SOPHOS
Security made simple.

Vawtrak v2

This paper provides a technical analysis of the most recent variant of the banking malware Vawtrak which we dub “version 2”. We describe alterations to the code, such as changed encryption, increased obfuscation and modularization. We give a breakdown of the financial institutions and other organizations that are currently being targeted within the following countries: United States, Canada, Japan, Romania, Israel, United Kingdom, Republic of Ireland and Czech Republic. We also present data gathered from a sinkhole of a Vawtrak version 1 command and control server which provides evidence of a new cluster in East Asia, suggesting many victims are running older, outdated software.

James Wyke and Erhan Sahin, SophosLabs

Contents

Introduction

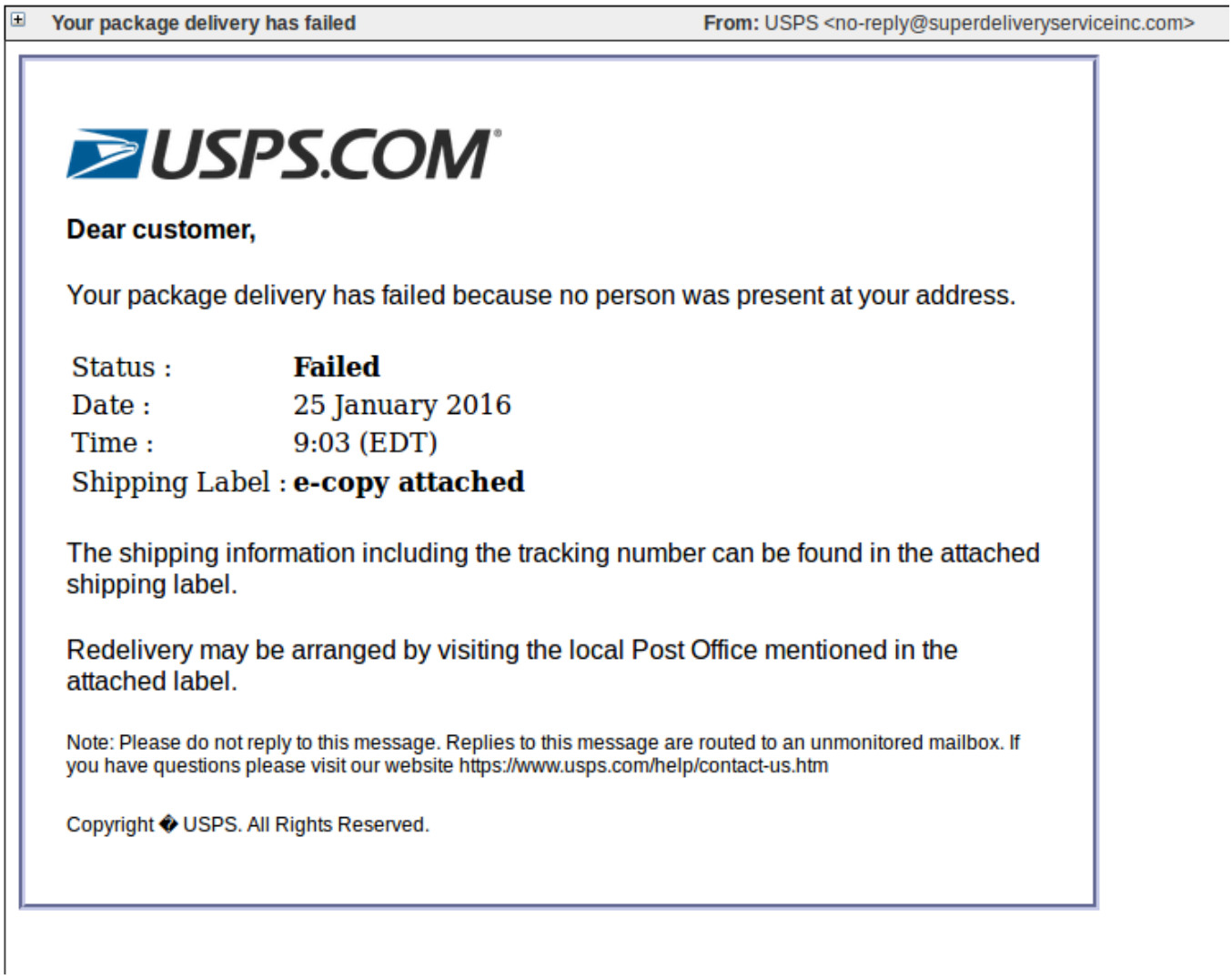
Since our previous analysis of the Vawtrak banking malware [1], there have been several important updates to the code and to the financial institutions and organizations being targeted. There have also been several widespread campaigns that have been utilized with great success to spread the new version of Vawtrak.

In this paper we highlight the campaigns being used to spread Vawtrak, as well as describe the updates that have been made to the malware code in "version 2". We also present details about the current targets based on the URLs found in decoded configuration files. Finally we present further details concerning the victims of Vawtrak by analyzing data retrieved from a sinkhole operation on a Vawtrak command and control address.

New Infection Campaigns

Although Vawtrak is still being distributed through a variety of infection vectors including exploit kits, the most successful methods in the recent campaigns have been via email and the Pony malware family. Pony is a password stealer that can also be sent download tasks via its command and control server. There are many active Pony botnets on the internet that are capable of downloading a variety of different malware components. The specific instances of Pony that distribute Vawtrak have been heavily spammed out using a variety of themes, although all have several discernible patterns.

The Pony executable is frequently embedded in a Microsoft Office document file. Usually this is a Word file. Some common themes observed in the spam message campaigns have been package delivery failures, fax deliveries and invoice notifications. *Figure 1* and *figure 2* show example spam messages that have been used to deliver Pony. *Figure 1* shows a delivery failure message purporting to be from *USPS* with the Word document containing Pony attached, and *figure 2* shows a delivery failure message from *UPS* where the document containing Pony is delivered by a link that the victim must click.



Your package delivery has failed **From:** USPS <no-reply@superdeliveryserviceinc.com>

USPS.COM

Dear customer,

Your package delivery has failed because no person was present at your address.

Status :	**Failed**
Date :	25 January 2016
Time :	9:03 (EDT)
Shipping Label :	**e-copy attached**

The shipping information including the tracking number can be found in the attached shipping label.

Redelivery may be arranged by visiting the local Post Office mentioned in the attached label.

Note: Please do not reply to this message. Replies to this message are routed to an unmonitored mailbox. If you have questions please visit our website https://www.usps.com/help/contact-us.htm

Copyright © USPS. All Rights Reserved.

Figure1 – Pony email 1

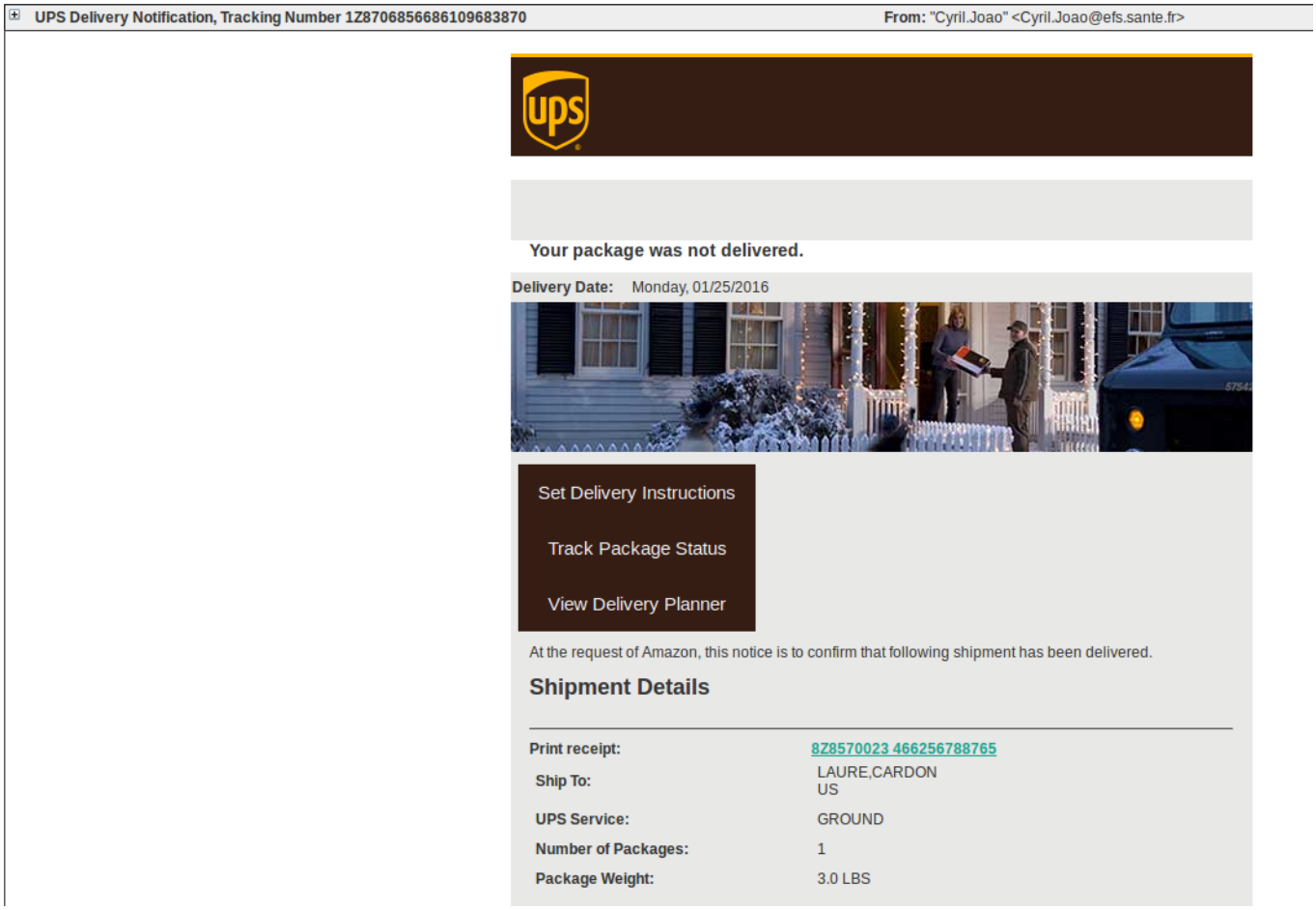


Figure 2 – Pony email 2

Some common attachment names for the malicious documents are included in *Table 1*. Several examples use the current date in numerical form as part of the file name and several others use a sequence of random numbers.

Common Pony attachment names
usps_label_88380594.doc
ups_invoice-01262016.doc
Jan_invoice.doc
Invoice_confirmation.doc
receipt_20160125.doc

myfax_173465726589.doc
NoCallerID-1209-084646-258.doc

Table 1 – typical Pony attachment names

When opened, the document will typically instruct a potential victim to enable macros through some form of social engineering - a technique that has become extremely common over the last few years. *Figure 3* shows the message that is used to trick the victim.

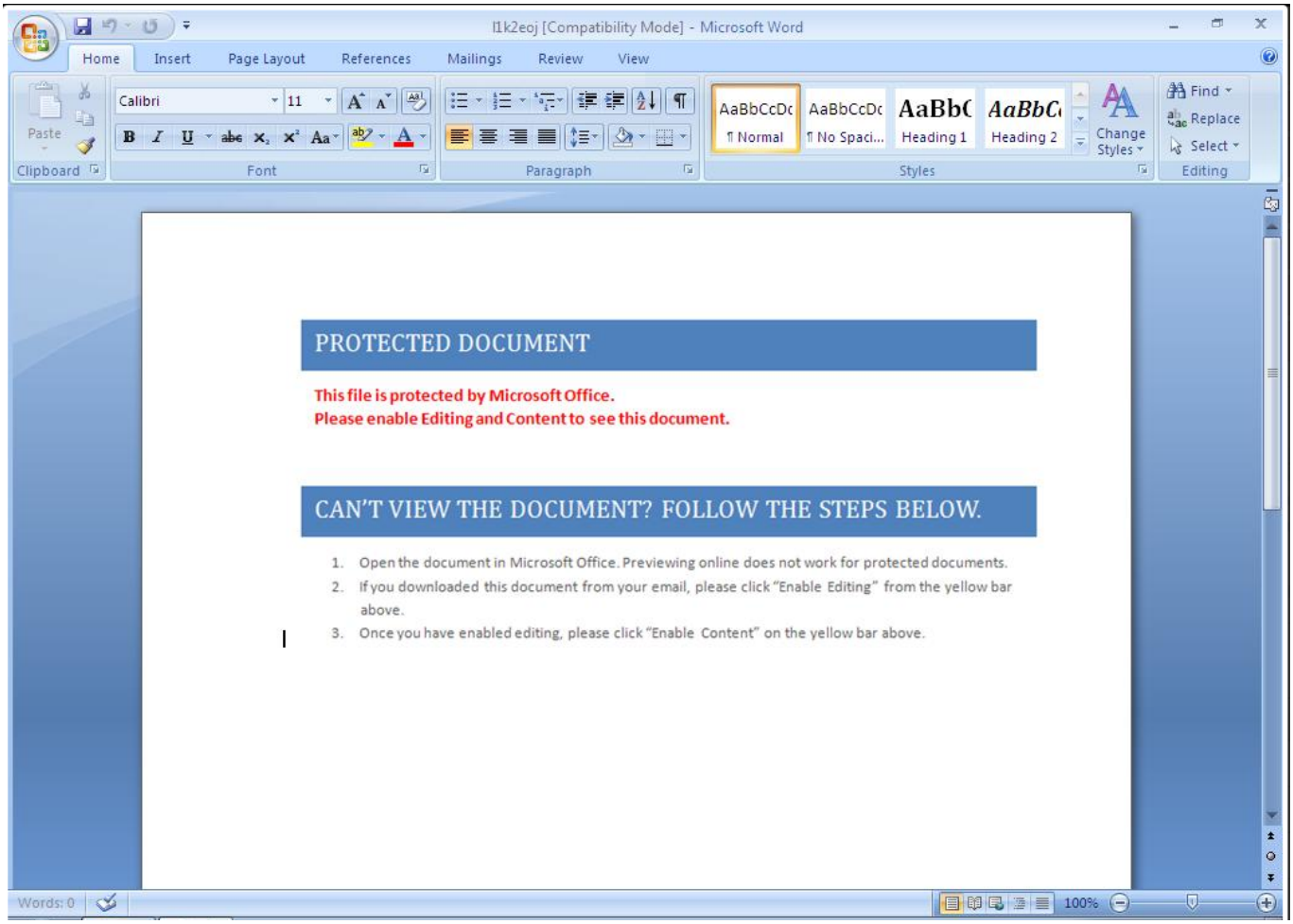


Figure 3 – Pony document

The format of the Word document appears to have remained relatively constant over the last few months, utilizing the same format that was documented in the analysis in [2].

When macros are enabled, the Pony executable is dropped to the TEMP directory and launched. In some cases, the macro saves the file as an RTF which automatically drops an exe file into the TEMP

directory so that it can be executed, as described in [5]. It steals saved credentials from a large number of applications, including browsers, email clients and FTP clients, sending them back to its command and control server which in turn sends back the Vawtrak download task. Vawtrak is then downloaded and installed on the system.

Updates to the Malware

We will now describe some of the updates to the Vawtrak code that have been made, and explain the rationale behind naming this “version 2”.

Versioning

Vawtrak samples include an embedded version value, referred to internally as “BUILD”. In December 2014 this value was 56 (0x38 in hexadecimal) and had been gradually increasing throughout the year, with minor updates to the code, reaching 67 (0x43 in hexadecimal) by September 2015 when we saw the first samples of the significantly updated version. *Figure 4* shows the location in the code where we can find the version value in older samples.

```
loc_270E02D:                ; CODE XREF: Construc
    push    6
    push    offset aBuild ; "BUILD:"
    push    esi
    call    edi ; StrCmpNA
    test    eax, eax
    jnz     short loc_270E042
    push    38h             ; BUILD number
    pop     ebx
    push    6
    jmp     short loc_270E08B
; ---------------------------------------------
```

Figure 4 – BUILD value in older samples

When the newer variants were first observed in the wild, this BUILD value was reset, with the first examples having a value of 2 (example sha1: 7a479295549330798bed66599e22b5cf5580194c). So, although the Vawtrak code can be traced back some considerable time (as described in the history provided in [3]) we feel a solid argument can be made to call this Vawtrak “version 2”, allowing us to differentiate this significantly different variant from the earlier ones.

Figure 5 shows the location where the BUILD value can be found in version 2. At the time of writing the latest BUILD version is 12 (0xc in hexadecimal).

```
loc_492CB4:                ; CODE XREF: DecryptPostString
    lea     eax, [esp+260h+var_224]
    push    eax
    push    offset unk_4A1D2C ; 0x4a1d34: BUILD:
    call    DecryptString ; No APIs
    pop     ecx
    pop     ecx
    push    6
    lea     eax, [esp+264h+var_224]
    push    eax
    push    esi
    call    ebp ; StrCmpNA
    test    eax, eax
    jnz     short loc_492CDA
    push    2               ; BUILD number
    pop     ebx
    push    6
    jmp     short loc_492D32
```

Figure 5 – BUILD value in version 2

Obfuscation and Encryption

Several of the updates included in Vawtrak version 2 center around complicating the analysis process and breaking existing tools used to decode interesting data used by the malware. These changes involve increased levels of obfuscation and changes to the encryption used.

As described in [2], many strings inside the binary that were previously discernible as plain text are now encrypted. A Linear Congruential Generator is still used for most encryption operations but the parameters have been changed from their previous values. The two key parameters are called the *multiplier* and the *increment*. Unsurprisingly, the multiplier is the value that is used for multiplication and the increment is used for addition. The previous version of Vawtrak used a multiplier of 214013 (0x343fd) and an increment of 2531011 (0x269ec3) which are the parameters used by the *rand()* function in Microsoft Visual C. The new version uses 1103515245 (0x41c64e6d) and 12345 (0x303) respectively which are the parameters used by glibc and ANSI C. For more information see [4]. Changing the parameters does not appear to offer any increased level of security. As such, the motivation for the change would appear to be an attempt to temporarily break existing tools that may implement the algorithms used by previous Vawtrak samples.

The strings that are now encrypted are decrypted dynamically as they are needed. This is a common technique used by a wide variety of malware that is intended to hinder analysis. A simple IDA Python script that decrypts the strings and highlights cross-references to them, is included in Appendix A.

Modularization

The Vawtrak authors have re-architected the main malware binary, making it leaner by pulling out various pieces of functionality and moving the code into DLL modules that are initially downloaded by the main module and thereafter stored in an encrypted form on the disk. This reduces the footprint of the initial payload used for infection and creates the potential for advanced features to be added and deployed as modules to select victims. However, so far we have only observed standard modules to be deployed that replicate the functionality previously found in the monolithic executable.

Modules are delivered through a specific command issued to the client after a check-in request to the command and control server. Vawtrak commands are simply numeric values. The command number for the *GET_MODULE* request is 3. The data blob sent back by the command and control server contains the command number as well as a list of URLs from which the modules will be retrieved. A *GET* request is made to each URL and the response contains the encoded module.

The encoded module data is decoded using the LCG decryption algorithm using a seed value found at the start of the data. The downloaded module data seem to always include two actual module files – one for x86 and one for x64 systems. *Table 2* describes the encoded module structure.

Offset	Field description
0x0	Seed
0x4	File name seed and version (Xor'ed with lower half of seed)
0x6	Size of data (this is the start of the LCG alg encrypted data)
0xa	Signature of module data
0x8a	Offset of module 1 data
0x8e	Same as above
0x92	Size of chunk including header
0x96	Size of chunk excluding header
0x9a	Size of second chunk

Table 2 – Module structure description

The module data itself then follows, compressed with LZmat. This compression algorithm has replaced aPLib which was used in the previous version. The reasons for this again appears to be to break existing

tools, as any performance or compression size benefits that one algorithm may offer over the other are likely to be irrelevant.

Once the module has been downloaded and verified it will be re-encrypted using a machine-specific key and written to a file on disk that is generated utilizing the *file name seed* value included in the data previously received from the command and control server.

The re-encryption again uses the LCG algorithm but uses the size of the file combined with the *VolumeSerialNumber* of the drive that the path returned from *GetTempPathA* is located on. The file name for each module is deterministically generated using the lower byte of the main module's file name and the volume serial number of the drive as seed material for a pseudo-random name generation algorithm. This means that on the same system each plugin with the same *file name seed* will produce the same file name. The authors appear to have devised this mechanism to ensure that only one version of the module will get loaded. The top byte of the *file name seed* value appears to be used as a version value as we have observed this value to increase with updates to the module.

The decoded modules can be identified by their DLL name as well as their *file name seed* value. So far we have only observed 4 modules, each of which have had several revisions. *Table 3* lists the observed modules.

DLL name	File name seed	Revision	Architecture	Sha1
bc_32.dll	0x02	0x10	32	8e2e0333e71abf3484faa2f3523cf520c65bd149
bc_32.dll	0x02	0x11	32	92e53476b4eb5fff29cbd0bca6363bc9d557dca2
bc_32.dll	0x02	0x0f	32	d5136c610a9eb131c6d80f52778ee1387ad45063
bc_64.dll	0x02	0x10	64	788cea3100441a7bf3a129eee54f4aca124406ba
bc_64.dll	0x02	0x11	64	f0951af66102578650b48d1014ae92d0fe7c6763
bc_64.dll	0x02	0x0f	64	d7ebecab2a85dabe6f7d9ab5014ecc9cb1c61a77
dg_32.dll	0x03	0x02	32	7f50d80ecc7efc0f0da66db74915d4c515176df8
dg_32.dll	0x03	0x03	32	78d158b6c5c2dba501acc1e8d94a582c7d83fb63
dg_64.dll	0x03	0x02	64	5660802485100ec9e5452ccfc77ea4a369ca6a42
dg_64.dll	0x03	0x03	64	73d0e7bff72bc331839549e70ae5d22ea7019004
injecter_32.dll	0x01	0x03	32	30406b375470481908251f56eef20208c7cc919d
injecter_32.dll	0x01	0x05	32	39534408c4a30200b15e9ccf94813106e4aa9f5c
injecter_32.dll	0x01	0x07	32	43ff0f1da764ac068e79629c3c35e9d17b0b8f2e
injecter_32.dll	0x01	0x08	32	45b72b787aa9ee4020295d0c5ea59aad5ab05d70
injecter_64.dll	0x01	0x03	64	751ab060bde470f16b1b8297f902b598cf74ae61
injecter_64.dll	0x01	0x05	64	c9554cd0c5cf0c1d604fff9539185b7a1f2f6bbc
injecter_64.dll	0x01	0x07	64	87d8af383b3f8bc21e8922e02b2b4f8dbec824a6
injecter_64.dll	0x01	0x08	64	cdaafb366f5e55e3308ba2599472b407fbec3f49
pony_32.dll	0x04	0x02	32	c4b8334cab3cbd4dd64d3292a185a1a6f45afca7
pony_32.dll	0x04	0x03	32	6ea5de34218bc2bf31ec76316f8f3ff2211a8d6d
pony_32.dll	0x04	0x04	32	0510c0718afff091c250486de55ac5983c704078
pony_64.dll	0x04	0x02	64	c75260b0d03076e157eb34ab8131f8c4e33e3d6d
pony_64.dll	0x04	0x03	64	16d681ad85c28d9357616241692da5a6c37ebbca

pony_64.dll	0x04	0x04	64	5f4c536909d717f2081f1c6cb82451a31e35c1a5

Table 3 – observed modules

The *bc_32.dll* and *bc_64.dll* modules are primarily responsible for the VNC server and reverse proxy functionality (*bc* stands for *back connect* which is a common term for reverse proxy behaviour), though there is also other functionality included such as interaction with browsers and email client programs.

The *dg_32.dll* and *db_64.dll* modules are capable of stealing certificates and browsing history and cookies from Firefox and Chrome.

The *injecter_32.dll* and *injecter_64.dll* modules deal with the inline web page modifications or "web injects" in browser processes. It is inside these modules that we find the JavaScript framework code that is very frequently used as part of Vawtrak web injects. *Figure 6* shows that this framework is currently at version 3.

```
    db  49h ; I
Framework_code db 'return new function(Key){this._LastAsync=null;this.Version=3;this'
                        ; DATA XREF: sub_10001D8A+1F↑t
    db '.Query=function(Method,Url,Post,Callback){var xhr=new XMLHttpRequ'
    db 'est();var LastAsync=null;var thisfw=this;Async=(typeof(Callback)='
    db '=',27h,'undefined',27h,')?false:true;Url=',27h,'/',27h,'+Key+',27h,'/',27h,'+Math.r'
    db 'andom()+',27h,'/',27h,'+Url;if(Async==true){this._LastAsync=null;xhr.on'
    db 'readystatechange=function(){try{if(xhr.readyState==4){if(xhr.stat'
    db 'us!=200||xhr.responseText==',27h,'-',27h,'){thisfw._LastAsync=false;if('
    db 'typeof(Callback)==',27h,'function',27h,'){Callback(false)}}else{if(xhr.'
    db 'responseText==',27h,'+',27h,'){thisfw._LastAsync=true;if(typeof(Callbac'
    db 'k)==',27h,'function',27h,'){Callback(true)}}else{thisfw._LastAsync=xhr.'
    db 'responseText;if(typeof(Callback)==',27h,'function',27h,'){Callback(xhr.'
    db 'responseText)}}}}}catch(e){thisfw._LastAsync=false;if(typeof(Call'
    db 'back)==',27h,'function',27h,'){Callback(false)}}}}xhr.open(Method,Url,A'
    db 'sync);xhr.send(Post);if(Async==true){return true}try{if(xhr.ready'
    db 'State==4&&xhr.status==200){if(xhr.responseText==',27h,'-',27h,'){return'
    db ' false}else{if(xhr.responseText==',27h,'+',27h,'){return true}else{retu'
    db 'rn xhr.responseText}}}return false}catch(e){return false}};this.G'
```

Figure 6 – Framework version 3

The *pony_32.dll* and *pony_64.dll* modules carry the stored credential theft capability taken from the leaked *Pony* source code originally included in the main Vawtrak module.

Command and Control Protocol

The command and control protocol used by Vawtrak version 2 has undergone several significant updates. Data is still sent and received over HTTP, but the data structures and the encoding scheme have been changed. The configuration file received from the command and control server has also been updated with the addition of several new sections.

The previous version included a format string specifier embedded inside the binary that formed the URL used to communicate with the command and control server. Elements of information about the bot could be encoded in this URL. One such example is:

/blog/{TYPE:HB}/post.php?topic={PROJECT_ID:HD}&uid=?{BOT_ID:HD}&v={BUILD:HW}&u={UPDATE_VER:HW}

We can see from this string that details such as the *PROJECT_ID* and the *BOT_ID* can be embedded in the URL. In version 2 this type of URL is still possible, as the code is still present to parse any items wrapped in curl braces '{' and '}', however, all observed URL strings omit the use of any of these parameters. Instead, the URL strings are all simple strings designed to look like regular internet traffic. The following is a list of the URL strings that we have observed:

/Work/new/index.php

/project/i.gif

/news/feed

/img/i.gif

/data/feeder

/rss/feed/stream

Information about the bot and the infected machine is still sent to the command and control server, but it is now encoded in the POST data and in a new *cookie* header field, as described in [2].

Both the encryption and the format of the configuration data received from the command and control server have been changed in Vawtrak version 2. When the client checks in with the command and control server it receives a blob of data containing a list of commands for the client to execute. Each command has a numeric value distinguishing the command, as well as an optional series of arguments that are passed to the command. This is the command used to store the configuration data in the registry (command 0x2), download and load modules (command 0x3), update without a reboot (command 0x1c) and carry out various other types of functionality. The format of the response data can be found in Appendix B.

When the received command is 0x2, the argument data will be the encoded configuration data that will be stored in the registry. This structure contains a number of encoded blocks that hold the different types of configuration data. It is compressed and encoded. We need to decode it using the same LCG algorithm used previously, and then decompress it with LZmat. The format of this decoded structure can be found in Appendix B. Each block includes a value that indicates what type of data this is. The configuration data types that have been observed are listed in *table 4*.

Config type value	Type of data
1	Web injects – URLs, page placeholders, flags and the injected code. Similar to the web injects from version 1
2	List of URLs, typically banking pages. Possibly these are trigger URLs that will start video recording.
3	List of URLs from which POST data will be collected
4	Modify URLs. URLs which will be blocked or redirected to other URLs, optionally based on a regular expression.
5	List of keyword strings. If browsed to pages contain these strings then the page source will be compressed and sent back to the command and control server.
6	<ip address>:<port> string, possibly used as a server address for the BackConnect module.

Table 4 – configuration data types

Each block is encoded in a slightly different way. They each share the same header structure followed by the data. The data is formatted differently for each block. Generally speaking, numeric, length and flag values are not encoded but any text data will be. The encoding scheme for text data involves generating a substitution box using the LCG algorithm and a seed value taken from the configuration data block (or a global seed value if the block value is zero), and substituting values from the encoded data for values from the S-box. A Python implementation of the substitution algorithm can be found in Appendix A.

Version 1 included a fallback command and control channel that involved downloading a list of new command and control servers from Tor hidden services, accessed through the *tor2web* proxy, steganographically encoded in an *ico* file, as described in [6]. The Tor hidden service addresses were included in the command and control address block that is embedded in the Vawtrak binary. Version 2 still includes the ability to retrieve fallback command and control server addresses through Tor2web. However, to date, no samples have been observed that include Tor hidden service addresses, so this functionality is currently dormant.

Modules and updates are all signed, as they were in version 1. They are verified using a public key embedded in the binary. An interesting fact is that this public key was the same for all version 1 samples, indicating that one entity is signing all of these files. In version 2 there is a new public key but once again it is the same for all version 2 samples. It is extremely unlikely that the corresponding private key would be shared amongst multiple groups of people, indicating that one entity maintains overall control of the Vawtrak botnet.

Current Targets

As with our previous paper on Vawtrak version 1, we can divide the larger botnet into sub-botnets based on the financial institutions and organizations being targeted in the configuration files. We find that targets can once again be grouped based on the geographic locations of the targeted entities. We have observed the following clusters:

1. **United States.**
2. **Canada**
3. **Japan**
4. **Romania and Israel**
5. **United Kingdom and Republic of Ireland**
6. **Czech Republic**

Compare this list with the targets identified in our previous paper:

1. **Germany and Poland**
2. **Japan**
3. **United States + others**
4. **Saudi Arabia, UAE, Malaysia, Portugal, Poland**
5. **United Kingdom**
6. **United Kingdom, Germany, Spain**

Although there are some similarities, we also see that there are new targets and some targets are now missing. The missing targets are an interesting issue. It might be possible that the Vawtrak criminals no longer have customers that are interested in victims from those countries. It might also be possible that those customers do still exist but we have not found related samples. Another possibility is that we have not been served the configuration files that target those countries during our analysis due to server-side checks such as a GeoIP lookup of the victim IP address. It is quite possible that configuration files that target those countries will only be served if the victim is also located in one of those countries.

Another interesting detail is that we have observed a small number of *project_id* values being re-used for multiple targets. This may indicate that the numeric values are being reassigned over time or it may indicate that the configuration file that is pushed to the infected machine depends on more factors than just the *project_id*, for example the client build version, URL string and domain name may all be contributing factors.

We will briefly describe the banks and other organizations being targeted in each case. A list of sample hashes and their associated cluster and project IDs, as well as a list of domains is included in Appendix C.

United States

This cluster had the largest number of inject targets and the largest number of *project ids* associated with it. The majority of the inject target URLs belong to banks and financial companies based in the US. However, there are a few notable exceptions such as online retail companies, telecommunications, a small number of UK and Canadian banks, and perhaps most interestingly, the online UK Government portal used for UK businesses to manage their VAT payments.

Table 5 lists all the URLs that have been targets for web injects that we have observed in this cluster.

URL
'!!!^sso\\.unionbank\\.com\\/unp\\/(SSO(Login|AceAuth)Servlet|passcode\\.jsp)(#)??$'
'!!^(?!.*=https).*\\/wcmfd\\/wcmpw\\/CustomerLogin.*$'
'!^express\\.53\\.com\\/portal\\/auth\\/login\\/Login'
'!www.paypal\\.com\\/.*cgi-bin\\/webscr.*(\\?|&)cmd=(%5f|_)(login-done|account|home)'
'(investor.vanguard.com\\/home\\/)|(investor.vanguard.com\\/my-account\\/log-on)'
'(jpmorganaccess\\.com|access\\.jpmorgan\\.com).*\\.js'
'(jpmorganaccess\\.com|access\\.jpmorgan\\.com\\/jpmalogon)'
'(pbi_pbi|PBI_PBI|ebc_ebc|EBC_EBC)1961'
'/business/j_security_check'
'/onlineserv/CM/'
'\\/bbw\\/cmserver\\/welcome\\/default\\/verify\\.cfm$'
'\\/pub\\/html\\/login\\.html$'
'\\/pub\\/js\\/jquery\\.js$'
'^(?!.*=https).*\\/wcmfd\\/wcmpw\\/CustomerLogin.*$'
'^(chaseonline|mfasa)\\.chase\\.com\\/((M|m)y(A|a)ccounts\\.aspx|auth\\/fcc\\/login|auth/auth-stoken-osl\\.html\\?.*auth_deviceCookie|(S|s)ecure\\/(O|o)(S|s)(L|l)\\.aspx)'
'^(cm|www)\\.netteller\\.com\\/.*\\/(A|a)uthentication\\/(V|v)iews\\/(R|r)sa(G|g)oId(A|a)uthentication\\.aspx'
'^(www.)??huntington.com(/)??$'
'^(www.)??lanb.com/access/login-ab.asp'
'^(www.)??signatureny\\.web-access\\.com\\/signat\\/cgi-bin\\/(welcome|login)\\.cgi'
'^(www.)??tdetreasury\\.tdbank\\.com\\/.*\\/logon\\/sbuser'
'^(www.)??treasury\\.pncbank\\.com\\/.*\\/login\\.ht'
'^(www\\.)??cashanalyzer\\.com\\/(cgi-bin\\/carsa.*\\.dll|caloadbalance\\.aspx)'
'^businessbanking.*\\.tdcommercialbanking\\.com\\/WBB\\/Login'
'^businessonline.huntington.com/BOLHome/BusinessOnlineLogin.aspx'
'^cashproonline.*\\.js$'
'^cashproonline\\.bankofamerica\\.com\\/.*\\/loginMain\\.faces'

'^cmo\\.cibc\\.com\\/wp\\/wps\\/portal\\/bbdsignon'
'^express\\.53\\.com\\/portal\\/auth\\/login\\/Login'
'^express\\.53\\.com\\/portal\\/versioned_content\\/.*\\/js\\/app\\.js'
'^online\\.(citibank|citi)\\.com\\/.*\\/(portal\\/((H|h)ome|(I|i)ndex)\\.do|ain\\/.*\\/flow\\.action|signon\\/(P|p)rocess(U|u)serna me(S|s)ignon\\.do|signon\\/(A|a)greement\\.do|signon\\/(C|c)heck(T|t)and(C|c)\\.do)'
'^online\\.americanexpress\\.com\\/myca\\/(accountsummary|acctmgmt)\\/us\\/(accounthome|myaccountsummary)'
'^online\\.citi\\.com\\/.*\\.js'
'^online\\.wellsfargo\\.com\\/das\\/(cgi-bin\\/session\\.cgi\\?screenid=SIGNON_PORTAL_PAUSE|channel\\/accountSummary)'
'^online\\.wellsfargo\\.com\\/das\\/cgi-bin\\/session\\.cgi\\?sessargs='
'^secure\\.bankofamerica\\.com\\/administer-accounts\\/manageDebitCard\\/displayCards\\.go'
'^securentrycorp\\..*\\.com\\/(A|a)uthentication\\/zbf\\/'
'^sso\\.unionbank\\.com\\/unp\\/(SSO(Login|AceAuth)Servlet|passcode\\.jsp)(#)??$'
'^www.amazon\\.(com|ca|de)\\/'
'^www.amazon\\.(com|ca|de)\\/.*\\/signin'
'^www8\\.comerica\\.com(\\/pkmslogin.form|\\/cma\\/portal\\/mybusinessconnect|$\|\\/$)'
'^www\\.(cibconline\\.)??cibc\\.com'
'^www\\.onlinebanking\\.pnc\\.com\\/alservlet\\/((V|v)erify(P|p)assword(S|s)ervlet|(M|m)y(A|a)ccounts(S|s)ervlet|(O|o)nline(B|b) anking(S|s)ervlet|(S|s)ignon(I|i)nit(S|s)ervlet|(S|s)ecurity(I|i)nformation(S|s)ervlet)'
'^www\\.paypal\\.com\\/(signin$\|myaccount\\/.*country_lang\\.x=true\|myaccount\\/home\|myaccount\\/$|.*webscr\\?cmd=(%5f| _)(login-done|account|home))'
'access.jpmorgan.com/jpmalogon'
'b2b.verizonwireless.com/sms/'
'bankline\\..*\\/CWSLogon\\/.*\\.do'
'blilk.com/Core/Authentication/MFAPassword.aspx'
'businessaccess.citibank.citigroup.com/cbusol/signon.do'
'businessonline.tdbank.com/CorporateBankingWeb/Core/CustomerService/ModifySecurityQuestions.aspx'
'businessonline.tdbank.com/CorporateBankingWeb/Core/InformationReporting/AccountPortfolio.aspx'
'client\\.schwab\\.com\\/(Accounts\\/Summary\\/Summary\\.aspx|secure\\/cc\\/accounts\\/summary)'
'db-direct\\.db\\.com\\/.*\\.serv'

'discoverbank.com/bankac/achome/summary?sa_status=1'
'discoverbank\\.com\\/bankac\\/achome\\/(summary|processachome)$'
'discovercard.com/cardmembersvcs/achome/homepage'
'discovercard.com/dfs/accounthome/summary'
'discovercard\\.com\\/cardmembersvcs\\/intercept\\/action\\/intercept(L|l)anding.*src=(%2Fcardmembersvcs%2Fachome%2Fhomepage|\\/cardmembersvcs\\/achome\\/homepage)'
'ebanking-services\\.com\\/(E|e)am(W|w)eb\\/(A|a)ccount\\/((R|r)emote(L|l)ogin(R|r)edirect|(P|p)assword(E|e)ntry|(L|l)ogin)\\.aspx'
'fps.fidelity.com/ftgw/Fps/Fidelity/RSAAnalyzeChallengeRetail/Maintain/Init'
'login.fidelity.com/ftgw/Fas/Fidelity/RtlCust/Login/Init'
'mblogin.verizonwireless.com/amserver/loginflow/main'
'myapps\\.paychex\\.com\\/'
'myapps\\.paychex\\.com\\/.*\\/userPassword\\.partial\\.html'
'myapps\\.paychex\\.com\\/.*\\/validateSecQuestion\\.partial\\.html'
'oltx.fidelity.com/ftgw/fbc/oftop/'
'online\\.hmrc\\.gov\\.uk\\/vat/(vat-variations|trader)\\/(\\d+)$'
'personal.vanguard.com/us/MPSecurity01'
'personal.vanguard.com/us/MPSecurity01?APP=PE&dbOnly=false&crossover=false&SelectedPlanId=095850&planSummaryMask=425886&CALLHANDLER=0'
'personal.vanguard.com/us/MyHome'
'retirementplans.vanguard.com/VGApp/pe/PublicHome'
'retirementplans.vanguard.com/VGApp/pe/faces/SHome.xhtml'
'secure.bankofamerica.com/login/edit/sm/redirectSecurityCenter.go?target=challengequestion'
'secure\\.(lloydsbank|bankofscotland)\\.co\\.uk\\/personal\\/a\\/logon\\/entermemorableinformation\\.jsp'
'secure\\.bankofamerica\\.com\\/myaccounts\\/(signin|brain)\\/(sign(I|i)n|redirect)\\.go\\?.*(return(S|s)iteIndicator=|target=accountsoverview)'
'silvmafo.net/citi/login.js'
'silvmafo.net/citi/main.js'
'silvmafo.net/jpmorgan/login.js'
'silvmafo.net/jpmorgan/main.js'

'silvmafo.net/wells/login.js'
'silvmafo.net/wells/main.js'
'singlepoint.usbank.com/cs70_banking/logon/sbuser'
'sso.verizonenterprise.com/amserver/sso/login.go'
'vesidm.verizonwireless.com/idm/secure/profile/main'
'wellsfargo.com'
'wellsoffice\\.wellsfargo\\.com'
'wellsoffice\\.wellsfargo\\.com\\/portal\\/signon\\/(index\\.jsp|failure|$)'
'www(1|2)\\.secure\\.hsbcnet\\.com\\/uims\\/portal\\/IDV_OTP_CHALLENGE'
'www.fidelity.com/'
'www.fidelity.com/login/accountposition'
'www\\.(nwolb|rbsdigital)\\.com.*login\\.aspx'
'www\\.(nwolb|rbsdigital|ulsterbankanytimebanking)\\.(com|co\\.uk).*login\\.aspx'

Table5 – Inject target URLs

Canada

This cluster predominantly includes URLs belonging to banks in Canada, though we have also observed some US and UK banks.

Table 6 lists the URLs that we have observed for this cluster.

URL
'.royalbank.com'
'acc.*desjardins\\.com'
'alterna.*\\.c'
'bankline\\..*\\/CWSLogon\\/.*\\.do'
'banquenet\\.td\\.com\\/.*login\\.htm'
'beaubear.ca'
'bmo.com'
'bvi.bnc.ca'
'caissepopclare.com'

'ccunl.c'
'cibc.com/'
'conexus.c'
'copperfin.c'
'credit.*\\.c'
'cu\\..*c'
'cua.c'
'direct.net'
'easyweb\\.td\\.com\\/.*banking'
'easyweb\\.td\\.com\\/.*login\\.htm'
'entegra.ca'
'financial.c'
'firstcalgary.com'
'gffg.com'
'hald-nor.on.ca'
'hscunl.ca'
'implicity.ca'
'memberone.ca'
'mtlehman.com'
'northsave.com'
'noventis.ca'
'omista.com'
'online\\.hmrc\\.gov\\.uk\\/vat/(vat-variations|trader)\\/(\\d+)$'
'prospera.ca'
'provincialemployees.com'
'reddyk.net'
'savings.c'
'scotiabank.com'

'secure\\.(lloydsbank|bankofscotland)\\.co\\.uk\\/personal\\/a\\/logon\\/entermemorableinformation\\.jsp'
'solutions.c'
'tandia.com'
'valleyfirst.com'
'vancity.com'
'wherewebank.com'
'www\\.(nwolb|rbsdigital)\\.com.*login\\.aspx'

Figure 6 – Target 2 inject URLs

Japan

This cluster focuses exclusively on URLs that belong to companies based in Japan. *Table 7* lists the URLs observed. This cluster has also been described in [7].

URL
'\\/SystemContents\\/CIBMZS01\\.js'
'\\/\\d{4}\\/B\\/js\\/KBA_Common\\.js'
'\\/docs\\/javascript\\/aaui\\.js'
'bk\\.juroku\\.co\\.jp\\/JRIK0\\d\\/cck\\/forms\\/IKP\\/JrBank\\.js'
'direct1\\.82bank\\.co\\.jp\\/HCIK0\\d\\/cck\\/forms\\/IKP\\/HcBank2_PC\\.js\\?.*'
'direct\\.chugin\\.co\\.jp\\/CGIK0\\d\\/cck\\/forms\\/IKP\\/CgBank\\.js'
'direct\\.ib\\.hirogin\\.co\\.jp\\/HRIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js\\?.*'
'direct\\.jabank\\.jp\\/ib\\/.*\\.do'
'direct\\.ryugin\\.co\\.jp\\/RKIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js\\?.*'
'direct\\d?\\.smbc\\.co\\.jp\\/aib\\/js\\/jquery\\-1\\.8\\.2\\.min\\.js'
'direct\\d?\\.smbc\\.co\\.jp\\/aib\\/js\\/mymenu_jquery\\.js'
'ib1\\.awabank\\.co\\.jp\\/AWIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js\\?.*'
'ib1\\.musashinobank\\.co\\.jp\\/MSIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js.*'
'ib1\\.yamagatabank\\.co\\.jp\\/YGIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK_PC\\.js\\?.*'
'ib\\.daishi\\-bank\\.co\\.jp\\/DSIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js.*'
'ib\\.hokkokubank\\.co\\.jp\\/HKIK0\\d\\/cck\\/forms\\/IKP\\/HkBank2_PC\\.js\\?.*'

'ib\\.resonabank\\.co\\.jp\\/IB\\/'
'ib\\.tsukubabank\\.co\\.jp\\/KTIK0\\d\\/cck\\/forms\\/IKP\\/KtBank\\.js.*'
'login\\.japannetbank\\.co\\.jp\\/wctx\\/.*\\.do'
'mib\\.miyagin\\.co\\.jp\\/MYIK0\\d\\/cck\\/forms\\/IKP\\/emusc_IK\\.js\\?.*'
'netbk.co.jp'
'parasol\\.anser\\.ne\\.jp\\/ib\\/.*\\.do'
'www11\\.ib\\.shinkin\\-ib\\.jp\\/webbk\\/pages\\/global\\/js\\/gcommon\\-\\d+\\.js'

Table 7 – Target 3 inject URLs

Romania and Israel

This cluster contains target URLs that we did not observe in our analysis of Vawtrak version 1. The websites targeted include social media and banks in Romania and Israel. It is not very common to observe banks in these countries being targeted with web injects by high profile banking malware, so this cluster demonstrates the diversity of the customers of the Vawtrak botnet. *Table 8* lists the URLs observed.

URL
'.google.'
'.live.com'
'facebook.com'
'hb2.bankleumi.co.il'
'login.bankhapoalim.co.il'
'start.telebank.co.il/LoginPages/Logon?multilang=he&t=P&pagekey=home&bank=d'
'start.telebank.co.il/LoginPages/LogonMarketing2?pagekey=home&multilang=he&t=p&bank=d'
'twitter.com'
'www.raiffeisen.ro'
'www.raiffeisenonline.ro/eBankingWeb/Controller'
'www.raiffeisenonline.ro/eBankingWeb/login'
'www.volksbankromania.ro/InternetBanking/SignIn'
'www.volksbankromania.ro/vbdirect/Login'
'yahoo.'

'youtube.'

Table 8 – Target 4 inject URLs

United Kingdom and Republic of Ireland

This cluster is very similar to the UK-focused cluster observed in our analysis of Vawtrak version 1. The targeted URLs are largely banks based in the UK. *Figure 9* lists the URLs observed in this cluster.

URL
'.*(webtrends|webtrendslive).com.*'
'.*\\.liveperson\\.net.*'
'^(ask|chat|smetrics)\\.barclays\\.co\\.uk.*'
'^(chat|online|www7)\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/.*'
'^(glass|room)\\.business\\.santander\\.co\\.uk.*'
'^(online|www|secure)\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/personal\\/(logon|a)\\/.*?$'
'^(online|www|secure)\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/personal\\/(unauth\\/|)assets\\/lib\\/jquery\\-min[0-9]{8}\\.js\\?[0-9]{8}$'
'^(online|www|secure)\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/personal\\/assets\\/lib\\/adrum\\-ext\\.e97e872f9a55953b65cb4029d2f76d20\\.js\\?[0-9]{8}$'
'^(online|www|secure)\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/personal\\/assets\\/lib\\/adrum\\.js\\?[0-9]{8}$'
'^(online|www|secure)\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/personal\\/static\\/(desktop\\/|)scriptsnippet\\.jspf\\?[0-9]{8}$'
'^(online|www|secure)\\.(tsb|lloydsbank|halifax-online|bankofscotland)\\.co\\.uk/personal.*'
'^(online|www|secure)\\.(tsb|lloydsbank|halifax-online|bankofscotland)\\.co\\.uk/personal/assets/lib/adrum\\.js'
'^(press|fc1)\\.retail\\.santander\\.co\\.uk.*'
'^(rbs|cdn)\\.tt\\.omtrdc\\.net.*'
'^(retail|business)\\.santander\\.co\\.uk.*'
'^(roll|b8k|road).*nationwide.'
'^(www3|nwb)\\.bankline\\.(rbs|natwest|ulsterbank)\\.(com|co\\.uk)/.*'
'^(www|login\\.myproducts|banking)\\.tescobank\\.com.*'
'^assets\\.adobedtm\\.com.*'
'^bank\\.barclays\\.co\\.uk.*(SeeStatements|EStatementHistoryOpen).*'

'^bank\\.barclays\\.co\\.uk/.*\\.action.*'
'^bank\\.barclays\\.co\\.uk/olb/auth/LoginStep1(Without|With)AssistCookie_display(|PreloadedStep1)\\.action.*'
'^bank\\.barclays\\.co\\.uk/olb/payments/(TransferMoneyStep1|RegularPayBillStep1|PayBillStep1)\\.action.*'
'^bank\\.barclays\\.co\\.uk/olb/payments/(ViewManageBipsAndChapsPayments|DeletePayeesStep1|ChangePayBillAndTransferStep1|StandingOrderTransferStep1|BIPSPaymentsOverview|CHAPSStep1GettingStarted)\\.action.*'
'^bank\\.barclays\\.co\\.uk/olb/payments/PaymentStatusStep1\\.action.*'
'^bank\\.barclays\\.co\\.uk/olb/payments/PaymentStatus\\.action.*'
'^barclaysbankplc\\.tt\\.omtrdc\\.net.*'
'^check2\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/fp/check\\.js\\?org_id=.*'
'^check2\\.(tsb|lloydsbank|halifax-online|bankofscotland)\\.co\\.uk/fp/check\\.js.*'
'^marketing\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/(lloydsimages|halifaximages|tsbimages|bankofscotlandimages)[0-9]{2}\\/[0-9a-zA-Z]{4}\\.js$'
'^marketing\\.(tsb|lloydsbank|halifax-online|bankofscotland)\\.co\\.uk/(lloydsimages|tsbimages|halifaximages|bosimages)[0-9]{2}/[a-zA-Z0-9]{4}\\.js$'
'^promotions\\.(lloydsbank|halifax\\-online|tsb|bankofscotland)\\.co\\.uk\\/scripts\\/lloyds\\/sloth_inc\\.js\\?[0-9]{8}$'
'^statse\\.webtrendslive\\.com\\/dcs[0-9a-z_]*\\/wtid\\.js\\?callback=Webtrends\\.dcss\\.dcsobj_0\\.dcsGetIdCallback$'
'^www\\.(nwolb|rbsdigital|ulsterbankanytimebanking)\\.(com|co\\.uk)/.*\\.asp.*'
'^www\\.(nwolb|rbsdigital|ulsterbankanytimebanking)\\.(com|co\\.uk)/Brands/clickerjs\\.aspx.*'
'^www\\.(nwolb|rbsdigital|ulsterbankanytimebanking)\\.(com|co\\.uk)/Brands/jq_scripts/CreatePayment\\.js.*'
'^www\\.(nwolb|rbsdigital|ulsterbankanytimebanking)\\.(com|co\\.uk)/ServiceManagement/(TealeafSDK|TealeafSDKConfig)\\.j.*'
'^www\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/.*\\.asp.*'
'^www\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/OnlineEnrolmentRegister\\.asp.*'
'^www\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/ScriptCombiner\\.axd.*'
'^www\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/default\\.asp.*'
'^www\\.(nwolb|ulsterbankanytimebanking|rbsdigital)\\.(com|co\\.uk)/login\\.asp.*'
'^www\\.bankline\\.(rbs|natwest|ulsterbank)\\.(com|co\\.uk)/CWSLogon/.*'
'^www\\.bankline\\.(rbs|natwest|ulsterbank)\\.(com|co\\.uk)/bankline/.*'
'^www\\.paypal\\.co.*'
'^www\\.rbsdigital\\.com/.*\\.asp.*'

'^www\\.splash-screen\\.net.*'

Figure 9 – target 5 inject URLs

Czech Republic

The final cluster we observed focuses entirely on URLs that are located in the Czech Republic address space. The web injects for this cluster have previously been observed being deployed by Vawtrak version 1. *Table 10* lists the targeted URLs.

URL
'(login.szn.cz)|(www.seznam.cz\\/?$)'
'^email.seznam.cz'
'cz.unicreditbanking.net\\/disp.*'
'cz.unicreditbanking.net\\/infinity\\/uifw\\/http\\/javascript\\/gws-closure-man_v.*\\..*'
'ibs.internetbanka.cz\\/.*\\/ControllerServlet;.*'
'ibs.rb.cz/IB/ControllerServlet'
'klient.*.rb.cz\\/ebts\\/.*ProductInformation.*'
'klient.*.rb.cz\\/ebts\\/version.*\\/cz\\/JSCode\\/loginpage\\..*'
'klient.*.rb.cz\\/ebts\\/version.*\\/cz\\/Login.*\\.html'
'muj.erasvet.cz'
'muj.erasvet.cz/klient/vitejte-kliente'
'muj.erasvet.cz/prihlaseni'
'sign.mojebanka.cz/cexiLogin.html'
'www.business24.cz/ebanking-b24/ib/base/usr/aut/login?execution='
'www.fio.cz/scgi-bin/hermes/dz-internetbanking.cgi'
'www.fio.cz/scgi-bin/hermes/dz-zustatky.cgi'
'www.servis24.cz/ebanking-s24/ib/base/inf/productlist/home?execution='
'www.servis24.cz/ebanking-s24/ib/base/usr/aut/login?execution='

Table 10 – cluster 6 inject targets

Sinkhole Data

As an interesting addition to the data and analysis of Vawtrak version 2 that we have produced in this paper, we also present data collected from a sinkhole of a command and control server used by Vawtrak version 1. This data is very useful for filling in some of the blanks that cannot be obtained from looking at samples in isolation.

A total of 9,594 unique (based on the bot ID value) bots were observed while the sinkhole was running. The sinkhole was operational from the end of 2014 through the early part of 2015. The largest number of bots online at any one point was 2,334 on 22 October 2014. Since only one of the command and control servers was sinkholed, this represents only a fraction of the whole botnet.

The first data point describes the number of bots observed hitting the sinkhole, the *project IDs* and the geographical breakdown based on a GeoIP lookup of the client IP address. *Table 11* shows that the largest numbers of bots are associated with *project ID* 0x9c and the majority of victims are located in Saudi Arabia (SA), United Araba Emirates (AE) and Malaysia (MY). This corresponds to the fourth cluster we identified in our previous publication and is one of the notable clusters that we do not have data for as being a target for Vawtrak version 2.

Project ID	Bots	Top 3 targets (in percent)
0x9c	3051	SA (55) AE (19) MY (15)
0x2a4	2339	US (73) AU (9) CA (6)
0x6e	1439	US (97)
0xb6	1433	DE (97)
0xc1	382	DE (94) GB (4)
0xac	299	DE (22) CH (19) CA (10)
0x98	167	DE (71) SA (20) JP (2)
0xa2	70	DE (94)
0x6a	68	CA (93) US (7)
0x6d	67	US (94)
0x258	61	TH (38) VN (16) KR (5)
0x7c	28	DE (86) US (7) SA (4)
0x86	16	PL (100)

0xb7	1	UK (100)

Table 11 – geographic breakdown and bot numbers per project ID from sinkhole data

We can also observe some of the other clusters we identified, including the United States, Canada and Germany. We also notice one cluster that we have not previously observed – *project ID* 0x258 which has the majority of victims located in the Far East with client IP addresses mostly appearing to originate from Thailand, Vietnam and South Korea. We were able to locate a sample belonging to this *project ID* but we were not able to find any configuration files or web injects that would have been delivered. This shows the value of sinkhole data when attempting to understand the full scale of a threat. Our previous ignorance of this cluster may have been due to a variety of reasons such as country-based filtering on the Vawtrak command and control servers or simply low representation in these countries amongst Sophos' customer base, but these factors are eliminated when we can observe all incoming traffic to the server.

Figure 7 shows the numbers of unique bots seen per country. This data shows that when counting bot numbers there are more in the United States than anywhere else, followed by Germany and Saudi Arabia.

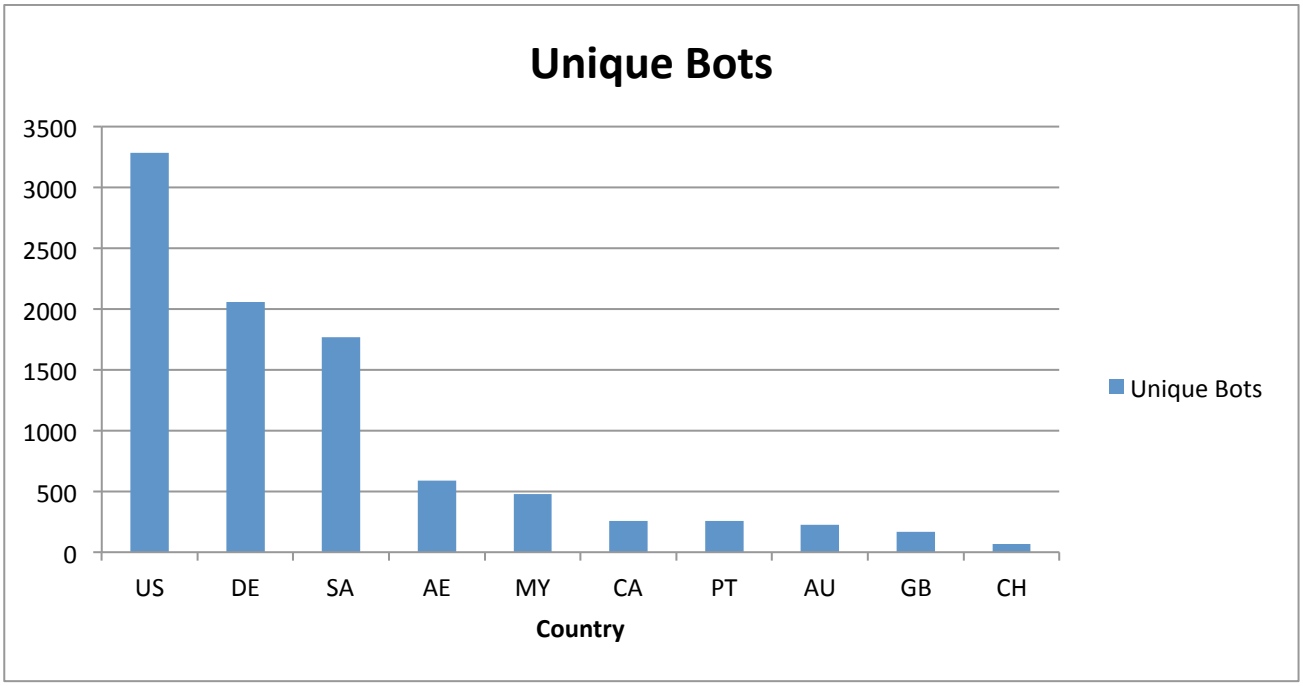


Figure 7 – Unique bots per country

By examining the *User-Agent* field of incoming requests we were also able to produce data on the browsers and OS of victim machines. This data is interesting as it strongly indicates that large numbers of infected machines are running older hardware, on older OS versions. *Figure 8* shows the breakdown of bots by the OS version.

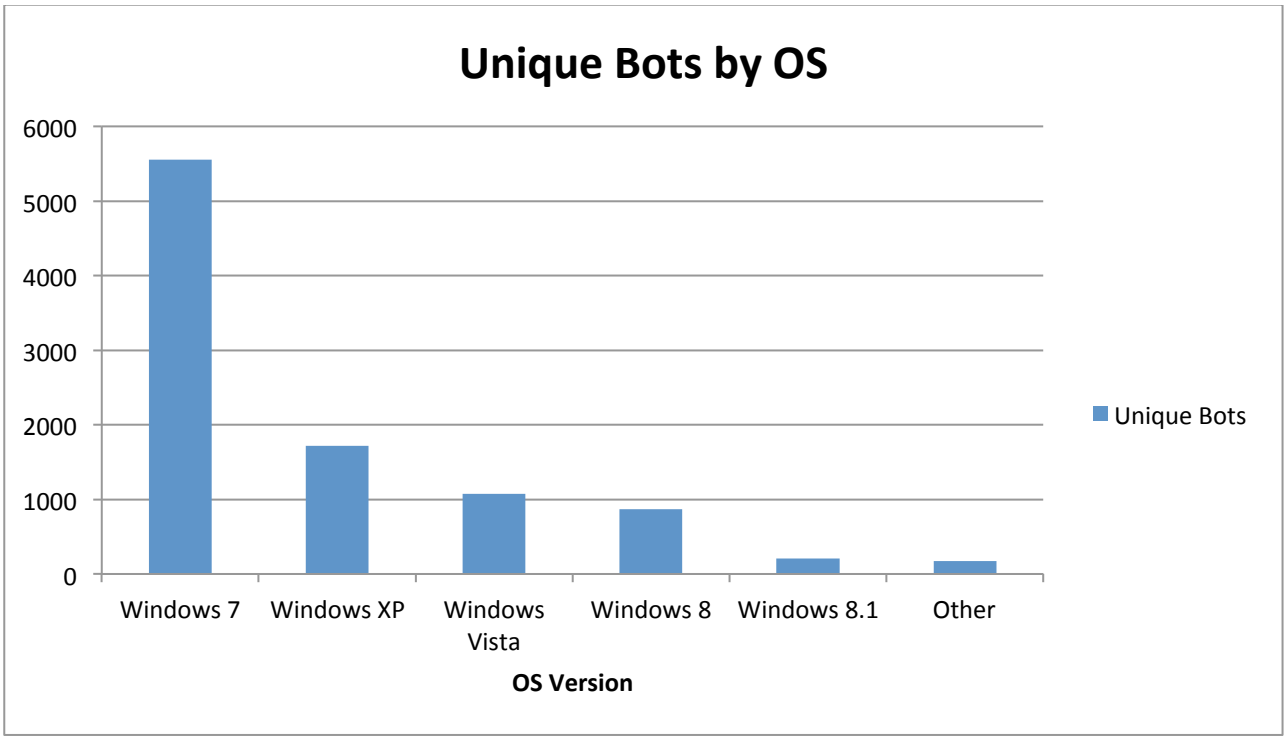


Figure 8 – Unique bots by OS version

Although more than half of the victims are using Windows 7, there is a significant amount found to be using Windows XP, support for which had been discontinued by Microsoft more than six months before the sinkhole was operational. There is a high chance that unpatched vulnerabilities exist on these machines which would make them easy targets for infection through Exploit Kits that exploit patched vulnerabilities, which is one of the infection vectors historically identified for Vawtrak.

Figure 9 describes the architecture breakdown of the victim machines. The data shows that we observed twice as many victims running under x86 as we did running an x64 version of the OS.

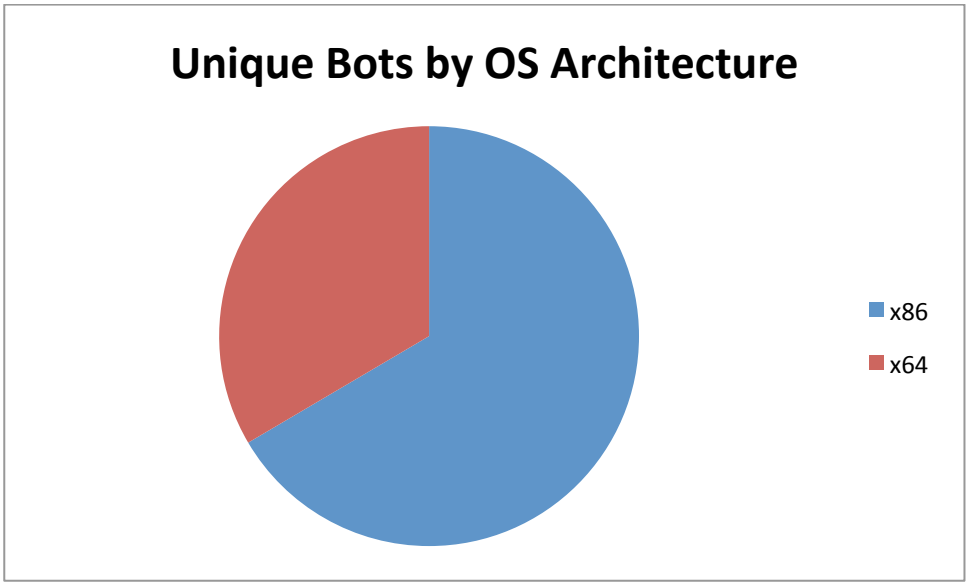


Figure 9 – unique bots by OS architecture

This is an indication that the hardware being used by the majority of victims is older than perhaps we might expect when considering all users of the internet. It would be rare that a new machine purchased over the last several years would not have an x64 processor and a corresponding version of the OS installed.

The final graph, *figure 10*, shows the breakdown of browsers observed hitting the sinkhole.

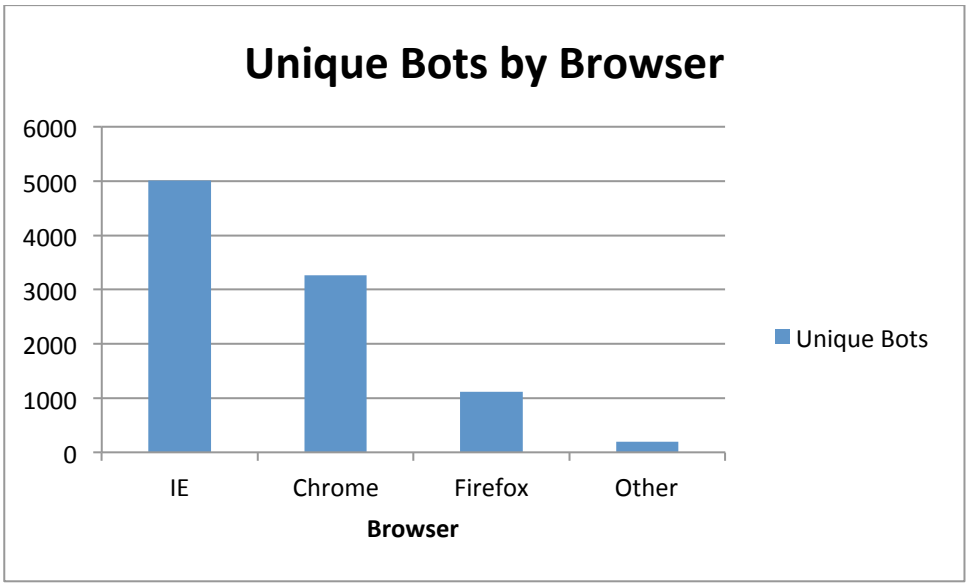


Figure 10 – unique bots by browser

The most common browser observed is Internet Explorer, followed by Chrome. According to most online sources (for example http://www.w3schools.com/browsers/browsers_stats.asp) Chrome is by far the most popular browser on the internet, so it is surprising that we see Internet Explorer coming out on top. This may be another indicator that many victims are using outdated versions of the software as we may tentatively suggest that people who are not updating their software with critical security updates may also be unlikely to change their browser settings from the default supplied by the OS.

Conclusion

The new version of Vawtrak shows that the botnet is very much alive, with active developers and a thriving customer base.

The pace with which new build versions are introduced shows that product releases are happening frequently. New command and control addresses are being observed on a regular basis which shows the botnet administrators have no problems acquiring new infrastructure. We have observed new banks in new geographic regions introduced as targets for web page code injection which indicates that new customers have been brought onboard.

The authors have improved the code base by moving to a modular architecture and increasing the level of obfuscation to hinder analysis. Modularization also opens the door for a market in custom modules if an SDK is released by the authors, though we have no evidence to suggest this is planned at the current time.

Appendix

Appendix A – Python Code Snippets

IDA Python script for string deobfuscation

```
'''
Vawtrakv2 string decryption script.
Place the cursor at the start of the string decryption routine,
then execute the script.
All cross-references to the decryption routine
will be enumerated, the strings decrypted and
the cross-references highlighted.
'''


def decrypt_string(data, seed):
    count = 0
    dataOut = bytearray(len(data))
    while count < len(data):
        next_val = (seed * 0x41c64e6d) & 0xffffffff
        next_val = (next_val + 0x3039) & 0xffffffff
        seed = next_val
        outb = ((data[count] & 0xff) - (seed & 0xff)) & 0xff
        dataOut[count] = outb
        count += 1
    return dataOut

# Place the cursor at the start of the string decryption routine
print "-----Vawtrak2 string decrypt start-----"
address = ScreenEA()
refs = XrefsTo(address)
no_pushes = []
```

```
done_refs = []
for ref in refs:
    prev_ins_address = ref.frm - 5
    prev_ins = Byte(prev_ins_address)
    if prev_ins == 0x68:
        string_offset = ref.frm - 4
    else:
        prev_ins_address -= 2
        prev_ins = Byte(prev_ins_address)
        if prev_ins != 0x68:
            #TODO: handle cases where the string address is set
            # differently than a simple push
            print "ERROR: did not get push before xref at: %x" % ref.frm
            no_pushes.append(ref.frm)
            continue
        string_offset = prev_ins_address + 1
    string_address = Dword(string_offset)
    if string_address in done_refs:
        continue
    seed = Dword(string_address)
    size = (seed ^ Dword(string_address+4)) >> 0x10
    print "size: %x" % size
    if size == 0:
        print "Skipping size 0 at: %x" % string_address
        continue
    count = 0
    offset = string_address + 8
    dataIn = bytearray(size)
    while count < size:
        dataIn[count] = Byte(offset+count)
        count += 1
    decData = decrypt_string(dataIn, seed)
```

```
    print "decData at %x: %s" % (string_address, str(decData))
    count = 0
    while count < len(decData):
        PatchByte(string_address+8+count, decData[count])
        count += 1
    MakeStr(string_address+8, string_address+7+len(decData))
    add_dref(prev_ins_address, string_address+8, dr_T)
    actual_string = GetString(string_address+8)
    comment = hex(string_address+8)
    comment += ": " + actual_string
    MakeComm(prev_ins_address, comment)
    done_refs.append(string_address)

# print out the locations where the decrpt routine was called but
# we failed to work out what the string address was.
print "Fails:"
for a in no_pushes:
    print "%x" % a

print "-----Vawtrak2 string decrypt end-----"
```

Substitution algorithm for config data blocks implementation

```
def gen_sbox(seed):
    mul = 0x41c64e6d
    add = 0x3039
    dataOut = bytearray(0x100)
    tmp = bytearray(0x100)
    index = 0
    while index < len(tmp):
        tmp[index] = index
        index += 1
```

```
    index = 0

    while index < len(tmp):
        x = tmp[index]
        next = (seed * mul) & 0xffffffff
        next = (next + add) & 0xffffffff
        seed = next
        swap_index = seed & 0xff
        tmp[index] = tmp[swap_index]
        tmp[swap_index] = x
        index += 1
    index = 0
    while index < len(dataOut):
        targ_ind = tmp[index]
        dataOut[targ_ind] = index
        index += 1

    return dataOut

def do_sub(dataIn, sbox):
    dataOut = bytearray(len(dataIn))
    index = 0
    while index < len(dataIn):
        b = ord(dataIn[index])
        outb = sbox[b]
        dataOut[index] = outb
        index += 1

    return dataOut
```

Appendix B – Data Structures

Structure of C & C response data

Offset	Field
0x0	Size of whole data
0x4	Number of command blocks
0x5	Command number (second byte = index into command handler table, first byte = unknown)
0x7	Number of arguments
0x8	Size of first argument
0xc	Data for first argument
0xc + size of first argument	Length of second argument
…	

Format of configuration data structure (argument to command 0x02)

Offset	Field
0x0	Number of arguments (expected to be 1)
0x1	Argument length
0x5	(Data from here is initially encrypted with LCG and compressed with LZmat) Size of entire config data
0x9	Number of config blocks
0xb	Global seed (use if block seed value = 0)
0xf **(Block Start + 0x0)**	(Config blocks start here, each has a header followed by data)

	Size of block
Block start + 0x4	Block seed value
Block start + 0x8	Block ID
Block start + 0xa	Number of items in block
Block start + 0xc	Start of block config items

Appendix C – Hashes and Domains

Domains

atlasbeta[.]com

basislabel[.]com

beatdiner[.]com

begiekee[.]com

beifamen[.]net

bezeacooni[.]com

brokerbox[.]net

camelcap[.]com

careerplaza[.]net

castuning[.]ru

ceazivie[.]net

ceteixoo[.]com

cherrystore[.]net

cieku[.]net

circlewear[.]net

ciroxi[.]com

codexbase[.]com

cookitie[.]net

cooxieneifea[.]com

dadry[.]com

deehiesei[.]com

deiceezo[.]net
desertcast[.]com
dietoog[.]com
easysecure[.]net
eurofeeke[.]com
finehotels[.]net
fotologes[.]su
futooke[.]com
geecamer[.]com
geeseazei[.]net
geimal[.]net
giegux[.]net
guesstrade[.]com
heagiex[.]com
heapeih[.]net
heepoom[.]net
helloalliance[.]net
hiesook[.]net
hooxoovie[.]com
hybridtrend[.]com
ideagreens[.]com
infototal[.]net
investweek[.]net
kacac[.]com
keanees[.]com
kikuveid[.]com
kooxuse[.]com
korieg[.]net
labadinom[.]com
linemus[.]com
lokagbuuses[.]com
mafeepaseiz[.]com

mafoovoo[.]com
maziepug[.]com
meceexu[.]com
mefealo[.]net
megeer[.]net
memiepi[.]com
mgsmedia[.]ru
needeakor[.]net
neezood[.]com
negut[.]net
neimucher[.]net
neinoove[.]net
niheeree[.]net
nikeifi[.]com
ninthclub[.]com
nordijors[.]com
nusivee[.]net
omasm[.]com
ovead[.]com
pausephone[.]com
peazor[.]com
peisaho[.]net
peruzes[.]net
plazaforc[.]su
pooxete[.]net
poxeeki[.]net
puropea[.]com
rainbowfarm[.]net
raveiv[.]com
raveiver[.]com
reecudoo[.]net
riegeevoo[.]com

seaboy[.]net
searalihid[.]com
seevu[.]net
seviv[.]net
siloovoox[.]net
skyflex[.]net
skynavigation[.]net
teaseeras[.]com
teezeapeesei[.]net
teeziekeih[.]net
terearizoo[.]com
textidea[.]com
tibie[.]net
tobeacea[.]com
tozosei[.]net
trillionstudio[.]com
tugeivi[.]com
veiginip[.]net
veipapeitee[.]net
vintageselects[.]com
vmark[.]su
volexuc[.]com
wearcafe[.]com
wildclick[.]net
xeaberal[.]com
xeixol[.]com
xogeikei[.]net
xphotos[.]su
zazoogo[.]net
zeediefan[.]net
zeetea[.]net
zeileagei[.]net

ziefisea[.]net

zofienie[.]com

zooreizei[.]net

Hashes (sha1)

Cluster 1 – United States

Project ID 0x10:

057163181423535bc7765bbaadc086f4eb6516e7,05a185bdef3df78257544f4904f0ef2e08f4d861,06661c72cb7d5c6f647e1d46a068ef69aab9ae40,09a1d8cfea3875ca5c00fc67842b057246305b46,12177c4e110efe08e2dc06ecdfa3b2445dd52257,15bfa9d6ffd93c4ba3b4ca4120a1da6abea80cf1,1f20747ba91363285d071c65d79a5cef34ff5490,2699eb86d59520aa4ff7ec85e51b4793e74f608a,278d357b33af3ef0d559c69945e47a2702bef5ec,27be419981d9ef852587afd7b4fbcc1ff445fcfa,289a8cce7b9ca6f124112ae807e48629d52e132f,30cfa7703cd0aa2d217cbb676d63b5a5fc00dc0d,3248a1593686a9b4d2ff03277295e0c25f79149a,33dfe4b8b6746502df1d0465d41ef2833d0604bf,3c19310f43f84f1a3947bb26ea6b9edb14a09c65,448a94dcba6f8d8ce470980dfc4c6b6689438846,476ec2208b2eba41348c741003855b684d3caa01,5d65d50c986f19f1fe3e74a133cb0493ba202766,678dc7fb4ad438ecd30e6dc96a884bd148c98a3a,7737055a88138d74cc6d9f9ba586f8dd42402097,7ce5d09074f9bb512a592dc9638cf7967d4dacc9,897a9d83504b97a294edd7e94f2d1084f08f85c7,898583db39ef9bad779fe0b7fe8ede1596625c05,9276db4ff107353bfb564f9e27c6a57eeb980aa6,93af6463d368dc0b7d408485fff99dfb97c74989,bec7722fd139df15ca7c00534771647bffd41a4a,d7047ac974092c6c0c998c90235c852f6dc248e1,f6c22b345b00f208d4eb1da196d57a1a9a1a318d

Project ID 0x11:

52c173c8da4c250a32298928e6c061f613cf2a48

Project ID 0x12:

049243802c1d245a34be1306f4128b84177ee6dd,0bf958337b5df4562f79ff3884f02a3c23eb5311,16ceaac53eb24d578f46b3be201acd8d0c7e4507,2be7ff6ebc0400da40585390aacd7bad977d8662,2e141ecc495e7aa40a1062749e712b0748d68ed6,31e0238d36fa2f64fa65628e25c0541723b0de2d,3deb2d0b711b224e31ff23cf84ebd0c266bc6f93,5f8685afc0998d2894fe23695d8fd1393f6e112d,a4ad52b17edd1dc831c567079caebc099afd3a94,d6162dd7a7bd179af7c79b1503005d61a3abe982,e53cf85028be504c8b050f3feeaf42843b5d0660

Project ID 0x19:

0d4004b50eb3a92550598d806597293a11d418b8,290c5390a20e1b2462ef5bc57caf3a71a2187f15,2b766f115f946a1baf1b9549feb8f2c17fa1efcb,2d4aa920e23b6e48119e8173e46765c1f4a9a0c6,324b3555feea17e8752bb45bebb5a991a94ba4a6,5fa93893516e6d6473b20d486c0b299baed2c602

Project ID 0x1e:

197bdc784fedcbba027c01936ac08dddee32b1ba,30e0db94b63c9ff1cf5063cd8b6c915701ab564c,38568d930aa0c7488ed3bfef71887f93f658d90b

Project ID 0x4:

5cb47cfee3001941e08686905059525b8fbdadf3

Project ID 0x6:

135f100fbdf51ac04c3ae30a5a584c14630e38af,2e2c75613fb562e58871a6001546d5ab5ae133f4,2e7fab79107b8d9eaf2a2fa751afde874ddee5fc,66ebf8285d40cf3ed7bedbe0bd31058f4531fe59

Project ID 0x9:

1b0ae5b7a96d04fed68c84f482705610786e0318,5ebc552a83441af3471505aa4d062d7af7bafecd,670d38b3b28cb288144876428de774e606783142

Project ID 0xa:

00aaed24eab73ae20f0c1ba5c05b76f277519441,639c755fc5f8262ed77d29c7b5cf0f2e8cfbf7b1

Project ID 0xb:

0d795dd6e25ed4c828996d4fc141f413420f2269

Project ID 0xc:

017c054c98eb3ba7eab47a947882296ba71c00ec,0e86e87461780df79e1fdd6b63383e97bf023d84,1315cc4013ff1fdd71eff91c3bf6612e2a034d08,2d310d8f284afa782b4c9c66c7ccd69a31d09734,604b955918e003fa8bd2502d859cd0e6b6707473

Project ID 0xd:

3e4b72babb4ef6906678f9d34d7f068538d0e9b3,3ffac95b825a63a89bc979f10fc3d754caa5e50c

Project ID 0xe:

09fd43c246953de68e65d582d5dc313d0a00f44c,142cbafbfc3800aef06b738fb164cfc81a97d7c5,2efbfa4b2e188015440cc5340c1c65f234f2eba1,4cf7c94ab0c5924e8550e59c6cb267eae1bcd55f,bb76625a40b6b9ea97e2229e4aba63e08700cef3,df0dab926b5683c88d42f6f92fdd4c340c1e7708,e390a8871183ea7c203b398ade86b57dd89cd76e,fa2cc987fae9e2db980be40ead3ebc4d724bd080

Project ID 0xf:

0145034ce96bbe7f4098755ad8b7a69cb074be1a,06038015cd6286b8c2cfbe87803ba6dee8a975d8,0922f15f512e2eb22ac1eb0c68ceba59b7e9da09,0af9f4e6bd7063603d1f2834bce5e6739e9da2e9,177e5435b0b5dcf06dd5efb86c81d9e434ccd5e5,19802971032c66b1b7f752bb090287ac437b8ae4,1ab7f8fe86a095b985c2ca1c3a5696bbc430f07d,1f5cd19323f28e9de46b61612ded34b5858d4376,36d6e159afd975f99f183ec1f749b741d9edafb4,47aa1f7891262fed27b5360eb1fad16be9dc4707,542acefc7ac9a22363247714edde93ff41ceba36,63353063bbdabe8cccffcf8955528665fc744acd,864af59dc531244906226f2bf038c245253b8c29

Cluster 2 – Canada

Project ID 0x7:

24313934ea3040f20a8c392990e31b224cd08610,290c1fc7b06c85e484bda99066601b3257d0cda3,4276b4c7d648cac75ad8c7407d7eed52d31ab130,5420a8568ced7ef167c8078f99aed44ce458f62d,6625a7312a9b3d845dbb42ffc88cdc3a926336b6

Cluster 3 – Japan

Project ID 0x1c:

11661d69d3882bc409f6a66169d40c74553d0db4,bbfdf61b24d5938779a5cc45ccd765dc37b6a4f4

Project ID 0x8:

02445099bc9b5e19ede7b65c9f9230509e4fc546,0753c0a68e91ab56e9e3e2106360482db28063f1,0a97175634fada5775840136e2a21013958038b2,3a24d81e9d4b0b2e02947472a5c36db16285d597,4146f24eb23cecd09f7907f398bcfb7f28c7b47d,4f496b70016dba95c6441b8ea816a0ab53421445,60b83e916484600fc87e3783000ea444fc0c83fe,63c88467a0f67e2f3125fd7d3d15cad0b213a5cb,c93735e655502a35046f45f59ce7875211ff7cb9

Cluster 4 – Romania and Israel

Project ID 0x14:

0c7c54f8df44e78dea158e90a6ae2a2ec6ae8ee0,0f34a4b6a0f4bdf60edc96ff8909d79bf14f3659,3b42948c565ced2aac0729a1c91503ca626ef0fc,49411ab8e2029ba5153617570433e227475d0b6a,5f1bcc9c18ba4c19957a309b5e91ea37c4779f2b,648a968683d1567eedd3f0a18c16e297e945e377

Project ID 0x15:

40bc93851f09d407d83d80b490185081f7d77a60

Cluster 5 – United Kingdom and Republic of Ireland

Project ID 0x3c:

009c8fa3d19fe2562b200204c0023e570a261998,017860c04b1f0117ed5da3abd2eb8bb077eee4a4,01fec46295471775d78d4d34a8790c6f59b70fb9,050520636ebfdade85dc5a7670f607d658f575d0,0909deff61c111ac35361caa1d6b1874e16145d2,0f79fecdeacd8f3d5102f261368be879530501fb,0fc542e564c30401ad438ffb0ce81c5cb93ff82c,1b8b6e9e943420e75d90d6880271d5fe115a6ca6,1e43cf92a343bcc5b7f6a191ec5f3f4aa82ae9d0,20436e67a7e4e7d2ff52db0b1c202755a2fdc08e,20bab1698973fa44c9efb31b17a85142cf3330e9,223db32d81eb0409fc823c945d77e98fe6a8828f,23c0e41efa6292dc230d3465e008f661b32c15eb,28f7762aedfa0f4bc0011d42232ea4baadf2d639,2d74207f52447672a3b02622c6557921404684e2,3064a214ffaf74a1f905e63b2cd3da8c4485ef83,30bc84c2ce721f1339593c1a39c4145328946e58,31bb5ce90bb3577255522d8327fb01faf5760c3a,31c490c555c2695b4711d1fa4e29d58f88ff3d20,31e403448e997cd9543e4e3db401baa86d9c1d1b,330eaf4bfe21d2242eef2ede8ef3d872490ce757,3ac11b7e86d6f6915435b02c38b30e5782cd9eb5,3e508d6868ab97044c0883abd78c2ea8f2f7536f,400ca050195a425e04eec9cba88b26b9c2401800,438acb5278b981dab1fd88edcbaf2576b4685c5d,43acda6cf48ea53f3ceaecc59ee06199a0391e76,4468f036a96ddf73d1a02342afaf8e7bf6962f7b,45de46f4e9474df546b5640a8b1c356ee906aa7d,4ebad5c139b3b19a433b809c2462d7d51080e676,5191a567eb8810481ac327a6c852ba059500da8d,522bc90ee2430f2c44d426bb4c37afdcf4b581a7,5427d2e72d1f3d435e3d8cf971e82092b33a91a2,56962d200a74fa57e89ba6d9e5978be09037fe4c,59f701ac439d31ccfda517603d544fcf44f9adab,5b541bf4aec06ff47e3645c856261d8c40a6ee14,62102f58382425d01472d9f5967becb9632a7645,63fd43abfa8d69f4a7f5f0d98b59c1e135b44001,6439cce9c76913effb02b8acb867eecaef984a0e

Project ID 0x3d:

1cbe2a4f3d12f28bba061108d2e0bda9410dd9a0,2459544d5c837606d552c2539f36c420e7c0637a,4b8d
da4e92566916188ef68f7d38072eb058b4f9,4f5f86a1d24b7ba6841a73480195503725e5d257,5994ab2d
81f64b973d7b3927cc0e7e9243983225

Project ID 0x3f:

2aecb522fce01ae5a7cd53d98de59d17368cb55e,506bb1caa5a920838b120a9888b3ccedc7124189,5bcd
459b1950731fac7bc762e6dcdb1c0096188a,5bd14a0c0b2e496d240984a0a7839357dca2d38e

Project ID 0xb:

019caf4a823c9733bd114ca76e1e2fd3d1682262,0f2883826c767f8afaf83713b90aa7d2dbe1ad6e,158e1c
cae9734a3d281e2f10bbbe68187180745f,17465b8bda9bca7f7b7159d6f1244a27b925e9fe,1abb91b8343
3a1649c9ce9560fac5d6399271c7d,22c66b8eeb60ce3c1e9d039825dd94766299bcd0,2bab3c9a85231d4
c400749d1128cffa397b974a1,2d7aee3112c1ed88a55ce3afd2d35fe6442454a0,3adcec66e613d9135936
48c9d13e6451b566f201,3ba19614b7d9951d77f166e4ed4f8c2bf4e8233e,402c0e237680550746d359ea
7925b3b0530950c4,5058c3c32f92d0cd5d67e64d5fdb9ddab92a41f3,53651ccabb870a5d8f9ca6af3cb58c
db191ecb52,56944e2d39e0f6dfa28c996d70200426f213cf45,62435083cf5d66c0196328dae58a950b4f1
859d6

Project ID 0xc:

031d0ee0fe754d5745cd966531fd423ef9295407,04c1dda42cec4bf38376e4edc4269a8badef27fc,076d51
f4ebff3b99d69e7996cec357d8dbc40f3f,08dcdd1429f2ffb97d79e3e3832b474738f8035d,0f61af7196253
494af6b2c037badfe74a739dabc,18023b1ffcee9e579f1c1bbae9b874d1e64f0462,18a16d6933d5b5c61f3
41b987d4684126223bdb1,1d377775370727054fa33f6af402cc67a9b5e83d,20f557743bc6d2499cc78763
c8906a63a1fca6aa,237655597c18422e6eab2b1535834dbd07dbbdec,265bffcbd83f025b5b957384dd953
0ae6752ff82,2b7037ee7f88241af5e9de0e69237062406686d3,37c7e2b4617f1e1e2818d55b433c8d5697
5c4f38,3c42630869b9a29bff154011d14f66345a020041,3eaa25a68b78c4984811403025f245b2b391225
7,479671f2c1c06a3167eb7b90697a5ff8d0c561f4,48c30674645e5a000b2d969b48950283d0927e61,4ab
8610adbf9a437cb4c763f927fb581f98553d6,4c7826a52118439f6f75cbd18e93f7d301d80c31,4e57ca2ca
464b7dcbf54f5dec64abc7f85403c68,5315c602da8e022acd36173deb2ecd3dc89959a3,56de7895dd78f7
68ae672ab6c03e7769485876cb,573880feb188b08c807539000d008942855f2e94,5a278764345096ad2c
0d793cf9c7f2a091ea861f,6076f653ad969082233b9e6b85cbb90aff103916,639a9a5581991f5d6475f7c9
4b69dde5881eb17b,6ae08c54259cee7d4084bebad54afa78f4d0d0b3,733008151c463b026c7c3e91fe64
0fe1297fca28,7855acbfe90dd90418f55a9da7ff595839a51b47,7a8e5c9e6ad90cebd8c93e38b0f6f70db4
d81c78

Project ID 0xd:

0b8ae81630780d294a516b606110b849e55712df,13de3d5f2c20d61004f803f1405e9bdac744fc82,1aa0f
89e071e39357d09c3cc578e852cc01621d4,275fb0b32e7af50ebd444fa5a35b3a6c1a87e54c,27922fdc18
b5847a2df0e204c649fb4b947d2028,2bf65339ffef9aa6fdea6b8d0bc53775970ef0eb,35d1eac6f385bebf9
a9409d94de19be8bd69b6ac,37ff4cf5126d3bfcee43893871e0b83804ba28f9,413a211c4c758dde083789
a9c3e2f3cb201c62c9,42ba30433f1c54e666aceca68f84dd850453de14,473489f20a289511fada0faadad9
88791b670883,47d811bd8bba9ee11aa9a21cf5c30a296b4ff120,4a4b09245d7a2a362dfae369316823ec7
ad563b8,4ae0f2464222c19bd85dbbd6488a3f0a6232be8a,5bf81d316ffd0d6f1a9d8bda2786a33eb05a07
a9,5f10e7ff2eea9385407f735e6f744b35a15595cd,5f2d22185a771e29313e47388fb332826bb3321d,62c

c75ecdee1839ea970a855fc391552737a1ee0,645e15d0a27c5b876039acf23f17f70424e4cc8a,6ada337428414efc21f3a9d74b4b83407d419aca

Cluster 6 – Czech Republic

Project ID 0x32:

0ddebaa2090946163e315e71389540c482ca5067,1481fd7cf7a0816456acd25f616920226ea521cf,1cd048f93e508070fbf5548a787ca070999e26f1,2305c0d3003a791fa3d41d3ccf8f54716a2b9d0c,28fd991cc43b4f07fcbd57025516bfc6f4255307,35a3b32765843cb15a9711b14327d72101caac4f,3ebb787135bed911f85ad67a1465228bf9e8f4f1,444220172cabb9b1d23928d0caa0c793ddcd09b7,4665214997fe72cef05aeca564b6716cd4368000,48681ce9f718a75750ab50a6d0e59816d222335d,48a7f2532518ce409083ab33a81f66e67acac8d9,57f390be7a720ed75d7b51a43c6fddf73f695830,5d2b0f2346a39fa0ae4e8d6d964e7b868fed2395,61fce97639070dc0ca0ea42f0639b6512b393aa4,651be69871c2cea7b06989129b1a9ceb95995ed2

Project ID 0x33:

1220458a852eb9fae9de6893500927c628fa23e2,24ec9fee77d9fca80bbcd141c47c9ce0c3612cd7,274cc9583206e103b4facca551c3410566c7202b,30c84b598f517058760c4bbb75aa58dd5b224618,3749841f7fa4e4cacc2c2471c67b013a21072f52,37f1118b25bce39efd9716f9cd86946b69a56bf5,3faaded4066144e2c7a8b3f75ae1df5fd8c310a8,3fb6dede44f96b24564625735f69f96d470f7250,40ec4550d01ee6db0eaf9670698232bd406c37ac,47fa9f9737bb128668d8d833c3a377d6a0d49080,56ed4702f2b9f23ea671e4601a46c7a07beeb20e,5b4190eade7880f5cc7be90245495a5b50294fe0

Project ID 0x34:

619a53e8aff6d9c30745597515446e2ab3ba1adb

References

[1] James Wyke, Sophos: “Vawtrak – International Crimeware1 as a Service”, December 2014, https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/sophos-vawtrak-international-crimeware-as-a-service-tpna.pdf?la=en

[2] Darien Huss and Matthew Mesa, Proofpoint: “In the Shadows: Vawtrak Aims to Get Stealthier by adding New Data Cloaking”, October 2015, https://www.proofpoint.com/us/threat-insight/post/In-The-Shadows

[3] Don Jackson, PhishLabs: “The unrelenting evolution of Vawtrak”, December 2014, http://blog.phishlabs.com/the-unrelenting-evolution-of-vawtrak

[4] Wikipedia: “Linear congruential generator”, https://en.wikipedia.org/wiki/Linear_congruential_generator

[5] Vadim Kotov: “Macro Redux: the Premium Package”, January 2016, http://labs.bromium.com/2016/02/03/macro-redux-the-premium-package/

[6] Jakub Křoustek, AVG: “Analysis of Banking Trojan Vawtrak”, March 2015, http://now.avg.com/wp-content/uploads/2015/03/avg_technologies_vawtrak_banking_trojan_report.pdf

[7] Proofpoint, “Vawtrak and UrlZone Banking Trojans Target Japan”, February 2016, https://www.proofpoint.com/us/threat-insight/post/Vawtrak-UrlZone-Banking-Trojans-Target-Japan