Attack Activities by Quasar Family - JPCERT/CC Eyes

By 喜野 孝太(Kota Kino)

Published: 2020-12-09 · Archived: 2026-04-05 14:44:44 UTC

Quasar [1] is an open source RAT (Remote Administration Tool) with a variety of functions. This is easy to use and therefore exploited by several APT actors. JPCERT/CC has confirmed that a group called APT10 used this tool in some targeted attacks against Japanese organisations.
As Quasar's source code is publicly available, there are many variants of this RAT seen in the wild (referred to as "Quasar Family" hereafter). Some of them have been used in attacks against Japanese organisations, and they are seen as a threat as well as Quasar itself.
This article introduces the details of Quasar and Quasar Family.

Quasar overview

Quasar offers many functions which are intended for purposes such as device management, support operation and employee monitoring. Figure 1 describes Quasar's functions and its supported environment as specified on GitHub.

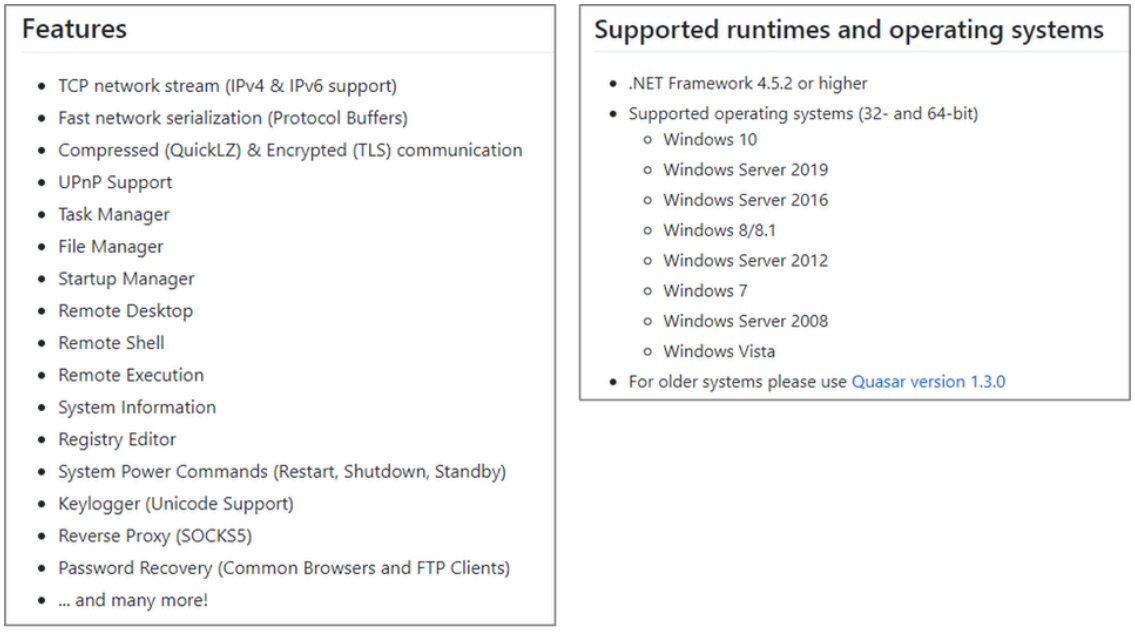


Figure 1: Quasar's functions and supported environment

This tool was called "xRAT" at the time of its initial release, however, it was renamed as "Quasar" in August 2015. The latest version is v1.4, released in June 2020.

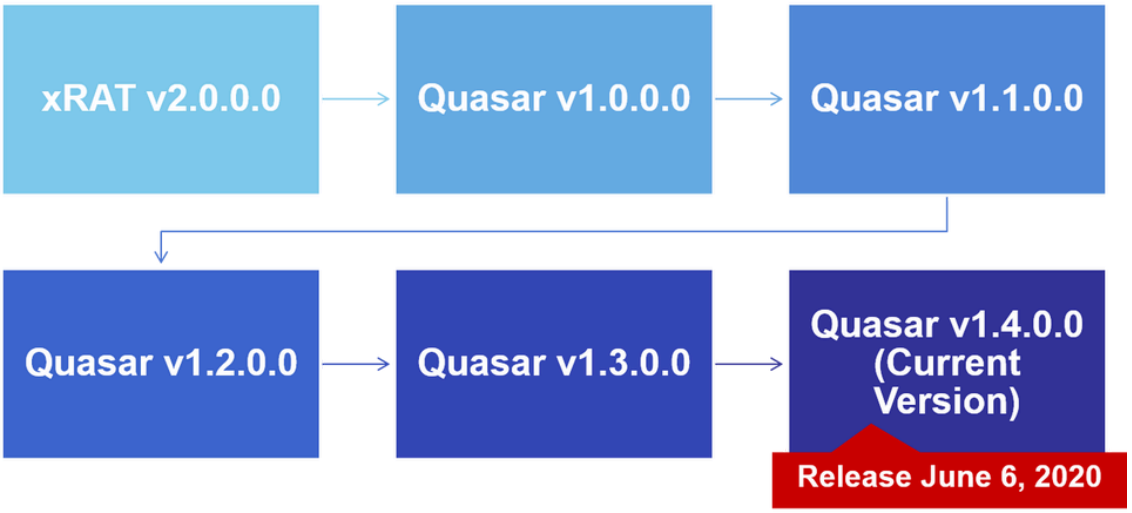


Figure 2: Quasar versions

As v1.3 and the earlier are still used in recent attacks, this article explains the functions of both v1.3 and v1.4.

Communication protocol

Quasar v1.3 uses its custom protocol which combines AES and QuickLZ. In v1.4, however, Protocol Buffer (developed by Google) is used for data serialisation instead. In addition, the entire communication is encrypted with TLS1.2.
Figure 3 shows the comparison of the communication format in v1.3 and v1.4.

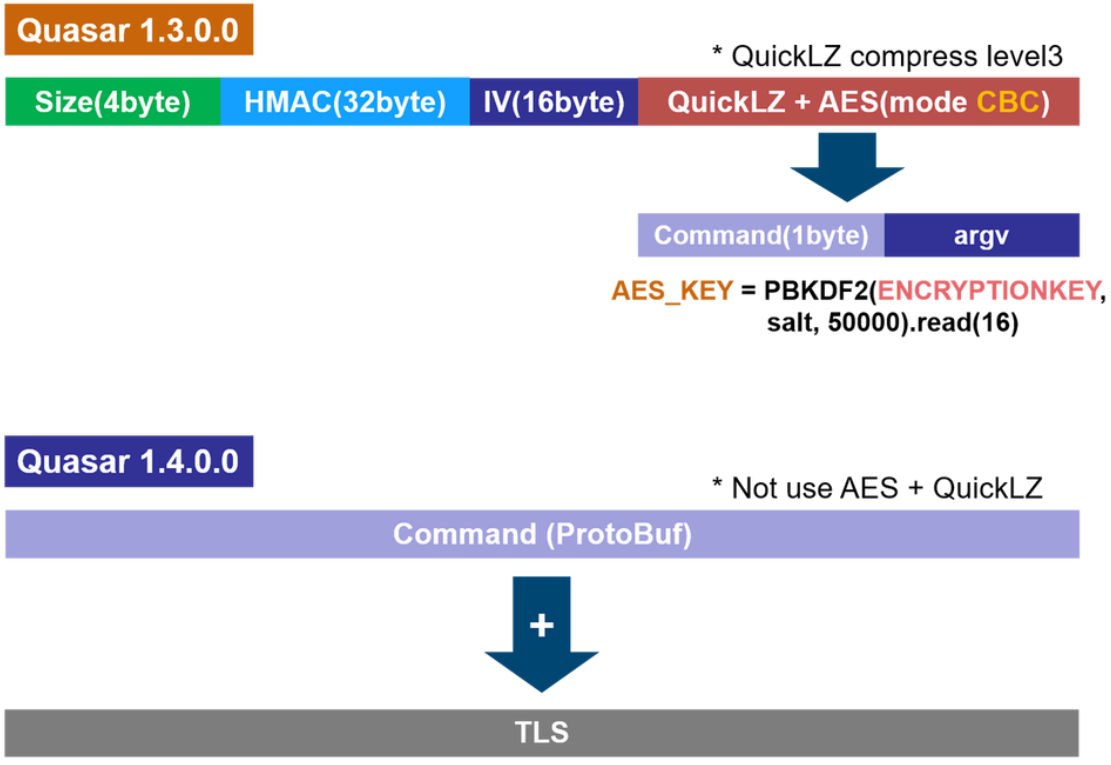


Figure 3: Quasar's communication format

Communication flow

In v1.3, once a client connects to a server, authentication is performed. After that, the main body of data including the commands are exchanged. On the other hand, the authentication is replaced by a TLS handshake in v1.4, and the data exchange begins after that.
Figure 4 illustrates Quasar's communication flow between a client and a server.

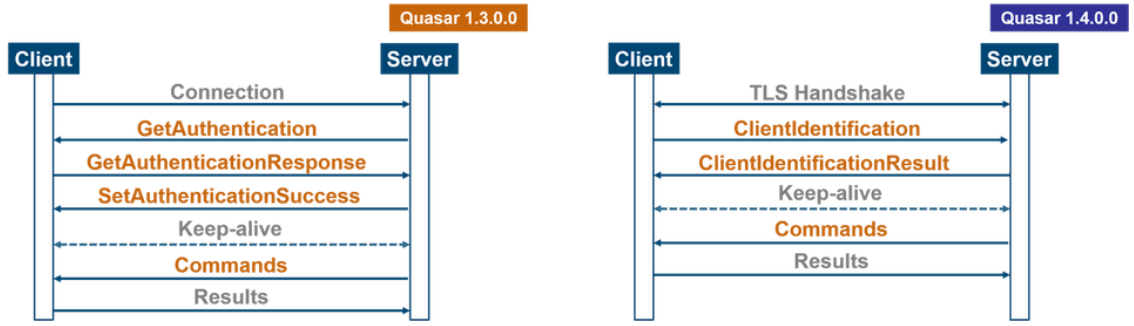


Figure 4: Quasar's communication flow

Configuration

Quasar possesses its configuration in itself. It is encrypted by the combination of AES and BASE64 encoding. It is decrypted with the value specified in "ENCRYPTIONKEY" in the configuration when executed.

```
// Token: 0x04000008 RID: 8
public static string VERSION = "UvADP/X6YS4Pi/TtR6RCQ49f9HntDsrnWRpHNg4Sp7M9cqQvJrvNpoozrQAdD2e39pa5nVYAe1k9B5+dvQGMEw==";

// Token: 0x04000009 RID: 9
public static string HOSTS = "Tqu77dMuK/ZIsz66TL5IQUxE8/LOhIir5uxAqtiTHCs4hvRJd5xp4o035SrBtDLtmdK+sDe2ewmNBkSJWjug8HEoms8v9d8aFtI28DIw4QgEkK5XI

// Token: 0x0400000A RID: 10
public static int RECONNECTDELAY = 3000;

// Token: 0x0400000B RID: 11
public static string KEY = "ebdDqS8O89sUEWPiItdABQ==";

// Token: 0x0400000C RID: 12
public static string AUTHKEY = "SQc2h9nqnECTE5yMFKY1PRF9QHSIbZLcqE0ccuiIHMgbDR8OXD7Fv9XDKcDJAU/mouBB4/KAEpaWUM9KJEJZBQ==";

// Token: 0x0400000D RID: 13
public static Environment.SpecialFolder specialFolder_0 = Environment.SpecialFolder.ApplicationData;

// Token: 0x0400000E RID: 14
public static string DIRECTORY = Environment.GetFolderPath(Init_config.specialFolder_0);

// Token: 0x0400000F RID: 15
public static string SUBDIRECTORY = "di7ALkOAISIPt5o4bbnsEuESGLSgwUEYBMBMHKzTpR9SUruwW6dmfsS9Z1XIyrt3I8xpkwaScvpuQA9aFCGCVA==";

// Token: 0x04000010 RID: 16
public static string INSTALLNAME = "/np6aAFTWsNmJIk0C428HV7MCpkYc+gwJURNOYibATKpCO0pqi9qblwFMLXptV/cDEbDjaYbN/Zfs2B8/JSBJQ==";

// Token: 0x04000011 RID: 17
public static bool INSTALL = false;

// Token: 0x04000012 RID: 18
public static bool STARTUP = false;

// Token: 0x04000013 RID: 19
public static string MUTEX = "X1T4grV+qtsrjEJxWiPtOWeVzcIi57988AaOTO8sIRtO0sEDWr1vO95E1237JEEUBuUNTBICXtHMavYPERq9LQMVUTWE/fa5pptXESQRxwA=";
```

Figure 5: Quasar configuration

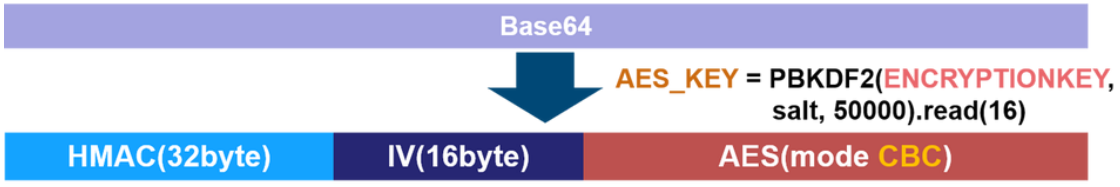


Figure 6: Configuration format

Table 1 details the configuration for Quasar.

Table 1: Configuration

VERSION	INSTALL	LOGDIRECTORY (1.3)
HOSTS	STARTUP	SERVERSIGNATURE (1.4)
PORT (xRAT only)	MUTEX	SERVERCERTIFICATESTR (1.4)
RECONNECTDELY	STARTUPKEY	SERVERCERTIFICATE (1.4)
KEY	HIDEFILE	HIDELOGDIRECTORY (1.3)
AUTHKEY	ENABLEUACESCALATION (xRAT only)	HIDELOGSUBDIRECTORY (1.3)
DIRECTORY	ENABLELOGGER	INSTALLPATH (1.4)
SUBDIRECTORY	ENCRYPTIONKEY	LOGSPATH (1.4)
INSTALLNAME	TAG (1.3)	UNATTENDEDMODE (1.4)

Commands

In v1.3, command sets are defined for “typeof” calls. Figure 7 shows some examples of commands defined in Quasar.

Figure 7: Commands

Quasar Family

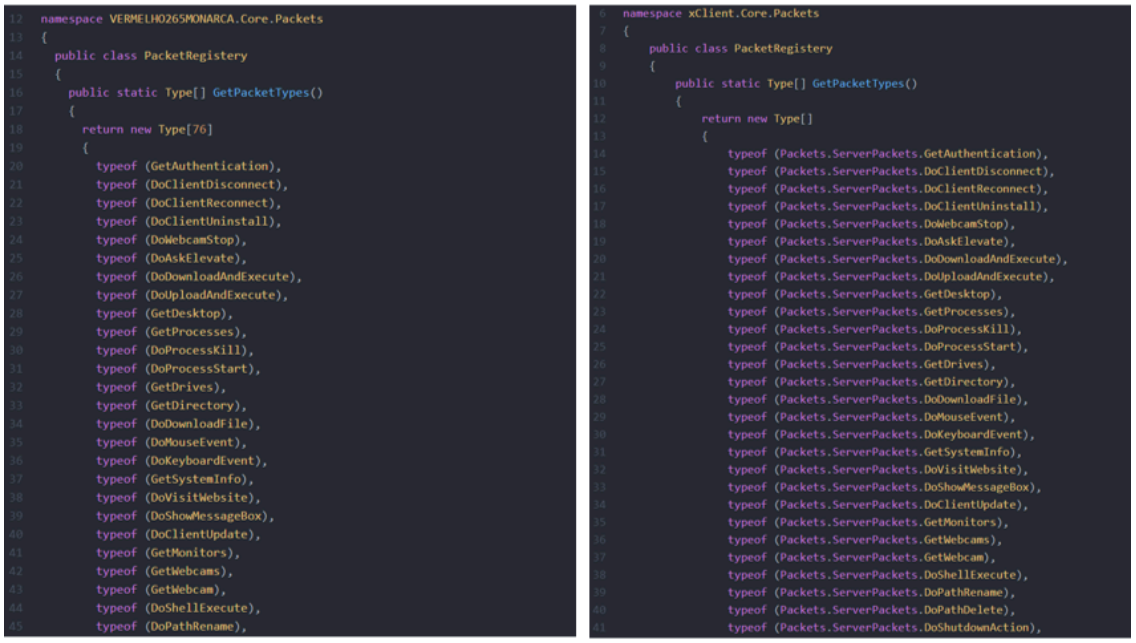
Table 2 is the list of Quasar Family derived from Quasar which JPCERT/CC confirmed.

Table 2: Quasar Family

Name	**Category**	**Configuration**	**Communication protocol**	**Use in attacks in the wild**
Golden Edition	Clone	Identical	Identical	Confirmed
XPCTRA	Clone	Custom	Identical	Confirmed
CinaRAT [2]	Clone	Identical	Identical	Confirmed
Xtremis 2.0 [3]	Clone	Identical	Identical	Not confirmed
QuasarStrike [4]	Clone	Identical	Identical	Not confirmed
VenomRAT	Clone	Identical	Identical	Not confirmed
RSMaster [5]	Partially copied	Custom	Identical	Not confirmed
Void-RAT	Partially copied	Custom	Identical	Confirmed
AsyncRAT [6]	Partially copied	Custom	Identical	Confirmed

* “Clone” in the category refers to variants which uses the entire source code of Quasar with some functions added or modified. “Partially copied” refers to variants created as a new RAT using parts of the original source code.

Figure 8 shows the comparison of commands embedded in XPCTRA and Quasar.

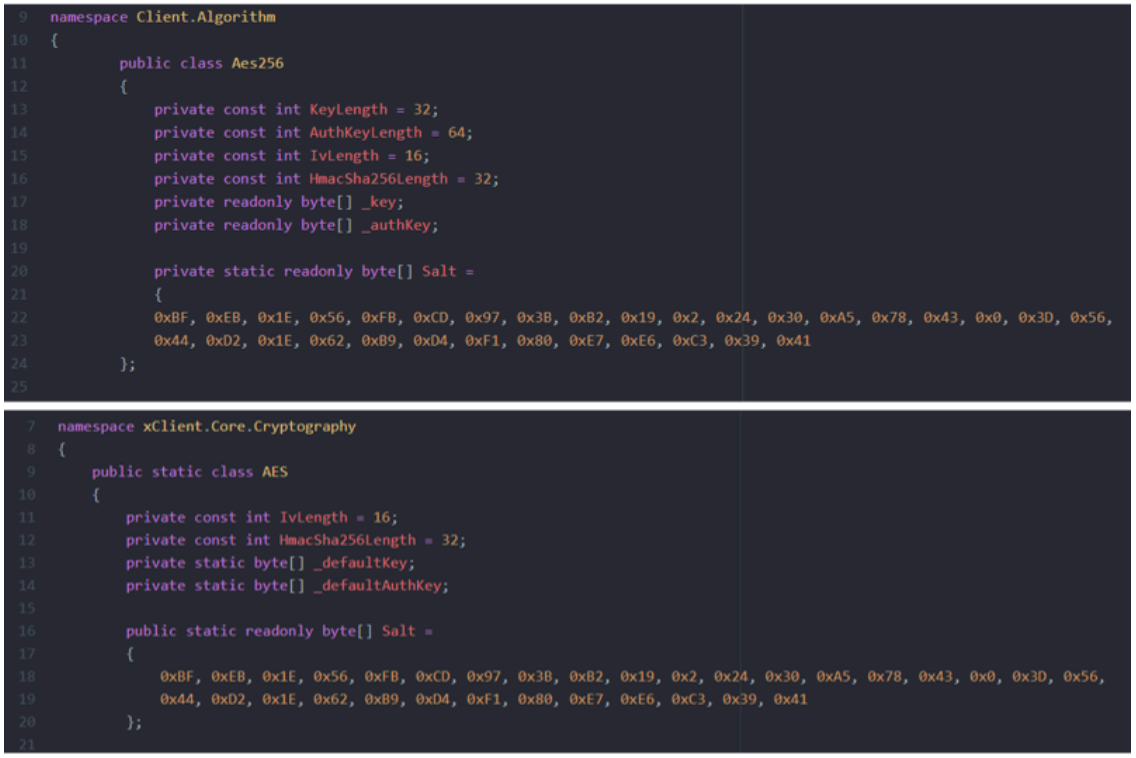


```
namespace VERMELHO265MONARCA.Core.Packets
{
  public class PacketRegistery
  {
    public static Type[] GetPacketTypes()
    {
      return new Type[76]
      {
        typeof (GetAuthentication),
        typeof (DoClientDisconnect),
        typeof (DoClientReconnect),
        typeof (DoClientUninstall),
        typeof (DoWebcamStop),
        typeof (DoAskElevate),
        typeof (DoDownloadAndExecute),
        typeof (DoUploadAndExecute),
        typeof (GetDesktop),
        typeof (GetProcesses),
        typeof (DoProcessKill),
        typeof (DoProcessStart),
        typeof (GetDrives),
        typeof (GetDirectory),
        typeof (DoDownloadFile),
        typeof (DoMouseEvent),
        typeof (DoKeyboardEvent),
        typeof (GetSystemInfo),
        typeof (DoVisitWebsite),
        typeof (DoShowMessageBox),
        typeof (DoClientUpdate),
        typeof (GetMonitors),
        typeof (GetWebcams),
        typeof (GetWebcam),
        typeof (DoShellExecute),
        typeof (DoPathRename),
```

```
namespace xClient.Core.Packets
{
    public class PacketRegistery
    {
        public static Type[] GetPacketTypes()
        {
            return new Type[]
            {
                typeof (Packets.ServerPackets.GetAuthentication),
                typeof (Packets.ServerPackets.DoClientDisconnect),
                typeof (Packets.ServerPackets.DoClientReconnect),
                typeof (Packets.ServerPackets.DoClientUninstall),
                typeof (Packets.ServerPackets.DoWebcamStop),
                typeof (Packets.ServerPackets.DoAskElevate),
                typeof (Packets.ServerPackets.DoDownloadAndExecute),
                typeof (Packets.ServerPackets.DoUploadAndExecute),
                typeof (Packets.ServerPackets.GetDesktop),
                typeof (Packets.ServerPackets.GetProcesses),
                typeof (Packets.ServerPackets.DoProcessKill),
                typeof (Packets.ServerPackets.DoProcessStart),
                typeof (Packets.ServerPackets.GetDrives),
                typeof (Packets.ServerPackets.GetDirectory),
                typeof (Packets.ServerPackets.DoDownloadFile),
                typeof (Packets.ServerPackets.DoMouseEvent),
                typeof (Packets.ServerPackets.DoKeyboardEvent),
                typeof (Packets.ServerPackets.GetSystemInfo),
                typeof (Packets.ServerPackets.DoVisitWebsite),
                typeof (Packets.ServerPackets.DoShowMessageBox),
                typeof (Packets.ServerPackets.DoClientUpdate),
                typeof (Packets.ServerPackets.GetMonitors),
                typeof (Packets.ServerPackets.GetWebcams),
                typeof (Packets.ServerPackets.GetWebcam),
                typeof (Packets.ServerPackets.DoShellExecute),
                typeof (Packets.ServerPackets.DoPathRename),
                typeof (Packets.ServerPackets.DoPathDelete),
                typeof (Packets.ServerPackets.DoShutdownAction),
```

Figure 8: Comparison of commands
(Left: XPCTRA / Right: Quasar)

In the comparison above, it is clear that commands in XPCTRA are mostly identical to those in Quasar.
Figure 9 shows the comparison of the salt value in AsyncRAT and Quasar.



```
namespace Client.Algorithm
{
        public class Aes256
        {
            private const int KeyLength = 32;
            private const int AuthKeyLength = 64;
            private const int IvLength = 16;
            private const int HmacSha256Length = 32;
            private readonly byte[] _key;
            private readonly byte[] _authKey;

            private static readonly byte[] Salt =
            {
            0xBF, 0xEB, 0x1E, 0x56, 0xFB, 0xCD, 0x97, 0x3B, 0xB2, 0x19, 0x2, 0x24, 0x30, 0xA5, 0x78, 0x43, 0x0, 0x3D, 0x56,
            0x44, 0xD2, 0x1E, 0x62, 0xB9, 0xD4, 0xF1, 0x80, 0xE7, 0xE6, 0xC3, 0x39, 0x41
        };
```

```
namespace xClient.Core.Cryptography
{
    public static class AES
    {
        private const int IvLength = 16;
        private const int HmacSha256Length = 32;
        private static byte[] _defaultKey;
        private static byte[] _defaultAuthKey;

        public static readonly byte[] Salt =
        {
            0xBF, 0xEB, 0x1E, 0x56, 0xFB, 0xCD, 0x97, 0x3B, 0xB2, 0x19, 0x2, 0x24, 0x30, 0xA5, 0x78, 0x43, 0x0, 0x3D, 0x56,
            0x44, 0xD2, 0x1E, 0x62, 0xB9, 0xD4, 0xF1, 0x80, 0xE7, 0xE6, 0xC3, 0x39, 0x41
        };
```

Figure 9: Comparison of salt value
(Above: AsyncRAT / Below: Quasar)

The salt value in AsyncRAT is identical to that in Quasar.
As Quasar Family applies some parts of the source code of Quasar, its configuration and communication protocol

are also identical. In some cases, some functions are customised, and as a result, some new configuration and commands are added.

Attack campaigns using Quasar

Quasar has been used in many attack campaigns. Table 3 lists the differences of Quasar used by each attack group.

Table 3: Example of Quasar used by attack group

Attack group	Quasar version	Customisation	Obfuscation
APT33	1.3.0.0	No	ConfuserEx v1.0.0
Gorgon Group	-	No	
APT-C-09	2.0.0.0 RELEASE3	No	
DustySky	1.1.0.0	No	
APT10	2.0.0.0(Custom Version)	Yes	ConfuserEx v1.0.0

The original Quasar with the default configuration value was used in most cases. Figure 10 shows an example configuration of Quasar used by APT 33.

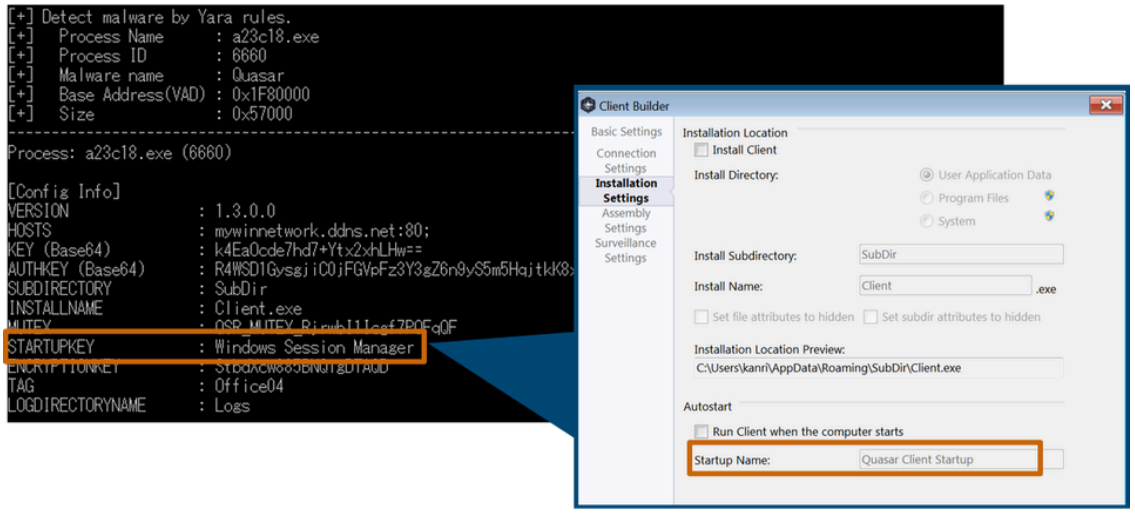


Figure 10: Configuration of Quasar used by APT33

In most parts, the default values of the builder generating Quasar are used as is, except for STARTUPKEY. This way, attacker groups use the default values as per the original to avoid leaving any distinctive evidence.

In some cases, attackers customise Quasar. For example, APT 10 updated some features and used it in some attacks. The following sections will cover the details of this custom Quasar.

Configuration

Quasar used by APT 10 (hereafter “custom Quasar”) has the following additional values in the configuration.

- DOWNLOAD_URL
- PROXY

Figure 11 shows the comparison of configuration in the custom Quasar and the original Quasar.

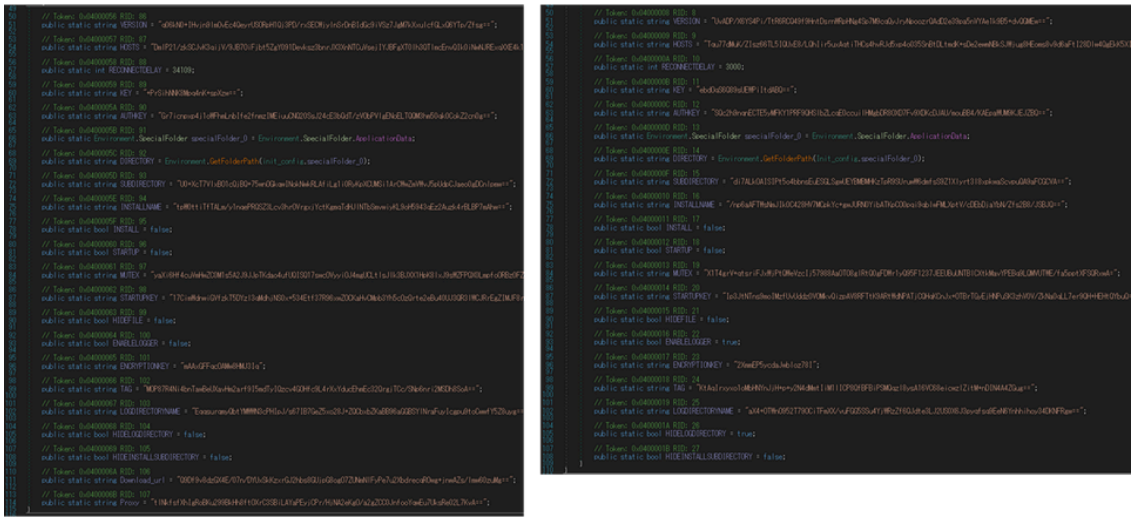


Figure 11: Comparison of configuration
(Left: custom Quasar / Right: original Quasar)

In “PROXY”, a proxy server URL can be configured. This ensures that the custom Quasar is able to communicate with a C2 server even if the target’s environment uses proxy servers.
While the original Quasar uses CBC mode when encrypting configuration with AES, the custom Quasar uses CFB mode.

Figure 12: Comparison of AES code
(Left: custom Quasar / Right: original Quasar)

Added/deleted commands

There are some changes to the commands in the custom Quasar. Figure 13 shows the comparison of commands in the custom Quasar and the original Quasar.

```
            typeof(object),
            typeof(DoLoadRegistryKey),
            typeof(DoCreateRegistryKey),
            typeof(DoDeleteRegistryKey),
            typeof(DoRenameRegistryKey),
            typeof(DoCreateRegistryValue),
            typeof(DoDeleteRegistryValue),
            typeof(DoRenameRegistryValue),
            typeof(DoChangeRegistryValue),
            typeof(SetAuthenticationSuccess),
            typeof(GetConnections),
            typeof(DoCloseConnection),
            typeof(GetAuthenticationResponse),
            typeof(SetStatus),
            typeof(SetStatusFileManager),
            typeof(object),
            typeof(GetDesktopResponse),
            typeof(GetProcessesResponse),
            typeof(GetDrivesResponse),
            typeof(GetDirectoryResponse),
            typeof(DoDownloadFileResponse),
            typeof(GetSystemInfoResponse),
            typeof(GetMonitorsResponse),
            typeof(object),
            typeof(object),
            typeof(DoShellExecuteResponse),
            typeof(object),
            typeof(object),
            typeof(object),
            typeof(GetRegistryKeysResponse),
            typeof(GetCreateRegistryKeyResponse),
            typeof(GetDeleteRegistryKeyResponse),
            typeof(GetRenameRegistryKeyResponse),
            typeof(GetCreateRegistryValueResponse),
            typeof(GetDeleteRegistryValueResponse),
            typeof(GetRenameRegistryValueResponse),
            typeof(GetChangeRegistryValueResponse),
            typeof(ReverseProxyConnect),
            typeof(ReverseProxyConnectResponse),
            typeof(ReverseProxyData),
            typeof(ReverseProxyDisconnect),
            typeof(GetConnectionsResponse),
            typeof(DoPluginResponse),
            typeof(DoPlugin)
        };
    }
```

```
            typeof(GetPasswords),
            typeof(DoLoadRegistryKey),
            typeof(DoCreateRegistryKey),
            typeof(DoDeleteRegistryKey),
            typeof(DoRenameRegistryKey),
            typeof(DoCreateRegistryValue),
            typeof(DoDeleteRegistryValue),
            typeof(DoRenameRegistryValue),
            typeof(DoChangeRegistryValue),
            typeof(SetAuthenticationSuccess),
            typeof(GetConnections),
            typeof(DoCloseConnection),
            typeof(GetAuthenticationResponse),
            typeof(SetStatus),
            typeof(SetStatusFileManager),
            typeof(SetUserStatus),
            typeof(GetDesktopResponse),
            typeof(GetProcessesResponse),
            typeof(GetDrivesResponse),
            typeof(GetDirectoryResponse),
            typeof(DoDownloadFileResponse),
            typeof(GetSystemInfoResponse),
            typeof(GetMonitorsResponse),
            typeof(GetWebcamsResponse),
            typeof(GetWebcamResponse),
            typeof(DoShellExecuteResponse),
            typeof(GetStartupItemsResponse),
            typeof(GetKeyloggerLogsResponse),
            typeof(GetPasswordsResponse),
            typeof(GetRegistryKeysResponse),
            typeof(GetCreateRegistryKeyResponse),
            typeof(GetDeleteRegistryKeyResponse),
            typeof(GetRenameRegistryKeyResponse),
            typeof(GetCreateRegistryValueResponse),
            typeof(GetDeleteRegistryValueResponse),
            typeof(GetRenameRegistryValueResponse),
            typeof(GetChangeRegistryValueResponse),
            typeof(ReverseProxyConnect),
            typeof(ReverseProxyConnectResponse),
            typeof(ReverseProxyData),
            typeof(ReverseProxyDisconnect),
            typeof(GetConnectionsResponse)
        };
    }
```

Figure 13: Comparison of commands
(Left: custom Quasar / Right: original Quasar)

In the custom Quasar, new commands DoPlugin and DoPluginResponse are added while some including keylogger are deleted.
With DoPlugin, new functions can be added by loading additional plugin modules. These new modules can be deleted with DoPluginResponse.
This change enables Quasar to dynamically extend its functions with commands while maintaining Quasar itself as simple as it can be. This suggests the attacker's intention to avoid detection by anti-virus software.

Error log creation

The custom Quasar has a function to create error logs. The file path of the error logs is hardcoded in itself.

```
    // Token: 0x060000DA RID: 218
    private static void create_error_log(Exception e)
    {
        if (e != null)
        {
            using (StreamWriter streamWriter = new StreamWriter(new FileStream("c:¥¥temp¥¥CEgibifi.tmp", FileMode.Append)))
            {
                streamWriter.AutoFlush = true;
                streamWriter.WriteLine(string.Format("==================[0]==================", DateTime.Now.ToString()));
                streamWriter.WriteLine(e.Message);
                streamWriter.WriteLine(e.StackTrace);
                if (e.InnerException != null)
                {
                    streamWriter.WriteLine(e.InnerException.Message);
                    streamWriter.WriteLine(e.InnerException.StackTrace);
                }
            }
        }
    }
```

Figure 14: Error log creation

Communication protocol

The encryption algorithms for communication with a C2 server also differs in the custom Quasar. While the original Quasar uses AES and QuickLZ, the custom Quasar also uses XOR encoding. Figure 15 shows the XOR encoding process added to the custom Quasar.

```
35      internal static byte[] request_encode(byte[] payload)
36      {
37          payload = SafeQuickLZ.Compress(payload, 3);
38          payload = decode_main.Aes_decode(payload);
39          byte[] array = new byte[payload.Length + class_decode.HEADER_SIZE];
40          for (int i = 0; i < payload.Length; i++)
41          {
42              payload[i] ^= class_decode.Enc_Key[i % 4];
43          }
44          byte[] bytes = BitConverter.GetBytes(payload.Length);
45          for (int j = 0; j < bytes.Length; j++)
46          {
47              bytes[j] ^= class_decode.Enc_Key[j];
48          }
49          Array.Copy(bytes, array, class_decode.HEADER_SIZE);
50          Array.Copy(payload, 0, array, class_decode.HEADER_SIZE, payload.Length);
51          return array;
52      }
```

Figure 15: XOR encoding process

For AES encryption, the custom Quasar uses CFB mode instead of CBC mode, as seen in the configuration. The encryption methods are as follows:

- Original Quasar: QuickLZ + AES (mode CBC)
- Custom Quasar: QuickLZ + AES (mode **CFB**) + **XOR**

C2 server activities

JPCERT/CC investigated the activities of Quasar Family C2 servers based on the characteristics discussed above. As of November 2020, 76 IP addresses running as C2 servers have been identified. Figure 16 shows the distribution of Quasar Family C2 servers which were revealed in this investigation.

Figure 16: C2 server distribution

Multiple C2 servers are still running in different countries, which indicates its activeness.

In closing

Besides Quasar, other open source RATs are being used in ongoing attack cases [7]. Attackers are taking advantage of these tools to make attribution difficult and reduce the cost for developing attack infrastructure. It is estimated that this attack trends may continue.
A tool to support Quasar analysis (compatible with Quasar v1.3 only) is available on GitHub. We hope you find it useful.

- Kota Kino, Shusei Tomonaga
- In cooperation with Tomoaki Tani
  (Translated by Yukako Uchida)

Reference

[1] GitHub: Quasar
https://github.com/quasar/Quasar

[2] GitHub: CinaRAT
https://github.com/wearelegal/CinaRAT

[3] GitHub: Xtremis 2.0
https://github.com/pavitra14/Xtremis-V2.0

[4] GitHub: QuasarStrike
https://github.com/Q-Strike/QuasarStrike

[5] GitHub: RSMaster
https://github.com/Netskyes/rsmaster

[6] GitHub: AsyncRAT
https://github.com/NYAN-x-CAT/AsyncRAT-C-Sharp

[7] Japan Security Analyst Conference 2020 (Opening Talk): Looking back on the incidents in 2019
https://jsac.jpcert.or.jp/archive/2020/pdf/JSAC2020_0_JPCERT_en.pdf

---