SPECIAL REPORT

GHOSTSCRIPT VULNERABILITY

고스트스크립트 취약점 이용한 악성 한글 파일 분석

사라지지 않는 '유령'... 한글 파일 타깃 공격의 실체

지난 2010년 이후 한글과컴퓨터(이하 한컴)의 한컴오피스의 한글 워드(이하 한글) 파일을 이용한 공격이 본격적으로 나타났다. 한글 프로그램은 주로 우리나라 공공기관과 일부 기업에서 사용하는 워드프로세서인 만큼 한글 파일을 이용한 지속적인 공격은 결국 이들에 대한 타깃 공격임을 짐작할 수 있다. 한글 파일을 이용한 공격을 지속적으로 추적, 분석해온 안랩 시큐리티대응센터(AhnLab Security Emergency response Center, 이하 ASEC)는 공격자들이 최근 2년간 고스트스크립트 취약점을 집중적으로 이용하고 있음을 확인했다.

지난 2년간 유포된 악성 한글 파일을 중심으로 고스트스크립트 취약점을 이용한 악성코드 동작 과정을 살펴본다.

한글 파일(.hwp) 공격은 대부분 공격 대상이 분명한 타깃형 공격이다. 주로 우리나라 공공기관과 일부 기업에서 한글 프로그램을 사용한다는 점 외에도 악성 한글 파일을 수신하는 대상, 즉 조직이나 사용자의 업무나 상황과 관련된 내용으로 위장하고 있기 때문이다.

한글 파일 공격은 대부분 한글 프로그램의 기능이나 취약점을 이용한다. 공격 시기에 따라 실행 방식이 계속 달라졌다. 안티바이러스 제품 사용이나 보안 패치 적용 등에 대한 국내 사용자들의 보안 인식이 높아짐에 따라 공격 성공률을 높이기 위해 공격 방식을 변경할 수밖에 없었던 것으로 보인다.

그런데 최근 악성 한글 파일을 이용한 일련의 공격에서 특징적인 모습이 포착됐다. 2017년 6월을 시작으로 2019년 6월 현재까지, 약 2년간 유포된 악성 한글 파일 중 상당수가 특정 취약점을 이용하고 있다는 점이다. 바로 한글 파일에 삽입된 EPS(Encapsulated PostScript) 파일 처리와 관련된 취약점이다. 한컴은 2017년 2월, 이와 관련된 보안 업데이트를 배포하고 한글 프로그램의 EPS 파일 삽입 및 보기 기능을 제거했다. 그럼에도 불구하고 여전히 EPS 파일을 이용한 악성 한글 파일이 유포되고 있다.

EPS, 그리고 고스트스크립트 취약점

2017년 2월 보안 업데이트 이전의 한글 프로그램은 포스트스크립트(PostScript) 언어로 작성된 그래픽 파일을 문서 내에 이미지로 삽입하거나 읽는(보기) 기능을 제공했다. EPS 파일이란 일종의 그래픽 파일 포맷으로, 주로 고품질 인쇄 및 출력을 위해 사용된다. 구 버전의 한글 프로그램은 EPS 파일을 문서 내에 삽입하고 문서에 삽입된 EPS 그래픽 이미지를 볼 수 있는 기능을 제공했다.

EPS 파일을 이용한 악성 한글 파일 공격은 지난 2015년에 처음 확인됐으며, 현재 대부분의 악성 한글 파일은 EPS 파일을 이용하고 있다. 악성 EPS 파일은 동작 방식에 따라 몇 가지 유형으로 구분되지만, 최근 2년간 확인된 악성 한글 파일 중 상당수가 한글 프로그램에 내장된 EPS 파일 인터프리터인 고스트스크립트(Ghostscript) 프로그램 관련 취약점인 'CVE-2017-8291'을 이용하고 있다.

CVE-2017-8291은 포스트스크립트 언어를 처리하는 고스트스크립트 인터프리터가 EPS 파일을 처리하는 과정에서 발생하는 취약점으로, 일명 '고스트벗(GhostButt)'이라고 불린다. 공격자는 이 취약점을 이용해 미리 만들어둔 악성 EPS 파일을 공격 대상(사용자)의 눈을 속이기 위한 한글 문서 내부에 삽입하여 유포한다.

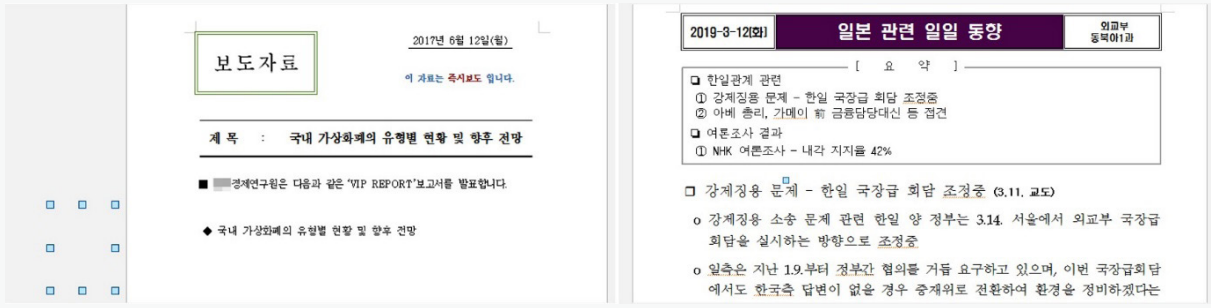
주요 악성 한글 파일 공격 사례

CVE-2017-8291 취약점을 이용한 악성 한글 파일이 처음 확인된 것은 지난 2017년 6월로, 이후 2년 가까이 동일한 방식의 악성 한글 파일이 제작, 유포되고 있다. 이들 악성 한글 파일은 주로 공공기관, 암호화폐 관련 기업을 대상으로 유포되었으며, 통일부나 외교부 등 정부 기관, 기업 또는 개인 구직자를 사칭하거나 사회적으로 이슈가 되는 내용으로 위장하는 등 타깃 공격의 특성을 보였다. [표 1]은 지난 2년간 유포된 주요 악성 한글 파일을 요약한 것이다.

시기	파일명 또는 내용	비고
2017년 6월	국내 가상화폐의 유형별 현황 및 향후 전망.hwp	· 유명 경제연구원 사칭
2017년 7월	입사지원서.hwp	
2017년 7월	양식1.hwp 외	· 국내 주요 은행 사칭 · 핀테크사업 관련 서류 위장
2017년 8월	(대검)2017임시113호(마약류 매매대금 수익자 추정 지갑주소 164건).hwp	· 비트코인 주소 관련 서류 위장
2017년 8월	[KMC]Self-Certification_Service.hwp	
2017년 8월	귀사에 대한 조사 사전예고 통지	· 공격 대상: 국내 암호화폐 거래소 · 공정거래위원회 사칭
2017년 10월	ComparativeAnalysisObservationReport_171011.hwp	· 공격 대상: 방위산업체
2018년 3월	대표이사 진술서_20141011_수정3.hwp	· 공격 대상: 법무법인업체
2018년 4월	이야랩스-거래처원장.hwp	· 공격 대상: 국내 암호화폐 업체
2018년 5월	미북 정상회담 전망 및 대비.hwp	
2018년 8월	홍은자 이력서.hwp	
2018년 8월	2018 대한민국 대중문화예술상[440].hwp 알트플래닛이해하기[448].hwp	· 공격 대상: 국내 암호화폐 업체
2018년 11월	파일명 미확인 (내용: 「국가안보실 정책자문위원회」 전체회의 계획)	
2019년 2월	파일명 미확인 (내용: 주간 국제 안보군사 정세)	
2019년 3월	3.17 미국의 편타곤 비밀 국가안보회의.hwp	
2019년 3월	20190312 일본 관련 일일동향(완).hwp	· 공격 대상: 주요 정부 기관 · 외교부 사칭
2019년 5월	이벤트 당첨자 개인정보 수집 및 이용 동의 안내서.hwp	· 국내 암호화폐 업체 사칭
2019년 6월	자료(확인).hwp	

[표 1] 주요 악성 한글 파일 (*기간: 2017년 6월~2019년 6월)

앞서 언급한 것처럼 EPS 파일은 한글 문서 내에 그래픽 이미지로 삽입될 수 있다. 즉, 공격자는 [그림 1]과 같이 공격 대상을 속이기 위한 한글 파일을 준비해 악성 EPS 파일을 삽입한다. 그러나 [그림 1]에서도 볼 수 있는 것처럼 삽입된 EPS 파일은 그림 개체로서 존재하긴 하지만 실질적인 이미지는 나타나지 않는다.



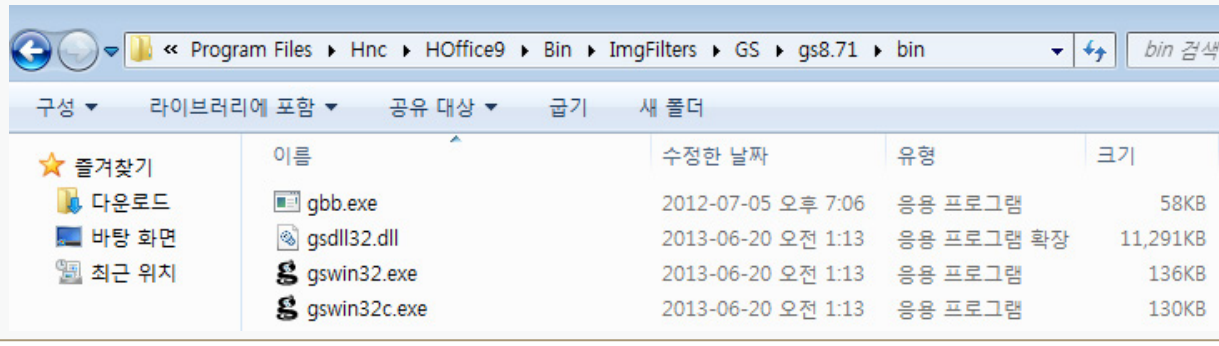
[그림 1] 한글 문서에 삽입된 악성 EPS 파일

2년간 동일한 취약점 이용, 왜?

공격자는 성공률을 높이기 위해 계속해서 공격 방식에 변화를 주는 것이 일반적이다. 그러므로 동일한 고스트스크립트 취약점을 이용한 공격이 2년 가까이 계속된다는 것은 상당히 이례적이다. CVE-2017-8291 취약점을 이용한 공격이 장기간 지속될 수 있었던 이유를 꼽는다면 해당 취약점의 동작 방식을 생각해볼 수 있다.

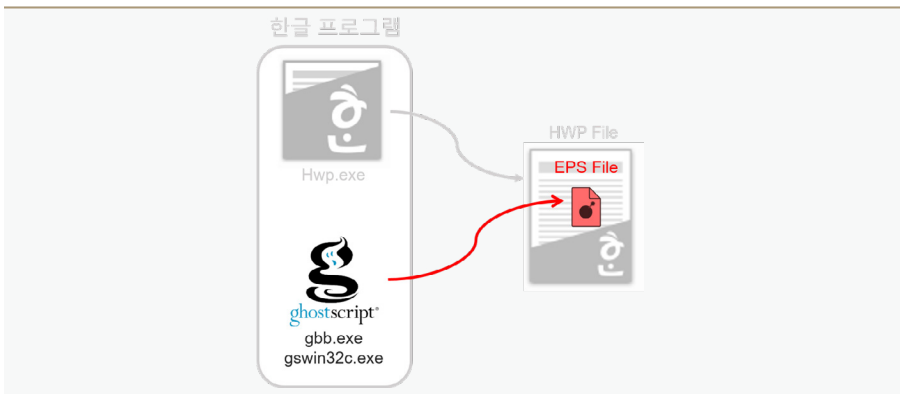
앞서 설명한 것처럼 EPS 파일은 포스트스크립트 언어로 작성된 규격화된 파일로, 이 파일을 화면에 표현하기 위해서는 인터프리터가 필요하다. 인터프리터의 종류는 다양하며, 대표적으로 고스트스크립트(Ghostscript) 외에 포스트캔버스(PostCanvas), 피닉스페이지(PhoenixPage) 등이 있다. 한글 프로그램에 내장된 포스트스크립트 인터프리터는 고스트스크립트로, 한글 프로그램 설치 경로에 별도의 파일로 존재한다.

[그림 2]는 한글2014 설치 시 확인할 수 있는 고스트스크립트 8.71 버전 인터프리터 파일의 경로이다. 기본적으로 4개의 파일로 구성되어 있는데, 인터프리팅 기능을 하는 핵심 파일은 gsdll32.dll 라이브러리이다. 그 외 EXE 파일은 라이브러리 함수를 호출하는 역할을 하는데, 한컴에서 제작한 gbb.exe 파일을 제외하면 모두 공개된 고스트스크립트 소스로 컴파일된 파일이다.



[그림 2] 한글 프로그램(한글2014)에 내장된 고스트스크립트 파일

한글 프로그램은 문서를 읽는 과정에서 EPS 그래픽 파일이 있을 경우 이에 대한 화면 표시만 고스트스크립트 인터프리터로 넘긴다. 한글 프로그램은 고스트스크립트를 단순 호출하고 작업 이후 종료 여부에 관해서만 확인할 뿐 고스트스크립트 내부에 일어나는 일에 대해서는 관여하지 않는다.



[그림 3] 고스트스크립트의 악성 한글 파일 실행 개념도

악성 한글 파일은 [그림 3]과 같이 한글 파일에 삽입된 EPS 파일이 고스트스크립트 인터프리터를 통해 로드되는 과정에서 악의적인 행위를 수행하는 것이다. 즉, 한글 프로그램이 설치되지 않은 환경에서도 고스트스크립트에 악성 EPS 파일을 실행 인자로 주면 악의적인 기능이 동작한다.

취약점 동작 과정 상세 분석

악성 한글 파일에 포함된 EPS 파일의 취약점 발생 원리와 공격 코드의 동작 방식을 상세히 살펴보자.

1. CVE-2017-8291 취약점 정보

[표 2]는 CVE-2017-8291 취약점에 관한 기본 정보를 요약한 것이다. [표 2]의 정보를 좀 더 상세히 설명하자면, CVE-2017-8291 취약점은 고스트스크립트 인터프리터가 '.eqproc' 함수에서 매개변수 타입 유효성을 검증하지 않아 피연산자 스택의 메모리 변조(Memory Corruption)를 허용한다. 즉, 악의적인 포스트스크립트 파일이 스택을 변조함으로써 임의의 코드를 실행할 수 있다.

취약점 ID	CVE-2017-8291
공개일	2017-04-26
영향 받는 버전	Artifex Ghostscript 9.21 포함 이하 버전
취약점 유형	잘못된 타입 컨퓨전 또는 캐스트 (CWE-704)
취약점 요약	타입 컨퓨전을 통한 원격 명령 실행 허용

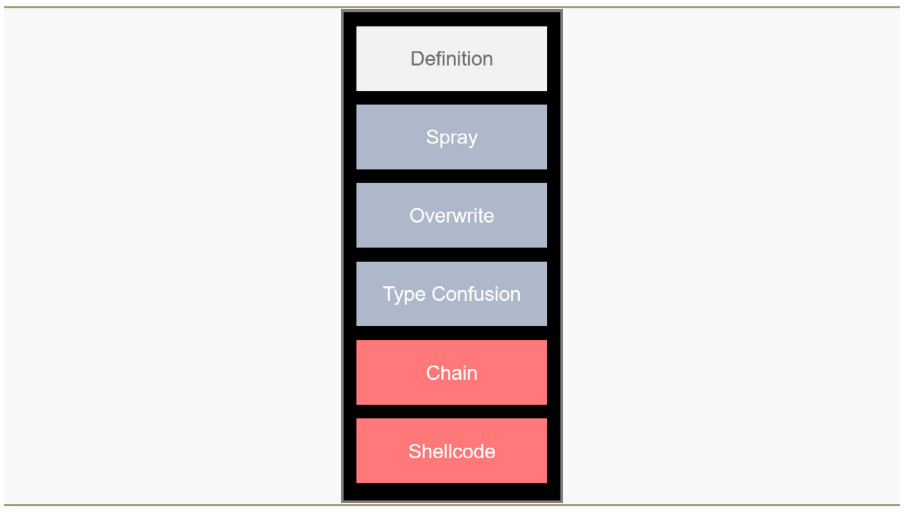
[표 2] 취약점 정보

고스트스크립트의 CVE-2017-8291 취약점을 이용한 악성 EPS 파일은 포스트스크립트의 메모리 관리 방식과 특정 연산자의 내부 실행 결과로 발생하는 타입 컨퓨전(Type Confusion) 버그를 익스플로잇(Exploit)한다. 한글 프로그램이 설치된 시스템의 경우, 고스트스크립트 기능이 구현된 라이브러리 파일인 gsdll32.dll에서 해당 취약점이 발생한다. 참고로 '.eqproc' 함수 외에 '.rsdparams' 함수에서도 동일한 취약점이 존재했다.

CVE-2017-8291 취약점을 이용한 악성 EPS 파일은 정교하게 제작된 포스트스크립트 코드를 이용해 메모리에 접근하여 실행 제어를 변경하는데, 코드가 포스트스크립트의 문법적인 특징을 이용하기 때문에 스프레이

(Spray) 등의 단계가 노출되지 않는다. 또한 스크립트 자체의 유연성을 이용해 숫자를 변수로 처리하거나 인코딩 단계 등을 추가함으로써 더욱 다양한 변형을 제작할 수 있다는 것도 장점이다. 공격 코드가 거의 노출되지 않으며 변형 제작이 쉽다는 것은 곧 안티바이러스 제품 등이 탐지하기 까다롭다는 의미이기 때문이다. 이것이 지난 2년간 이 취약점을 이용한 악성 한글 파일이 지속적으로 제작, 유포될 수 있었던 이유라 할 수 있다.

한편, 최초의 악성 EPS 파일이 유포된 이후 수집된 악성 EPS 파일들은 포스트스크립트 코드 패턴이 각기 달랐다. 대부분 인코딩 스타일에 따라 몇 가지 유형으로 분류할 수 있는데, 변형이 계속될수록 인코딩 방식 또한 복잡하게 변화했다. 또 인코딩 되어 있지 않은 상태에서 변수만 '/a1', '/a2' 등으로 치환된 형태도 확인되었다. 공격자에 따라 포스트스크립트 코드 패턴이 각기 다른 것으로 추정되는데, 결과적으로 이들 악성 EPS 파일이 최종적으로 실행하는 코드는 모두 동일하다. 그러나 EPS 파일의 외형이 각기 다르다는 점에서 안티바이러스 제품이 탐지하기 까다롭다.



[그림 4] EPS 파일의 실행 흐름

2. 익스플로잇 분석

고스트스크립트는 EPS 파일의 포스트스크립트 코드를 순차적으로 읽으면서 실행한다. EPS 파일의 포스트스크립트 코드는 [그림 4]와 같은 흐름을 갖는다. 이때 메모리에 실시간으로 연산 결과와 실행 단계의 스택을 포함한 컨텍스트가 반영된다. 단, 최종 실행하는 쉘코드 등 일부 변수의 선언 위치는 파일에 따라 다르다.

(1) 정의(Definition)

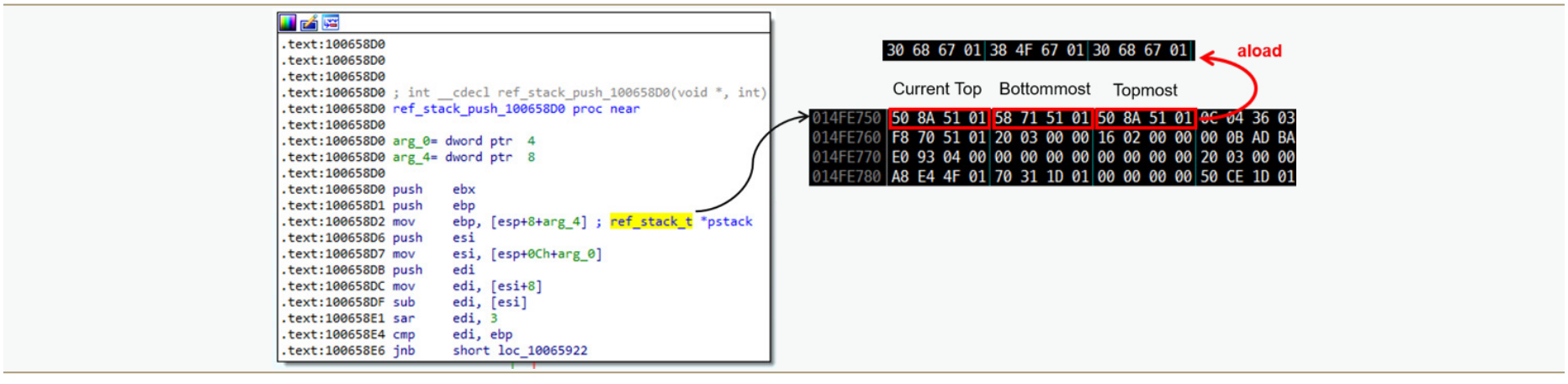
악성 EPS 파일이 쉘코드를 포함해 문자열, 배열, 프로시저 등 실행에 필요한 객체를 정의하는 단계다. def 연산자를 이용해 정의된 객체는 딕셔너리 스택에 키 값(Key Value)으로 추가된다. 실행 과정에서 해당 키를 가진 객체 이름이 확인되면 값으로 인식된다.

bin def는 정의하는 현재 시점을 기준으로 객체의 값을 정의한다. 포스트스크립트의 실행 과정에서 프로시저에 포함된 객체의 값이 변경될 수 있기 때문에 변경 사항이 적용되지 않은 상태의 객체 정보를 이용한다. 쉘코드 이름의 16진수 문자열은 악성 EPS 파일의 실행 후반에 이용되므로 정의는 파일의 시작 부분뿐만 아니라 중간에도 위치할 수 있다.

(2) 스프레이(Spray)

악성 EPS 파일은 스프레이 단계에서 스택 포인터를 높은 주소로 올리고 고정된 길의 문자열을 여러 번 반복해서 스택에 푸시(Push)한다. 스택의 주소를 올리기 위해서 aload 연산자가 여러 번 반복되어 호출된다. aload 연산자는 피연산자 배열의 모든 엘리먼트(Element)를 스택에 로드하는 역할을 한다.

악성 EPS파일은 16#31E 크기의 first_array 배열을 aload하고 16#215 크기의 second_array를 16#10회 aload한다. 이 크기는 스택의 최상위(Topmost)와 최하위(Bottom) 주소 간의 크기 차이인 16#18F8를 훨씬 초과하는 값으로, 새로운 스택 영역을 동적으로 할당받아 엘리먼트를 저장한다.



[그림 5] 스택을 초과하는 크기의 배열 로드 시 스택 정보 변경

16#152F 길이의 문자열 객체인 control_string을 선언하고 문자열을 내용을 모두 1로 채운 뒤, 해당 문자열을 버퍼(Buffers) 배열이 참조하도록 한다. 버퍼 문자열은 16#100 개의 엘리먼트가 있는데, 모든 엘리먼트가 별개의 control_string을 참조하도록 반복한다. 반복문 실행 이후에는 스택 포인터가 이전보다 더 증가한다. 스프레이 프로시저 종료 후에는 second_array와 final_array를 한 번 더 aload한다. second_array는 16#215 크기의 배열이고 final_array는 16#1 크기의 배열이다.

(3) 덮어쓰기(Overwrite)

eqproc 연산자를 이용해 현재 스택 포인터를 하나씩 감소시킨다. 버퍼의 16#100번째 Element가 가리키는 문자열 객체인 control_str의 시작 위치를 기준으로 16#150F 번째 문자를 overwrite_pos로 지정한다. 스택의 감소는 overwrite_pos 위치의 문자가 0이 될 때까지 진행된다.

스택 포인터가 control_str 문자열의 이전까지 오면 문자열의 마지막 8바이트를 True 값인 Boolean 객체로 인식하고 False 값인 Boolean 객체와 비교한다. 비교 결과는 당연히 False이며, 푸시(Push)로 인해 control_str 문자열 값이 일부 변경된다. 이 과정은 overwrite_pos로 지정한 16#150F번째 문자까지 지속되며 overwrite_

pos 위치까지 변경되면 .eqproc 연산의 반복은 종료된다.

(4) 타입 컨퓨전(Type Confusion)

스택 포인터는 값이 0으로 변경된 최초 위치(overwrite_pos)를 가리키고 있다. 이후 Element를 16#FFFF개 가지는 배열 leaked_array와 정수 0, 그리고 leaked_array를 한 번 더 스택에 푸시한다. 16#FFFF 크기는 배열이 가질 수 있는 가장 최대 크기이다.

overwrite_pos를 기준으로 +16#18, +16#19, +16#1A, +16#1B 위치에 각각 16#7E, 16#12, 16#00, 16#80 값을 put 연산자를 통해 저장한다. control_str은 문자열 객체이기 때문에 참조하고 있는 특정 위치에 직접 접근 및 연산이 가능하다. overwrite_pos에서 +16#18, +16#19, +16#1A, +16#1B 위치는 스택에 마지막으로 PUSH된 leaked_array 배열의 객체 정보이다. 입력된 값으로 인해 객체의 타입과 참조하는 주소의 크기가 변경되었다. leaked_array는 기존 16#FFFF 크기의 배열 객체에서 16#8000 길이의 문자열 객체로 변경되었다.

타입을 변경한 이후에 put 연산자를 통해 leaked_array 배열의 0번째 위치에 leaked_array 문자열 객체를 저장한다. 두 leaked_array는 변경 전 동일한 배열이었기 때문에 참조하는 주소가 같다. 배열을 통해서는 해당 주소 위치에 직접 접근이 불가능한 부분을 문자열을 통해 접근할 수 있게 된다.

(5) 체인(Chain)

위의 과정이 취약점을 이용해 권한을 갖고 메모리 영역에 접근하는 것이었다면, 지금부터는 본격적으로 쉘코드 실행을 위한 준비 단계이다. leaked_array 배열의 1번 엘리먼트에 {lt} 프로시저 객체를 저장한다. 객체 저장은 내부적으로 고스트스크립트의 zfilenameforall 함수의 주소 호출로 연결된다. zfilenameforall 함수의 주소를 획득해 현재 실행 중인 gsdll32.dll PE 구조를 메모리에서 찾는데 이용한다.

gsdll32.dll에서 로드하고 있는 Kernel32.dll 라이브러리를 찾은 다음 VirtualProtect 함수와 ExitProcess 함수의 호출 주소를 저장한다. 이 과정은 정의(Definition) 단계에서 구현된 PE 파일 파싱을 통해 실행된다. 두 개의 함수는 쉘코드를 실행하기 전과 후에 각각 실행된다. 이후 gsdll32.dll의 메모리에서 가젯으로 사용할 코드를 찾고, 그 주소를 저장한다. 가젯은 RETN과 스택 피봇(Pivot)을 목적으로, 의도적으로 코드 흐름이 쉘코드로 분기하도록 한다.

API 함수와 가젯의 주소를 획득하고 나면 쉘코드가 저장된 주소와 함께 순서대로 엮는 작업이 필요하다. leaked_array 배열의 1번 엘리먼트에 다음과 같은 3가지 핵심 정보를 순서대로 PUT한다. 배열에 저장된 이후

에는 문자열 객체 주소인 base_addr를 통해 주소 값에 접근할 수 있다.

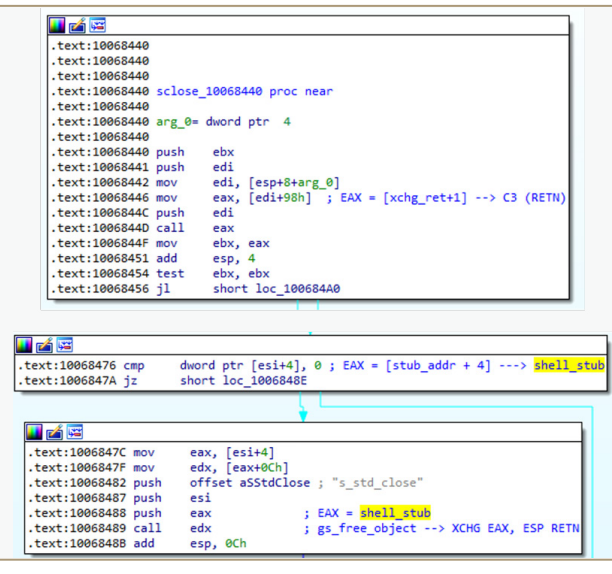
① 쉘코드가 저장된 주소를 shell_addr로 저장
② 16#100 길이의 문자열을 stub_addr로 저장
③ currentfile 연산자 실행으로 전달되는 stream_state 구조체 주소를 file_addr로 저장

API와 가젯 주소, shell_addr는 shell_stub에 지정된 상대 주소만큼 떨어져서 저장된다. 그리고 shell_stub은 stub_addr의 +16#30 위치를 시작으로 한다. 즉, stub_addr 주소만 알면 API, 가젯, 쉘코드를 모두 실행할 수 있다. stub_addr가 실행이 되기 위해서 포스트스크립트 연산자가 실행될 때 호출되는 함수와 연결하는 작업이 필요하다. 악성 EPS 파일은 이 부분을 파일 객체 연산자로 연결한다.

(6) 쉘코드(Shellcode)

closefile 연산자를 실행하는 과정에서 sclose 함수가 호출된다. sclose 함수는 stream_state 구조체를 참조하는 부분이 두 군데 있다. 첫 번째 참조 부분 *s->procs.close은 stream_state 구조체의 +16#98 위치에 있는 멤버이며, 앞 단계에서 C3(RETN)으로 변경하였다. 따라서 close 함수 호출이 되지 못하고 종료된 뒤 다음 코드를 실행한다. 두 번째 참조 부분은 gs_free_object(st->memory, st, “s_std_close”)이다. stream_state 구조체의 +16#B0 위치에 있는 stub_addr을 기준으로 상대 주소를 찾아간다.

교체된 코드는 gs_free_object를 하는 CALL EDX 시점에 스택을 피봇한 뒤 EAX로 분기를 변경한다. 변경된 EAX는 RETN 0C이며, 리턴 주소 변경 이후 VirtualProtect 함수로 올려둔 쉘코드의 실행 권한을 0x40으로 변경한다. 최종적으로 쉘코드 분기가 이루어진다.



[그림 6] sclose 함수에서 변경된 코드 호출

유령처럼 되살아나는 취약점 공격, 해법은?

지금까지 CVE-2017-8291 고스트스크립트 취약점을 이용한 악성 한글 파일의 동작 방식에 대해 살펴보았다. 2017년 2월 23일 관련 패치가 배포되었음에도 불구하고 2년 가까이 동일한 취약점을 이용한 동일한 방식의 공격이 계속되고 있다는 것은 그것이 여전히 성공률이 높은 유효한 공격이라는 의미다. 바꿔 말하면 패치가 적용되지 않은 취약한 버전의 한글 프로그램을 이용하는 사용자들이 많다는 뜻이다.

더 큰 문제는 이것이 한글 파일을 이용한 공격이라는 점, 그리고 한글 파일은 대개 공격 대상이 명확한 타깃형 공격이라는 점이다. 게다가 고스트스크립트는 윈도우(Windows) 시스템뿐만 아니라 우분투(Ubuntu), 레드햇(Redhat) 등 리눅스(Linux) 시스템에서도 실행된다. 즉, 이들 시스템에서 9.21 버전 이하의 고스트스크립트가 실행될 경우, 동일한 취약점이 발생한다. 따라서 공공기관 및 기업에서는 조직 내 다양한 운영체제와 프로그램과 관련해 최신 보안 패치가 빠짐없이 적용될 수 있도록 효과적인 패치 관리 방안을 마련해야 한다.

지금까지 살펴본 내용을 비롯해 상세한 익스플로잇 과정은 안랩 시큐리티대응센터(ASEC)가 발표한 '고스트스크립트 취약점을 이용한 악성 한글 파일 분석보고서'에서 확인할 수 있다.

▶ '고스트스크립트 취약점을 이용한 악성 한글 파일 분석보고서' 전문 보기