E39()에는 [그림 10]과 같이 취약한 strcpy() 가 그대로 남아있는 상태다.

 © AhnLab, Inc. All rights reserved.

```
00421E56  83C7 1C         ADD EDI,1C
00421E59  8BF2            MOV ESI,EDX
00421E5B  C1E9 02         SHR ECX,2
00421E5E  F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00421E60  8BC8            MOV ECX,EAX
00421E62  83E1 03         AND ECX,3
00421E65  F3:A4           REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00421E67  8B45 10         MOV EAX,DWORD PTR SS:[EBP+10]
00421E6A  C640 17 01      MOV BYTE PTR DS:[EAX+17],1
00421E6E  8B45 10         MOV EAX,DWORD PTR SS:[EBP+10]
```

[그림 9] 보안 업데이트에서 조치되지 않은 strcpy() 취약 구간

이 환경에서 앞서 살펴본 것과 동일한 예제 샘플을 실행하면 SUB_421E39() 내에서 복사 대상의 버퍼 크기인 0x20를 초과한 총 0x30의 데이터가 복사되지만, 쉘코드로의 분기는 성공하지 못한다. 그 이유는 복사하는 데이터가 리턴 주소를 덮어쓸 만큼 충분하지 않았기 때문이다.

코드를 좀 더 상세히 살펴보면, SUB_421774()는 4개의 인자를 필요로 하는데, 이는 [그림 11]의 lpLogfont, a2, a3, a4 에 해당한다.

```
int __cdecl VUL2_WRAP_sub_421774(LPCSTR lpLogfont, __int16 a2, int a3, int a4)
{
  char v5; // [esp+Ch] [ebp-D8h]
  __int16 v6; // [esp+2Dh] [ebp-B7h]
  int v7; // [esp+2Fh] [ebp-B5h]
  HGDIOBJ h; // [esp+34h] [ebp-B0h]
  LOGFONTA lf; // [esp+38h] [ebp-ACh]
  HGDIOBJ ho; // [esp+74h] [ebp-70h]
  __int16 v11[2]; // [esp+78h] [ebp-6Ch]
  CHAR Name; // [esp+7Ch] [ebp-68h]
  struct tagTEXTMETRICA tm; // [esp+A0h] [ebp-44h]
  __int16 v14[2]; // [esp+D8h] [ebp-Ch]
  __int16 v15; // [esp+DCh] [ebp-8h]
  int v16; // [esp+E0h] [ebp-4h]
```

[그림 10] CVE-2018-0802 취약점 관련 EQNEDT32.EXE 내부 함수(1)

하위의 SUB_421E39()는 3개의 인자를 받아 동작하며, [그림 12]에서 lpLogfont, a2와 LOGFONTA 형의 &lf를 확인할 수 있다.

 © AhnLab, Inc. All rights reserved.

```
v16 = 0;
*(_WORD *)a4 = 0;
v15 = sub_421C4B((char *)lpLogfont, a2);
if ( v15 )
{
  *(_WORD *)a4 = v15;
  v16 = 1;
}
else
{
  sub_420E87();
  VUL_2_sub_421E39(lpLogfont, a2, (LPARAM)&lf);
  lf.lfHeight = -(signed __int16)(24 * word_45BAFA / 72);
```

[그림 11] CVE-2018-0802 취약점 관련 EQNEDT32.EXE 내부 함수(2)

위에서 확인한 바와 같이 이들 두 함수는 lpLogfont와 a2 등 2개의 인자를 공유한다. SUB_421E39() 내부에서 strcpy() 호출 시점에 복사 원본은 lpLogfont가, 복사 대상은 &lf(+0x1C 지점)가 된다.

```
LPARAM __cdecl VUL_2_sub_421E39(LPCSTR lpLogfont, __int16 a2, LPARAM lParam)
{
  LPARAM result; // eax

  strcpy((char *)(lParam + 0x1C), lpLogfont);
  *(_BYTE *)(lParam + 0x17) = 1;
  EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
  *(_DWORD *)(lParam + 4) = 0;
  *(_DWORD *)(lParam + 8) = 0;
  *(_DWORD *)(lParam + 12) = 0;
  if ( a2 & 1 )
    *( DWORD *)(lParam + 16) = 700;
```

[그림 12] CVE-2018-0802 취약점 관련 EQNEDT32.EXE 내부 함수(3)

[그림 14]에서 확인할 수 있는 것처럼 lpLogfont의 데이터는 앞서 확인한 0x30 크기의 쉘코드와 일치한다.

```
0012F3D0  BB A9 A2 EE 07 81 C3 93 1A 57 F8 8B 03 8B 38 BA
0012F3E0  EE 03 DD BB 81 C2 C2 E3 E8 41 83 12 FF FF D2 05
0012F3F0  4D F5 8C F1 2D 21 F4 8C F1 FF E0 41 B1 C4 40 00
```

[그림 13] strcpy() 호출 시 복사 원본 데이터(lpLogfont)

SUB_421E39() 함수 진입 및 strcpy() 호출 전후의 스택 모습은 [그림 15]와 같다.

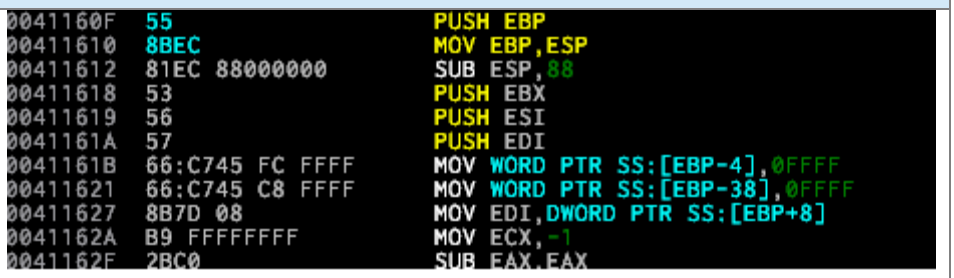
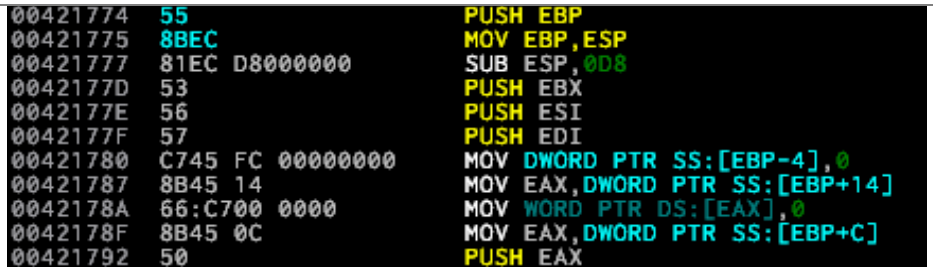
 © AhnLab, Inc. All rights reserved.

[그림 14] MS 보안 업데이트 후 CVE-2017-11882 취약점 익스플로잇 실패 스택

[그림 15]의 스택 데이터에서 확인할 수 있는 것처럼 strcpy()에 의해 @12F2F0부터 @12F320까지 총 0x30의 데이터가 성공적으로 복사되었다. 그러나 이 데이터는 리턴 주소를 덮어쓰기에는 부족하기 때문에 익스플로잇은 실패했으며, 이후 나머지 코드가 정상적으로 실행된다.

[그림 16]은 두 취약점 관련 함수의 지역 변수 사용 시 스택 영역을 비교한 것이다. SUB_421774() 함수에 진입 후에는 SUB ESP, 0D8 명령을 사용하여 총 0xD8 만큼의 지역 변수를 위한 스택 영역을 확보하는 반면, SUB_41160F() 함수의 경우에는 0x88 만큼의 스택 영역을 확보함을 확인할 수 있다.

 © AhnLab, Inc. All rights reserved.

함수	스택 영역
SUB_41160F()	 로컬 변수 영역 확보 = 0x88
SUB_421774()	 로컬 변수 영역 확보 = 0xD8

[그림 16] 두 취약점 관련 함수의 지역 변수 사용 공간 차이

SUB_41160F() 함수의 경우, strcpy()에 의해 데이터가 복사되는 대상 주소(EDI)는 @12F224이며, 리턴 주소가 저장된 위치는 @12F250이다. 주소 크기가 4바이트인 점을 고려하면 실제로는 0x30 만큼, 즉 @12F254까지 전부 덮어써야 한다. 샘플의 쉘코드가 정확히 0x30 바이트였기 때문에 해당 쉘코드는 위의 [그림 7]에서 확인한 바와 같이 리턴 주소를 덮어쓰는 것이 가능했던 것이다.

또한 [[그림 14]에서 확인할 수 있는 것처럼 SUB_421774() 함수에서 SUB_421E39()를 호출하여 strcpy()로 데이터를 복사할 때 대상 주소(EDI, &lParam + 0x1C 위치)는 @12F2F0이며, 리턴 주소가 저장된 위치는 @12F384이다. 앞 서와 같은 이유로 4바이트를 더하면 리턴 주소를 정확하게 덮어쓰기 위해서는 0x98만큼의 데이터가 필요하다는 결론에 도달한다.

이를 토대로 [그림 17]과 같이 0x98만큼의 데이터가 복사되도록 테스트를 진행했다. 0x94 크기의 NOP에 리턴 주소를 덮어쓰기 위한 4바이트를 추가하여 데이터를 구성하였으며, 덮어쓰는 리턴 주소는 EQNEDT32.EXE 내부의 'RETN' 명령어가 있는 임의의 주소(@421773)를 사용하였다.

테스트 결과, 예상한 바와 같이 리턴 주소를 덮어쓰는 것에 성공했다. 의도한 위치의 RETN 명령어를 실행하여 스택에 저장된 복사 원본 주소(ESI)로 분기가 이동하는 것을 확인할 수 있었다. 여기에서는 NOP 명령

 © AhnLab, Inc. All rights reserved.

으로 해당 영역을 채웠지만 쉘코드를 적절하게 구성할 경우 추가 악성 행위가 얼마든지 가능하다는 것을 알 수 있다.

[그림 15] EQNEDT32.EXE 보안 업데이트 후에도 리턴 주소를 덮어 쓰도록 조작한 예시

2. CVE-2018-0802

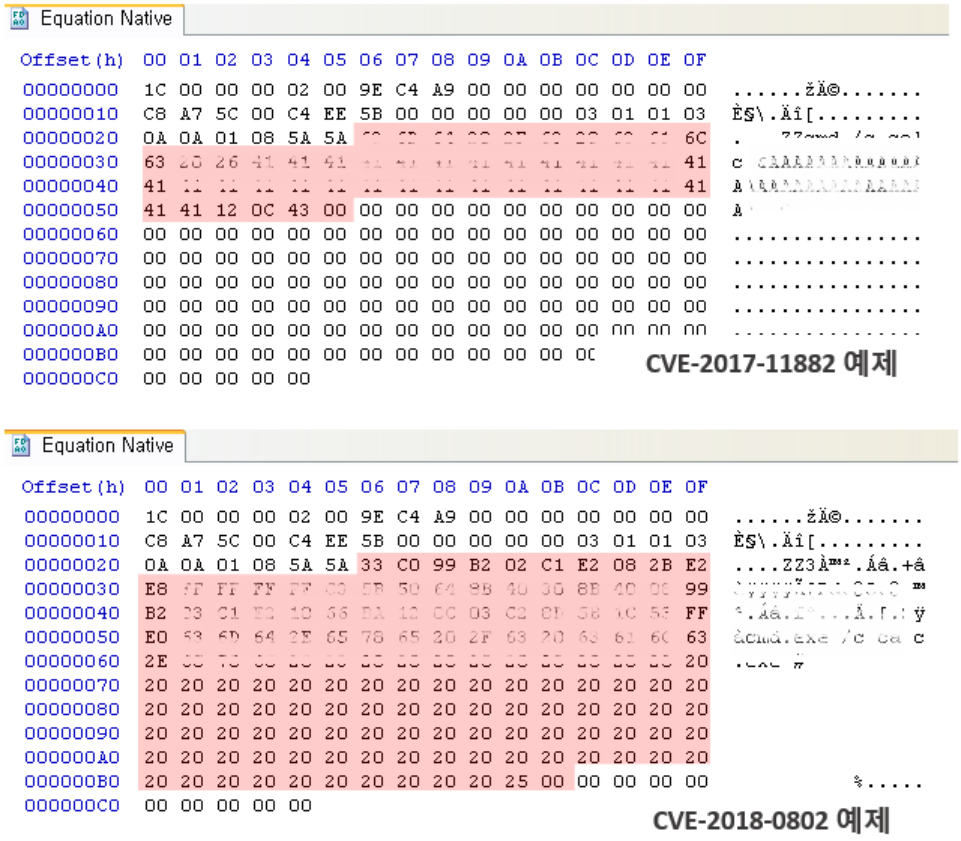
2.1) EQNEDT32.EXE 버전 2000.11.09.0 환경(보안 업데이트 적용 전)

CVE-2018-0802 취약점과 관련해 공개된 POC 샘플을 사용하여 분석한 내용이다. 해당 샘플 또한 Equation Native 객체를 포함하고 있다.

[그림 18]은 CVE-2017-11882 샘플과 CVE-2018-0802 샘플의 익스플로잇을 비교한 것이다. [그림 18]에 표시된 부분은 총 0x96* 바이트의 폰트명으로 복사되는 데이터이며, 익스플로잇에 사용되는 쉘코드를 포함한다.

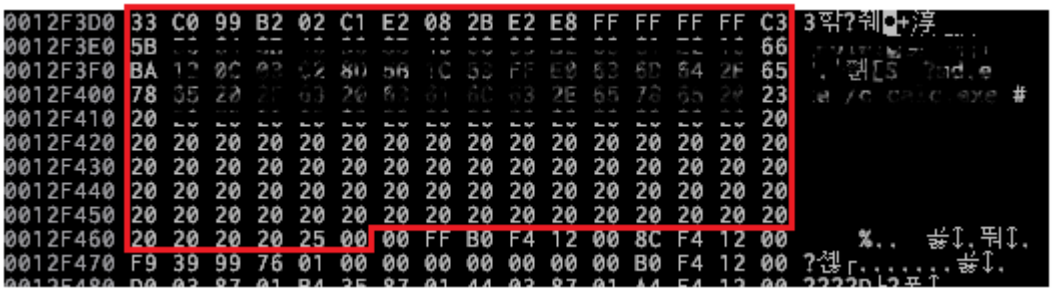
*(*직전에 NOP로 테스트한 데이터의 경우 0x98이었다. 이 차이에 대해서는 아래에서 추가로 설명한다.)*

 © AhnLab, Inc. All rights reserved.



[그림 18] CVE-2017-11882와 CVE-2018-0802 익스플로잇 비교

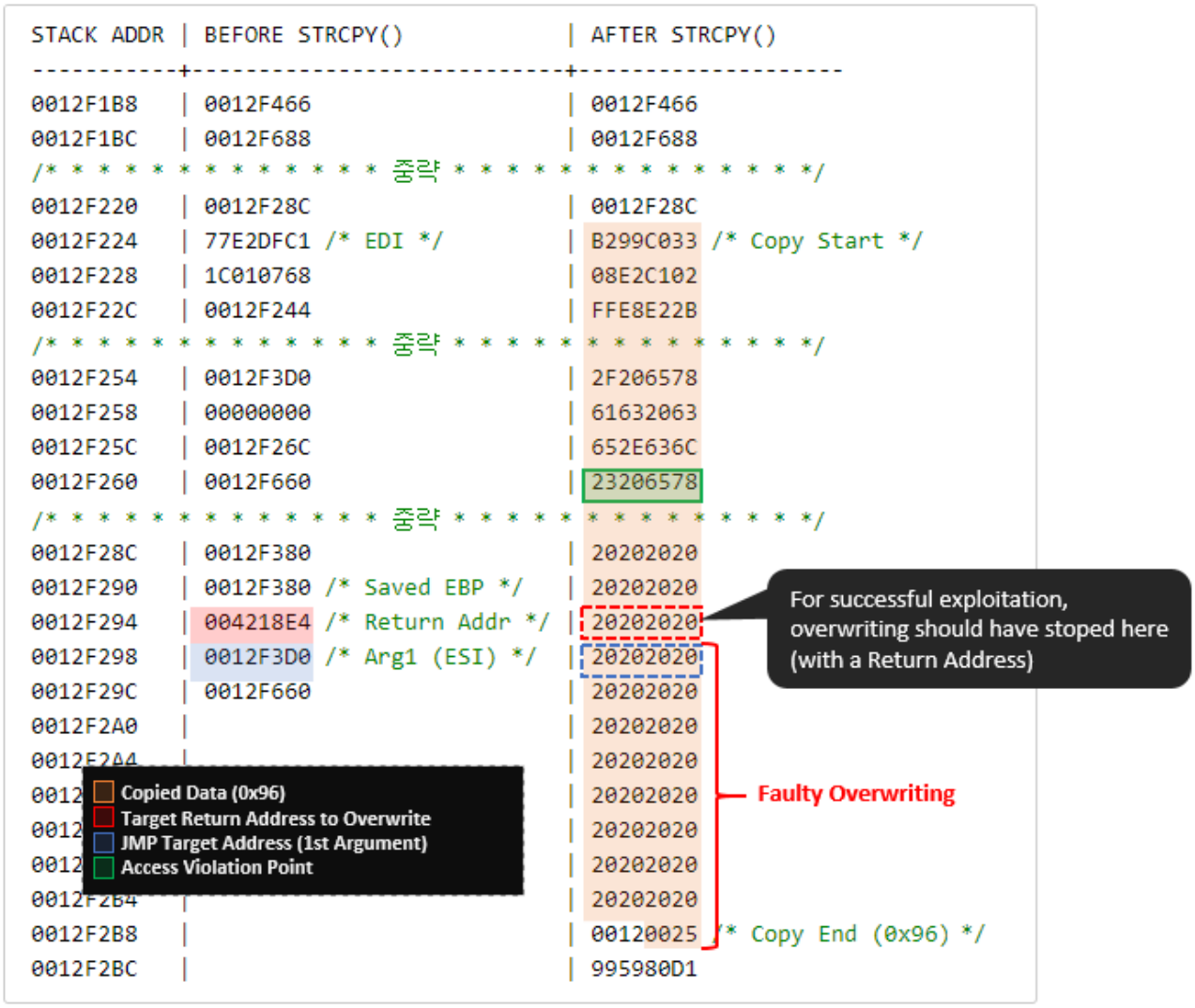
[그림 19]는 SUB_41160F()에서 취약한 strcpy() 함수를 사용하여 복사되기 직전의 모습이다. 현재 ESI (@12F3D0)가 가리키는 곳의 데이터가 [그림 18]의 표시된 0x26부터의 폰트명 부분 데이터에 해당하는 것을 확인할 수 있다.



[그림 16] CVE-2018-0802 취약점을 악용하는 폰트 데이터

strcpy() 함수가 실행되기 전후의 스택 모습은 [그림 20]과 같다.

 © AhnLab, Inc. All rights reserved.



[그림 20]17 CVE-2018-0802 익스플로잇 실패 시의 스택

SUB_41160F() 함수의 리턴 주소를 덮어쓰고자 한다면 @12F294까지만 데이터가 복사 되어야 했다. 그러나 [그림 20]의 스택 데이터에서 볼 수 있는 것처럼 만일 실제로는 @12F2B8까지 데이터가 복사되었으며, 이는 결과적으로 나중에 사용하기 위해 스택에 저장해 두었던 데이터 손상을 초래한다.

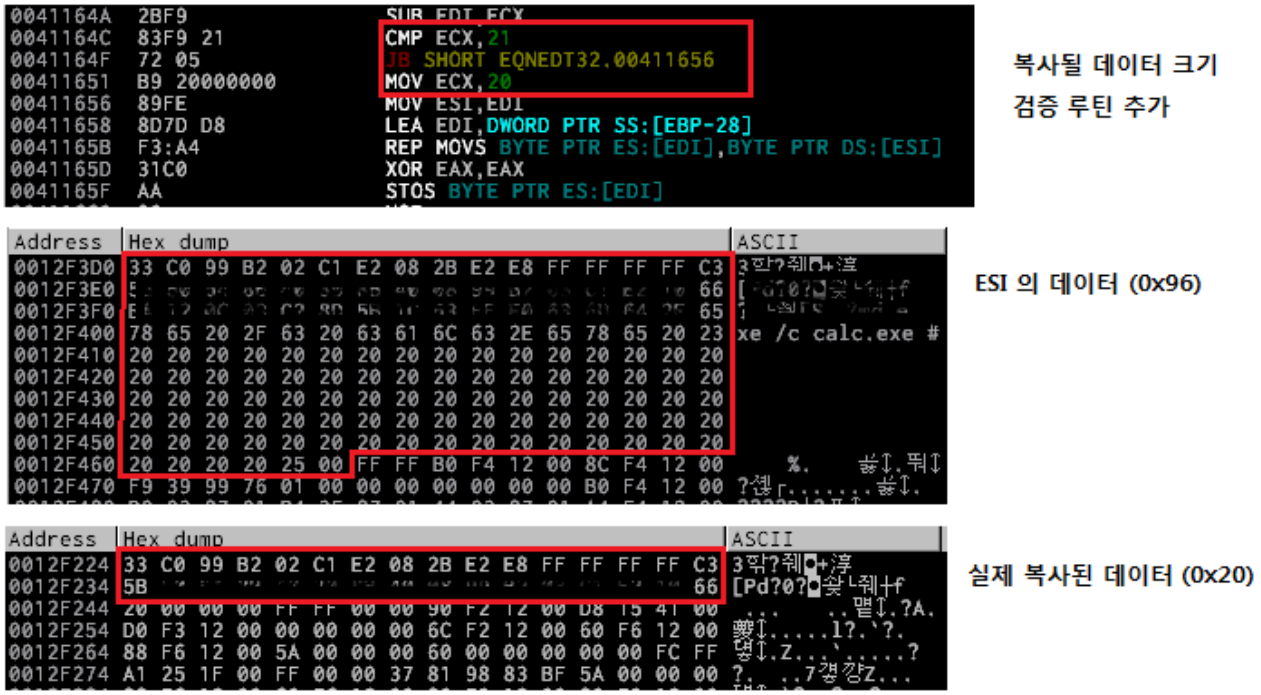
만일 SUB_41160F() 함수가 정상적으로 종료된다면 리턴 주소가 @20202020으로 변경되므로, 해당 주소 상의 코드 실행을 시도하다가 예외가 발생했을 것이다. 물론, 미리 해당 주소에 메모리 공간이 할당되어 있었고, 실행 가능 속성이 있지 않았을 경우에 한한다. 그러나 실제로는 SUB_41160F() 함수가 종료되기 전부터 스택 손상으로 인한 영향이 나타난다. 문제의 strcpy() 루틴 직후에 나타나는 strstr() 함수에서 참조하는 데이터가 올바르지 않은 주소를 참조하게 되면서 접근 위반(Access Violation) 오류가 발생하며, 따라서 해당 익스플로잇은 실패로 끝나게 된다.

 © AhnLab, Inc. All rights reserved.

2.2) EQNEDT32.EXE 버전 2017.08.14.0 환경(보안 업데이트 적용 후)

마찬가지로 위와 동일한 CVE-2018-0802 POC 샘플을 2017년 11월 보안 업데이트가 적용된 EQNEDT32.EXE 환경에서 사용하여 분석하였다.

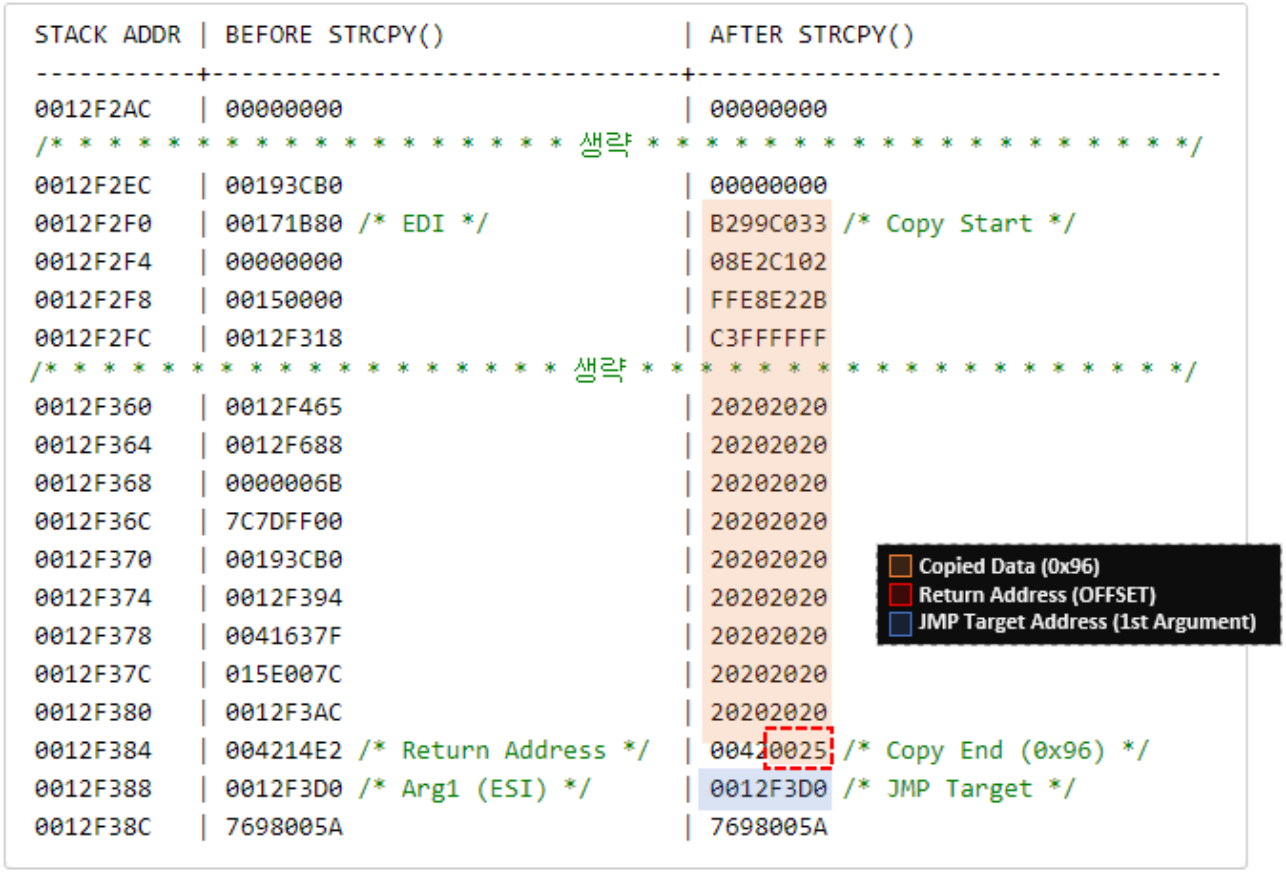
[그림 21]과 같이 보안 업데이트를 통해 취약했던 SUB_41160F() 내부에 데이터 크기를 검증하는 코드가 추가됨에 따라 원본 크기에 상관 없이 최대 0x20 크기만큼만 데이터가 복사된다. 즉, 계산된 문자열 길이가 0x21 이상인 경우, 0x20으로 길이를 강제 치환하기 때문에 더 이상 스택 오버플로우가 발생하지 않는다.



[그림 181] 보안 업데이트로 인해 수정된 데이터 복사 구간

그러나 해당 업데이트에서 수정되지 않은 SUB_421774() 함수에서는 내부의 SUB_421E39() 함수에서 여전히 strcpy()에 의한 스택 오버플로우가 발생하며, SUB_421774() 함수 종료 후 리턴 주소가 [그림 22]와 같이 변경된다.

 © AhnLab, Inc. All rights reserved.



[그림 192] 보안 업데이트 적용 환경에서 CVE-2018-0802 익스플로잇 성공 시 스택

[그림 22]의 스택에서 주목할 점은 리턴 주소가 덮어써진 방식이다. 앞서 확인한 바와 같이 NOP를 활용한 테스트 당시에는 0x98 크기의 데이터를 사용했지만 POC 샘플에서는 0x96 만큼의 데이터를 사용했다. 이 차이는 0x94의 데이터 이후에 추가되는 데이터 크기에서 기인한다.

- 4바이트를 추가한 경우: 리턴 주소 덮어쓰기에 전체 주소 복사
  - 0x94 바이트 + 00421773
- 2바이트를 추가한 경우: 리턴 주소 덮어쓰기에 상대 주소 복사
  - 0x94 바이트 + 0025
  - 4바이트의 원래 주소 004214E2에서 상위 2바이트를 남기고 하위 2바이트만 0025로 치환되어 00420025로 변경됨

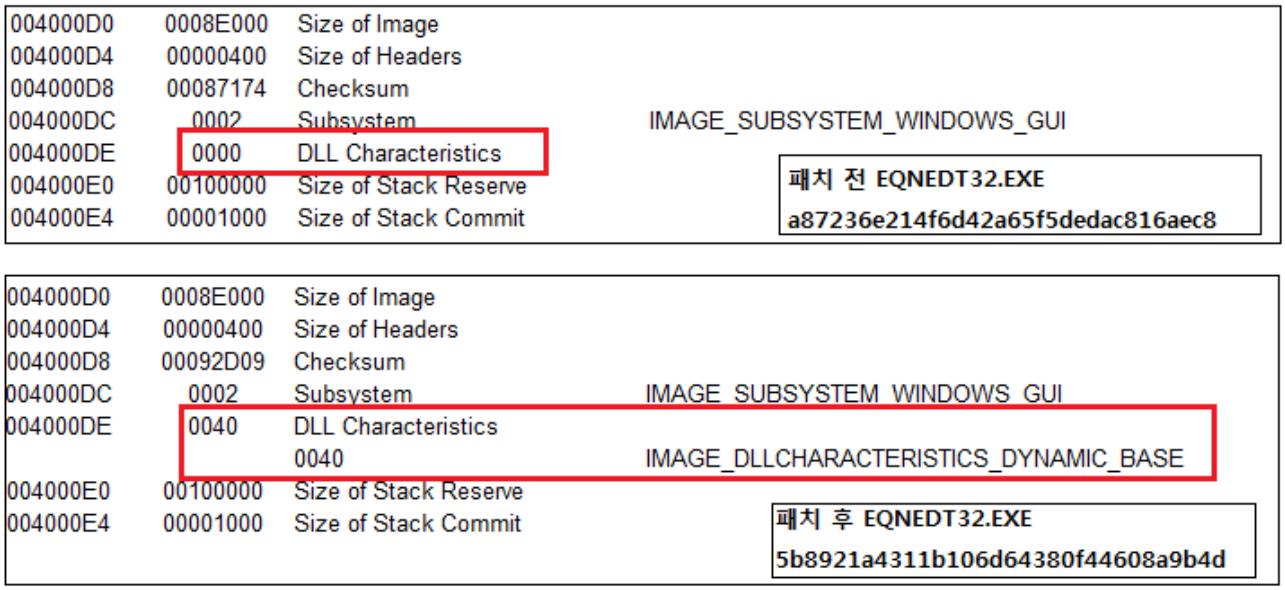
메모리 상의 @420025 위치에는 EQNEDT32.EXE 내부의 특정 함수의 종료 부분의 RETN 명령이 존재한다. 해당 RETN 명령이 실행되면 POP EIP와 동일한 효과가 발생한다. 즉, CVE-2017-11882와 동일하게 리턴 주소 바로 직전(@12F388)에 저장된 값(= SUB_421774() 호출 시 첫 번째 인자로 전달된 값(= lpLogfont = ESI = 쉘코드 시작점)으로 EIP를 변경하기 위한 방법이다.

 © AhnLab, Inc. All rights reserved.

[그림 203] 상대 주소로서 참조된 RETN 명령어

이전까지는 RETN 명령어의 주소를 참조할 때 스택 오버플로우를 통해 리턴 주소를 덮어쓰는 값인 4바이트 주소를 전부 사용했다. 그러나 이번 경우에서 하위 2바이트의 상대적인 오프셋만 사용한 이유는 EQNEDT32.EXE의 ASLR(Address Space Layout Randomization)을 우회하기 위한 목적으로 보인다. 2017년 11월 보안 업데이트 이전 버전의 EQNEDT32.EXE에는 ASLR이 활성화되지 않았지만, 해당 보안 업데이트를 위해 새로 빌드된 EQNEDT32.EXE 에는 ASLR이 적용된 상태이기 때문이다.

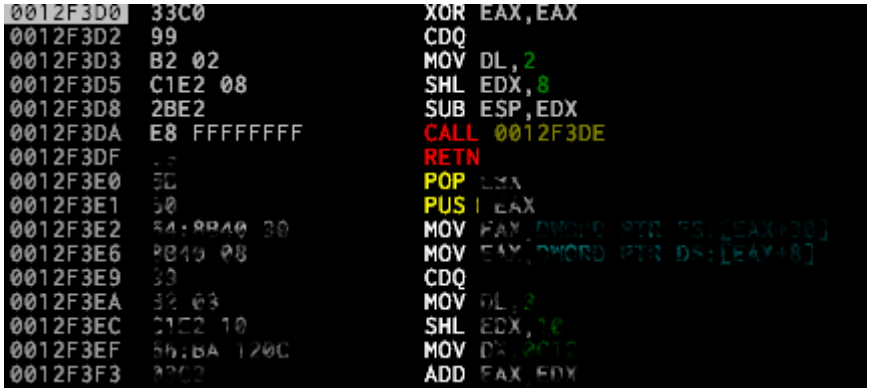
CVE-2018-0802가 제대로 동작하기 위해서는 2017년 11월 보안 업데이트가 적용되어야 한다는 전제 조건이 존재한다. 즉, 해당 익스플로잇이 동작할 EQNEDT32.EXE 환경에서는 ASLR이 활성화된 상태이며, 따라서 해당 프로그램/코드가 메모리 상에 로드된 베이스 주소를 보장할 수 없다. 이러한 환경을 극복하기 위해 전체 주소(4바이트)가 아닌 상대적인 오프셋(2바이트)만으로 RETN 명령이 있는 주소를 찾아 사용한 것으로 보인다.



[그림 214] 2017년 11월 보안 업데이트 전후의 EQNEDT32.EXE의 ASLR 설정 값

RETN 명령으로 인해 EIP 전환이 이루어지면 [그림 25]와 같이 strcpy() 시에 ESI로 사용된 @12F3D0 위치로 코드가 분기하여 쉘코드가 실행된다.

 © AhnLab, Inc. All rights reserved.



[그림 225] CVE-2018-0802 익스플로잇 성공 후 쉘코드

안랩 제품의 대응 현황

안랩 V3 제품군에서는 CVE-2017-11882 및 CVE-2018-0802 취약점 관련 악성코드를 다음과 같은 진단명으로 탐지하고 있다.

<V3 제품군 진단명>

- RTF/Cve-2017-11882
- RTF/Exploit
- RTF/Shellcode
- X97M/Downloader
- PDF/Dropper

 © AhnLab, Inc. All rights reserved.

결론

본 보고서에서 살펴본 MS오피스 수식 편집기(EQNEDT32.EXE) 프로그램의 취약점은 처음부터 존재했음에도 불구하고 배포 후 17년이 지나 보안 연구가들이 발견할 때까지 알려지지 않았다. 해당 취약점이 발표된 이후 악성코드 제작자들은 공격적으로 이를 이용한 악성 파일을 제작, 배포하는 사례가 다수 확인되고 있다.

이 사례는 우리에게 시사하는 바가 크다. 지금 이 순간에는 별 다른 문제 없이 사용하고 있는 프로그램이라도 단지 아직 알려지지 않았을 뿐 다수의 약점이 존재할 수 있으며, 동시에 언제라도 공격에 악용될 수 있다는 가능성을 내포하고 있기 때문이다.

알려지지 않은 취약점에 의한 보안 위협의 피해를 방지하기 위해서는 프로그램 제작사, 사용자, 그리고 보안 관계자의 노력이 동반되어야 한다. 프로그램 제작사는 지속적인 모니터링과 더불어 제품의 취약점 보완 시 미처 해결되지 않은 취약한 요소가 존재하지 않는지, 혹은 예상치 못한 영향은 없는지 더욱 신중하고 면밀하게 검토해야 한다. 사용자들은 최신 보안 업데이트를 적극적으로 적용하는 습관이 필요하다.

또한 기업과 기관에서는 알려지지 않은 공격에 대한 대응 방안을 마련하는 한편 다양한 운영체제 및 소프트웨어 프로그램의 보안 패치를 효율적으로 적용할 수 있어야 한다. 특히 보안 관리자는 평소 취약점 및 보안 업데이트 소식에 관심을 가져야 한다. 최근 많은 보안 연구가들이 취약점뿐만 아니라 보안 업데이트에 대한 분석을 수행하고 있으며, 업데이트로 인해 어떠한 부분이 변경 되었는지, 또 취약한 요소를 제대로 보완하였는지 등에 대한 상세한 자료를 공개하고 있다. 여기에 안랩을 비롯한 주요 보안 업체 및 관련 기관에서 제공하는 위협 인텔리전스를 적극적으로 활용하는 노력이 필요하다.

AhnLab © AhnLab, Inc. All rights reserved.

참고 자료

[1] https://nvd.nist.gov/vuln/detail/CVE-2017-11882

[2] https://nvd.nist.gov/vuln/detail/CVE-2018-0802

[3] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-11882

[4] https://embedi.com/blog/skeleton-closet-ms-office-vulnerability-you-didnt-know-about/

[5] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-0802

[6] https://0patch.blogspot.kr/2018/01/the-bug-that-killed-equation-editor-how.html

AhnLab © AhnLab, Inc. All rights reserved.