vb2023 LONDON

4 - 6 October, 2023 / London, United Kingdom

LET'S GO DOOR WITH KCP

Yoshihiro Ishikawa & Takuma Matsumoto

LAC Cyber Emergency Center, Japan

yoshihiro.ishikawa@lac.co.jp
takuma.matsumoto@lac.co.jp

ABSTRACT

In 2022 we observed the use of new APT malware by an unknown China-based APT actor across several incidents in Japan. The malware uses KCP protocol for backdoor communication and was coded in Golang on multiple platform operating systems – we named it 'gokcpdoor'.

This backdoor has 20 commands and connects with C2 servers via KCP over UDP. KCP is a communication protocol that maximizes bandwidth for reliable, low-latency communication. The protocol was designed by 'skywind3000', and its source code is publicly available [1]. KCP is commonly implemented in proxy software, streaming services, and online games. Most of the information about KCP is written in Chinese, so we think this protocol is relatively common among Chinese speakers.

Recently, it has been reported that China-based APT actors are using KCP protocol in some APT malware. However, there are few reports of this protocol being used in actual attack activity, and it is not in common use. Therefore, we think that gokcpdoor is an interesting piece of malware since it uses KCP protocol for C2 communication.

In this paper we describe the analysis results of gokcpdoor and related threats to help prevent similar attacks in the future.

A STUDY OF KCP

In this section we introduce the KCP protocol and kcp-go library [2], used by gokcpdoor.

KCP protocol

KCP is a fast and reliable automatic repeat-request (ARQ) protocol which provides low-latency communications. This protocol was devised by skywind3000 in 2011 and the code, written in C, was published on *GitHub* [1].

KCP protocol requires a transmission mode for sending and receiving the underlying data packets. Many implementations meet the requirement by utilizing UDP protocol as the transmission mode. According to KCP's *GitHub* page, the transmission speed of KCP over UDP is 30% to 40% faster than TCP, but wastes 10% to 20% of bandwidth. Therefore, the implementation is used in software or services that require real-time performance, such as proxy software, online games, and streaming services.

KCP message segment

The message segment exchanged in KCP communication consists of a 24-byte header and variable length data, as shown in Figure 1 and Table 1. The command in the header is an essential field to identify transmission, acknowledgement, and retransmission, like the TCP control flag. As shown in Table 2, there are four types of commands, and IKCP_CMD_PUSH and IKCP_CMD_ACK are used most frequently.

Header (24 bytes)								Body
conv (4B)	**cmd** (1B)	frg (1B)	wnd (2B)	ts (4B)	sn (4B)	una (4B)	len (4B)	**data** (variable)

Figure 1: KCP message segment.

Field	Size	Description
conv	4 bytes	Session number
cmd	1 byte	Commands
frg	1 byte	Number of fragments
wnd	2 bytes	Window size
ts	4 bytes	Timestamp
sn	4 bytes	Serial number
una	4 bytes	Number of KCP message segments received
len	4 bytes	Length of the data segment
data	variable	Data segment

Table 1: List of parameters in KCP message segment.

Command	Value	Description
IKCP_CMD_PUSH	81 (0x51 'Q')	Data message
IKCP_CMD_ACK	82 (0x52 'R')	Acknowledgement message
IKCP_WASK	83 (0x53 'S')	Window probe message
IKCP_CMD_WINS	84 (0x54 'T')	Window receive message

Table 2: List of commands.

KCP communication flow

As illustrated in Figure 2, a sender sends the data message with IKCP_CMD_PUSH and receives IKCP_CMD_ACK as an acknowledgement. Moreover, the sender can send KCP messages without waiting for an acknowledgement, because the KCP message has window size (wnd) defined. If the receiver cannot send back immediately, the sender consumes all the window and eventually stops sending. In this case, in order to prevent deadlock, the sender sends a window probe on a regular basis and check the receiver's window size.

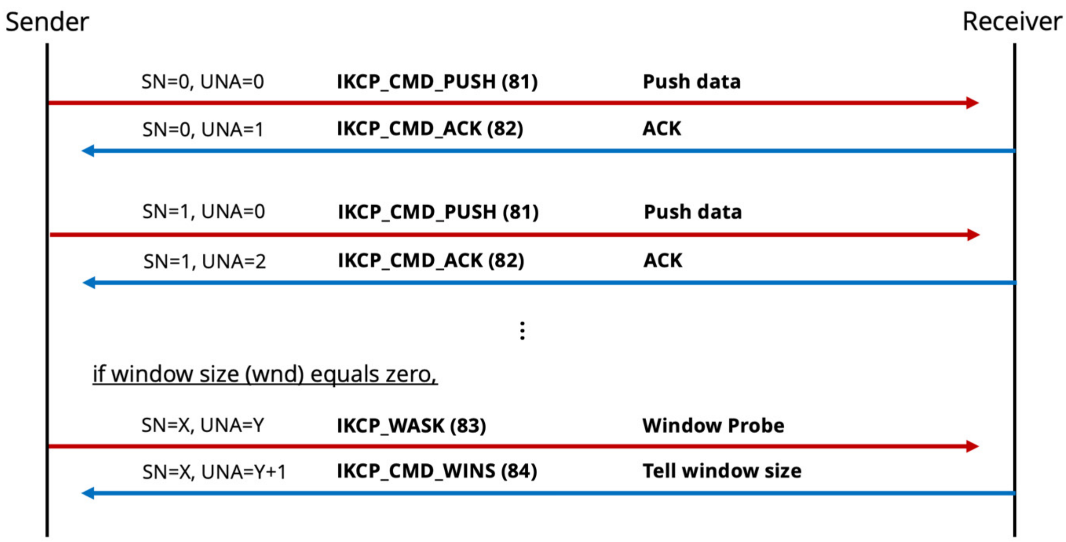

Figure 2: KCP one-way communication flow.

Kcp-go library

Kcp-go is a Reliable-UDP (RUDP) library for Golang with extensions based on KCP protocol. This library supports Forward Error Correction (FEC) with Reed-Solomon coding and packet-level encryption with AES, Blowfish, Salsa20, and so on. In addition, kcp-go uses CFB mode as a block cipher mode of operation; an initialization vector is hard-coded into the library.

Figure 3 and Table 3 demonstrate the structure of a kcp-go message. The structure consists of a 28-byte header and variable length body, and the KCP message is assigned to this body. In the header, kcp-go adds a Nonce, which is a random value, so that encrypting the same plaintext yields different results each time.

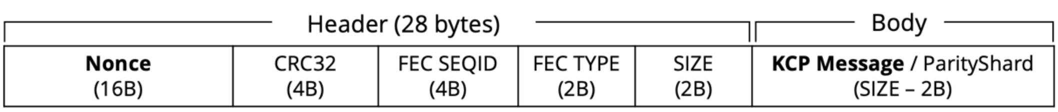

Figure 3: Structure of a kcp-go message.

Field	Size	Description
Nonce	16 bytes	Random value
CRC32	4 bytes	Checksum
FEC SEQ ID	4 bytes	Sequence ID for FEC
FEC Type	2 bytes	FEC type
Size	2 bytes	Length of data
Data	4 bytes	KCP message or parity shard for FEC

Table3: List of parameters in a kcp-go message.

APT MALWARE USING KCP PROTOCOL

In this section we discuss some APT malware that uses KCP for C2 communication.

Figure 4 shows a timeline of malware families using KCP protocol. We confirmed implementation of the KCP protocol code for the first time in the Crosswalk malware in April 2020. Much of the malware implementing the KCP protocol is related to China-based threat group APT41. There are few reports of this protocol being used in actual attack activity, and it is not in common use. In addition, we have not yet confirmed the implementation of KCP for PseudoManuscrypt. However, the newly confirmed (March 2022) gokcpdoor malware is different. This malware utilizes the kcp-go library to actually perform C2 communication with the KCP protocol. In the following, we will introduce the KCP implementation of each malware family.

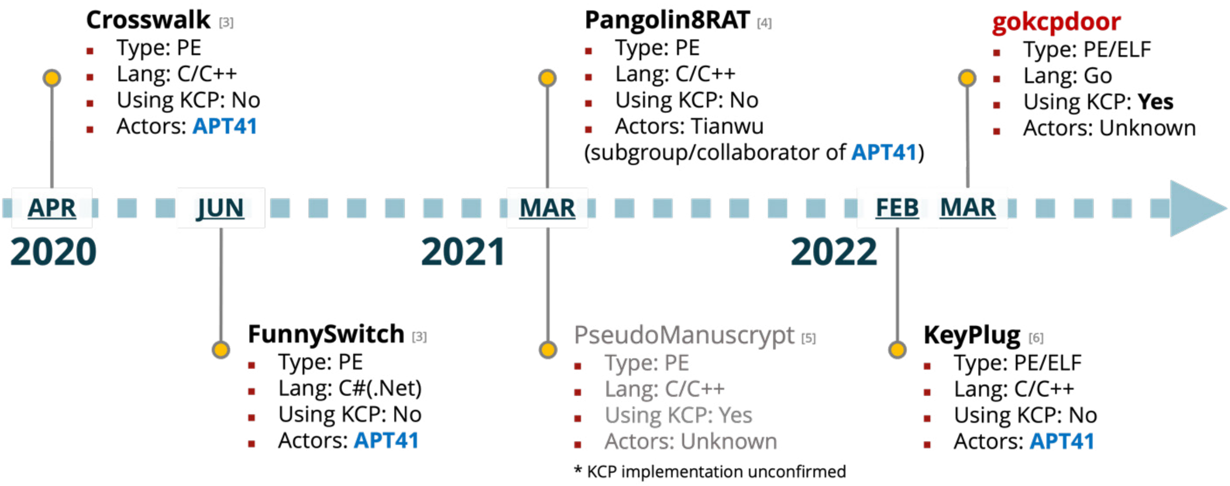

Figure 4: Timeline of malware families using KCP protocol.

Crosswalk, KeyPlug and Pangolin8RAT with KCP

We start with a comparison of Crosswalk, KeyPlug and Pangolin8RAT, developed in C language (Figure 5). The left-hand side is the original KCP source code in open source and on the right is the *IDA* decompiled code for each piece of malware. You can see that the same code is partially implemented.

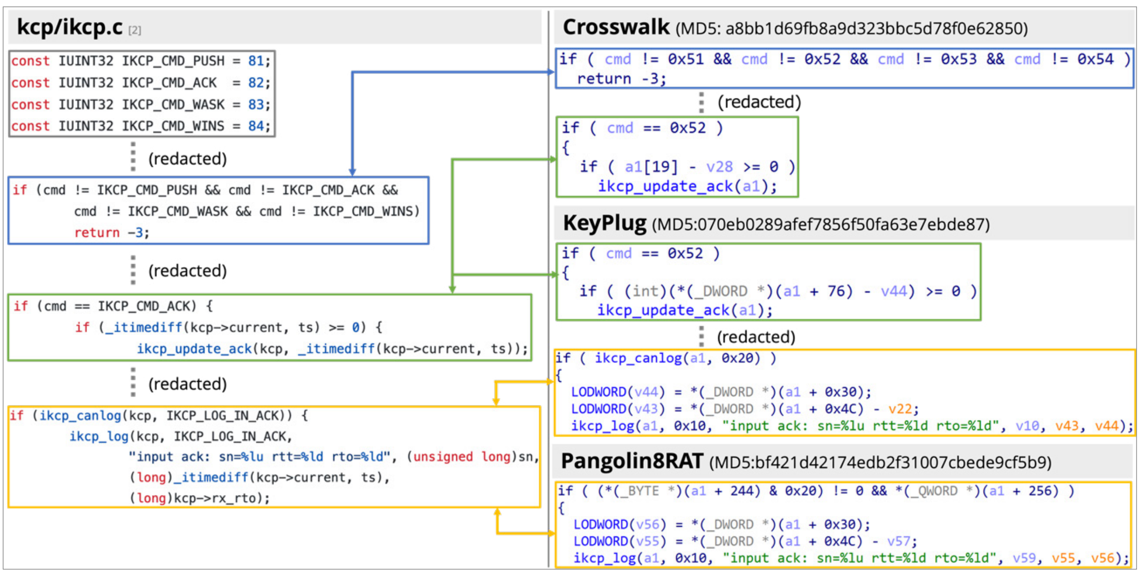

Figure 5: Comparison of Crosswalk, KeyPlug and Pangolin8RAT with KCP.

FunnySwitch with KCP

Next is a comparison of FunnySwitch, developed in C# language (Figure 6). The left-hand side is the kcp-dotnet source code in open source and the right side is the decompiled code of FunnySwitch. This is also the same code.

kcp-dotnet/KCP.cs [3]

```
namespace Network
{
    public class KCP
    {
        public const int IKCP_RTO_NDL = 30;
        public const int IKCP_RTO_MIN = 100;
```

⋮ (redacted)

```
public static void ikcp_encode8u(byte[] p, int offset, byte c)
{
    p[offset] = c;
}

// decode 8 bits unsigned int
public static byte ikcp_decode8u(byte[] p, ref int offset)
{
    return p[offset++];
}

// encode 16 bits unsigned int (lsb)
public static void ikcp_encode16u(byte[] p, int offset, UInt16 v)
{
    p[offset] = (byte)(v & 0xFF);
    p[offset + 1] = (byte)(v >> 8);
```

FunnySwitch (MD5: 2b0c692d9eafed5e24f2b52234ea0fa2)

```
namespace Network
{
    // Token: 0x0200003A RID: 58
    public class KCP
    {
        // Token: 0x06000114 RID: 276 RVA: 0x0000AEF0 File Offset: 0x000090F0
        public static void ikcp_encode8u(byte[] p, int offset, byte c)
        {
            p[offset] = c;
        }

        // Token: 0x06000115 RID: 277 RVA: 0x0000AF04 File Offset: 0x00009104
        public static byte ikcp_decode8u(byte[] p, ref int offset)
        {
            int num = offset;
            offset = num + 1;
            return p[num];
        }

        // Token: 0x06000116 RID: 278 RVA: 0x0000AF1C File Offset: 0x0000911C
        public static void ikcp_encode16u(byte[] p, int offset, ushort v)
        {
            p[offset] = (byte)(v & 255);
            p[offset + 1] = (byte)(v >> 8);
        }
```

⋮ (redacted)

```
// Token: 0x040000B9 RID: 185
public const int IKCP_RTO_NDL = 30;

// Token: 0x040000BA RID: 186
public const int IKCP_RTO_MIN = 100;
```

Figure 6: Comparison of FunnySwitch with KCP.

Gokcpdoor with KCP

Finally we look at gokcpdoor, which is developed in Golang (Figure 7). The left-hand side is the kcp-go source code and on the right side is the decompiled code of gokcpdoor. The identical code implementation shows that gokcpdoor uses the code from kcp.go.

kcp-go/kcp.go

```
func (kcp *KCP) Input(data []byte, regular, ackNoDelay bool)

if cmd != IKCP_CMD_PUSH && cmd != IKCP_CMD_ACK &&
        cmd != IKCP_CMD_WASK && cmd != IKCP_CMD_WINS {
        return -3
```

⋮ (redacted)

```
if cmd == IKCP_CMD_ACK {
        kcp.parse_ack(sn)
        kcp.parse_fastack(sn, ts)
        flag |= 1
        latest = ts
```

⋮ (redacted)

```
if windowSlides { // if window has slided, flush
        kcp.flush(false)
} else if ackNoDelay && len(kcp.acklist) > 0 { // ack immediately
        kcp.flush(true)
}
return 0
```

gokcpdoor (MD5: a6f4a5ec66b7c5f275e793be02885543)

```
// program/kcp.(*KCP).Input

if ( cmd != 0x51 && cmd != 0x52 && cmd != 0x53 && cmd != 0x54 )
  return -3LL;
```

⋮ (redacted)

```
if ( cmd == 0x52 )
{
  program_kcp__ptr_KCP_parse_ack(v83, v68, v31, a4, a5, 0x52LL
  a2 = v68;
  program_kcp__ptr_KCP_parse_fastack(v83, v68, v66);
  v9 = v76 | 1;
  v11 = v66;
  a1 = v83;
```

⋮ (redacted)

```
if ( windowSlides )
{
  program_kcp__ptr_KCP_flush(v50, 0LL, a3, v45, a5);
}
else if ( (_BYTE)a6 && *(__int64 *)(v50 + 216) > 0 )
{
  program_kcp__ptr_KCP_flush(v50, 1uLL, a3, v45, a5);
}
return 0LL;
```

Figure 7: Comparison of gokcpdoor with KCP.

DEEP DIVE INTO GOKCPDOOR

In the core of this paper, we look at the gokcpdoor malware samples in more detail, including the differences between them and the implemented functions.

Gokcpdoor is a piece of malware with backdoor functionality, coded in Golang and cross-compiled for *Linux* (ELF) and *Windows* (PE). There are minor differences, but both versions have the same functionality. Both gokcpdoor samples we have confirmed are built with go1.17.5 (Figure 8). Also, this malware uses multiple OSS libraries. For more information on OSS libraries, please see Appendix 1.

```
aGoBuildinf     db ' Go buildinf:'
                db 8                    ; pointer size
                db 0                    ; little endian
                dq offset off_7AF0E0    ; "go1.17.5"
                dq offset off_7AF130
```

Figure 8: Embedded Go build version.

We also named this backdoor malware 'gokcpdoor' because its compile path contained the string 'gokcpdoor', as shown in Figure 9.

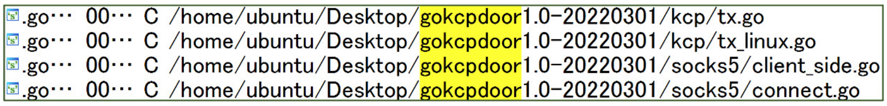

Figure 9: Compile path containing gokcpdoor strings.

Comparison of Linux and Windows gokcpdoor functions

Figure 10 shows specific functions implemented by gokcpdoor in *Linux* and *Windows*. Malware functionality is almost identical on *Linux* and *Windows*, but the *Windows* version has one characteristic function named 'main_WinExec'. The function literally executes the specified command by calling the WinExec API.

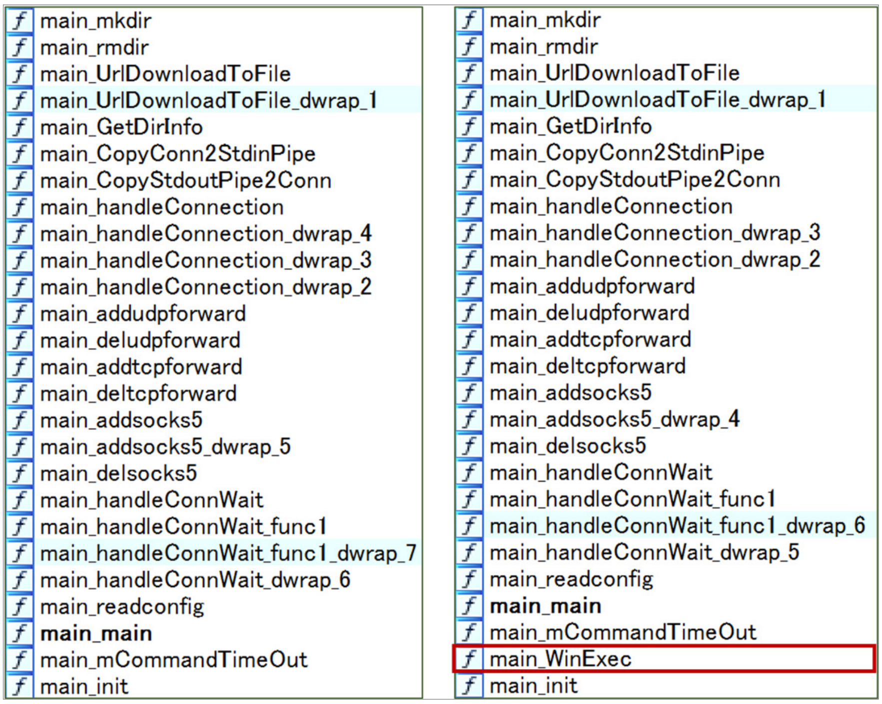

Figure 10: Gokcpdoor functions (left: Linux versions, right: Windows versions).

Backdoor function

Gokcpdoor starts opening a port with a hard-coded port number using the net_ResolveUDPAddr functions and net_ ListenUDP functions of Golang (Figure 11). Figure 12 is the result of executing the 'ss' command, which can display information about the socket. In this sample, we can see that 10054/udp is open. In addition, the backdoor port number differs depending on the sample.

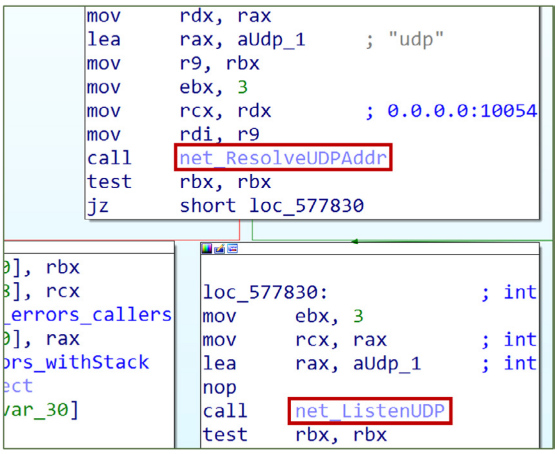

Figure 11: Opening 10054/udp using net package functions.

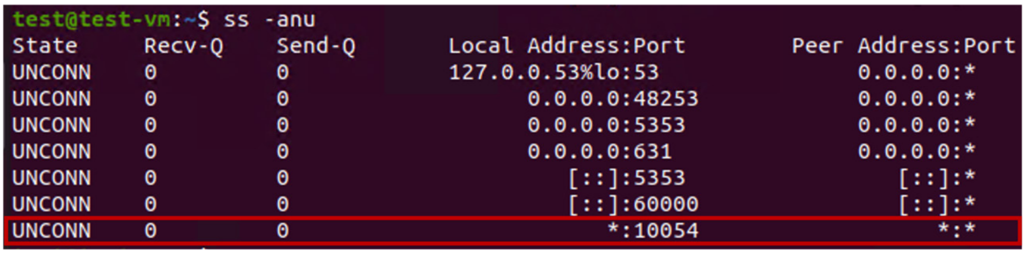

Figure 12: All open UDP ports listed by the 'ss' command.

Figure 13 shows part of a function that decodes the port number opened by gokcpdoor. The backdoor port number has been encoded by XOR and Base64. In this case, there is the encrypted binary data at offset '0x7AEFD0' in the blue-line frame. Decoding with the hard-coded XOR key and Base64, you can get the port number and the string 'nId2jUd3Ld1Fxe'. This is a fixed string sent when starting the backdoor C2 operation. By sending it once, multiple commands can be executed until the backdoor session expires.

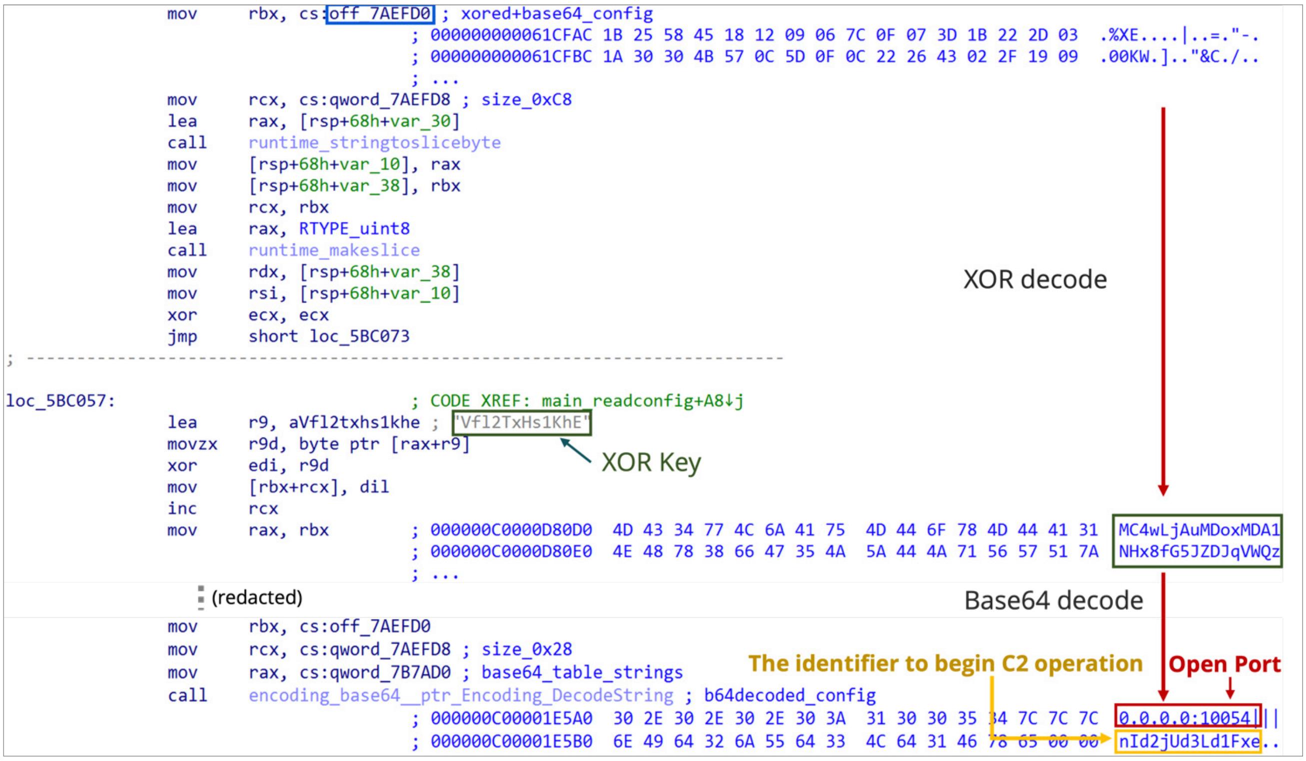

Figure 13: Port number and the identifier decoding.

C2 commands

Table 4 (on the following page) shows a list of C2 commands for gokcpdoor. The malware has 20 commands, for execution, uploading and downloading files, file manipulation, port forwarding, and so on. In particular, the exec, shell, upload and download commands play an important role in controlling the victim host.

Communication data format

Gokcpdoor sends and receives data in Base64-encoded strings and a newline code format. For example, the C2 commands to execute the *Windows* calculator (calc.exe) are Base64-encoded 'exec' and 'calc.exe'. Each command/result is sent separately with a trailing line feed (LF) from the C2 server to gokcpdoor as UDP data after it has been encapsulated and encrypted by the kcp-go library, as illustrated in Figure 14.

Encryption method

Figure 15 shows the code for gokcpdoor's encryption method. It uses PBKDF2, Key Derivation Function, with HMAC-SHA-1 and AES 256 bit. We can see password, salt, iterations, and derived key length on this code.

The derived key is shown in the area highlighted in Figure 15 in grey. C2 commands and executions results are encrypted with AES using this key and a hard-corded initialization vector into the kcp-go library.

Command	Description
exec	Execute a program
shell	Start reverse shell session
wget	Download a file from URL on infected host
upload	Upload a file from C2 server to infected host
download	Download a file from infected host to C2 server
dir / ls	List the contents of the specified directory
mkdir	Create a directory
rm	Remove the specified directory or file
cd	Change current directory
pwd	Get current directory path
whoami / id	Get username by executing 'whoami' or 'id' command
getos	Get OS information by executing 'wmic os get name' or 'uname -a' command
ps	List all running processes
Ifconfig / ipconfig	List all network interfaces
netstat	Get network statistics about all active connections
portfoward	list: List all port forwarding settings add: Add port forwarding setting which TCP or UDP can be selected delete: Delete port forwarding setting
socks5	list: List all SOCKS5 settings add: Add SOCKS5 setting delete: Delete SOCKS5 setting
charset	Change character set (gokcpdoor only supports UTF-8)
back	End C2 command operation
exitprocess	Terminate own process

Table 4: List of C2 commands.

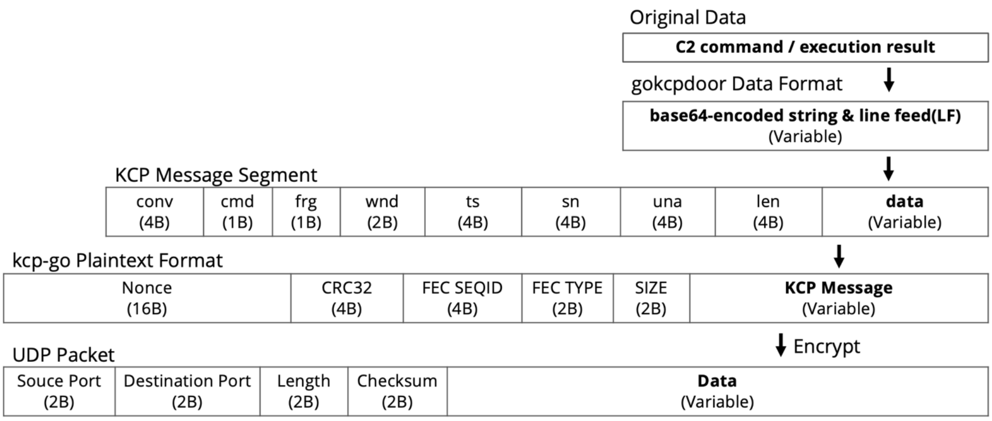

Figure 14: Encapsulation of transmitted data.

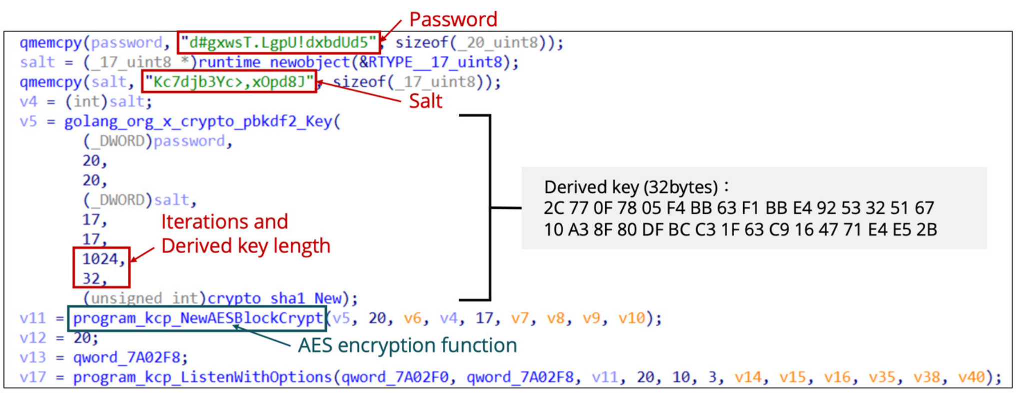

Figure 15: Gokcpdoor encryption method.

ATTRIBUTION

In this section we predict the attributes of the APT actors that use gokcpdoor.

Infection chain for gokcpdoor

Figure 16 shows an example of the gokcpdoor malware infection chain in 2021 to 2022. APT actors use stolen credentials to break into the victim's network and install malware using lateral movement. Gokcpdoor and the ABK downloader [8] were found on multiple servers and PCs. ABK has been used by Chinese APT actor Tick since 2019.

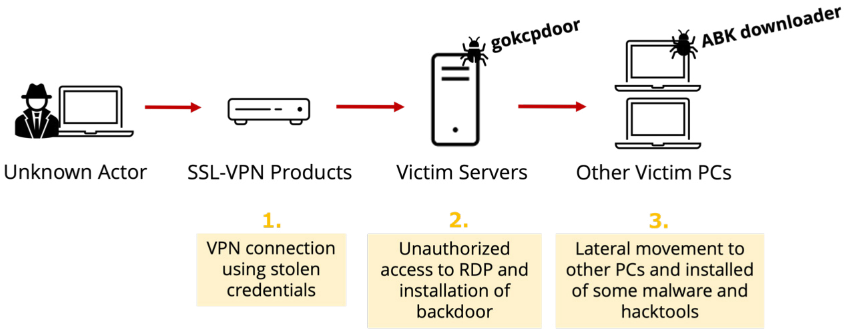

Figure 16: Example of a gokcpdoor infection flow.

Figure 17 illustrates the ABK downloader infection process. ABK is embedded as an encrypted payload in the OAED loader malware [9] (the payload is included after the yellow-line frame of Figure 18). The string 'v|xI?1bW' in the yellow-line frame is a marker to locate the payload. The OAED loader executes using the DLL side-loading technique and decrypts the payload with XOR. Then, the loader executes ABK via process hollowing into legitimate processes such as svchost.exe.

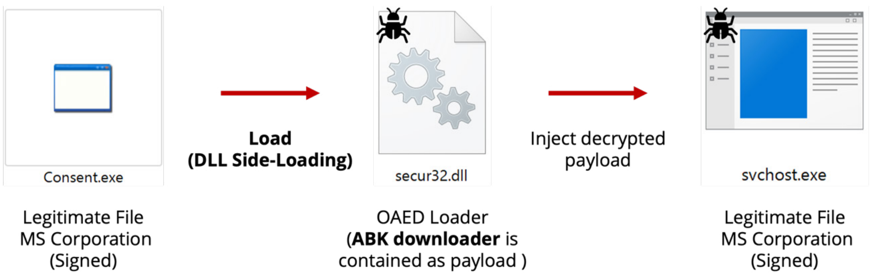

Figure 17: ABK downloader infection process.

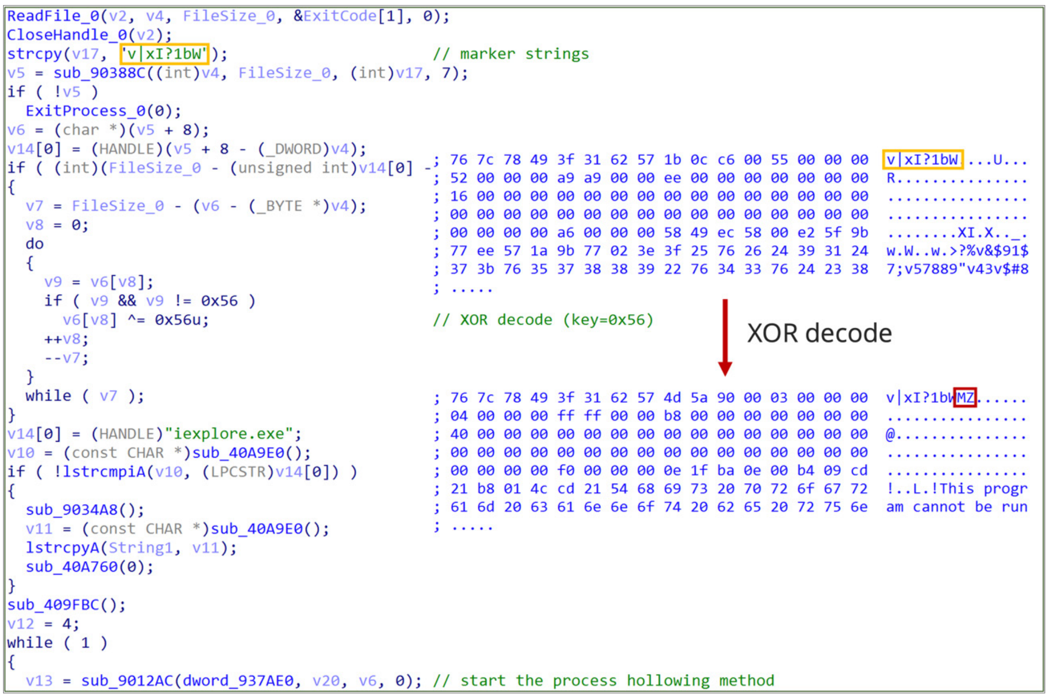

Figure 18: Decryption and process injection of the payload (ABK downloader).

The ABK downloader has four main characteristics:

- It detects some anti-virus products (Figure 19).
- It collects MAC address, system information and anti-virus product information and sends the details to C2 servers using no space User-Agent (Figure 20).
- It executes only during working hours (08:00 to 18:00) using the GetLocalTime API.
- It uses legitimate websites as C2 servers and downloads the next malware.

```
if ( !RegOpenKeyExA(
        HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Symantec\\Symantec Endpoint Protection\\CurrentVersion",
        0,
        0x20119u,
        &phkResult) )
{
  Type = 1;
  cbData = 1024;
  RegQueryValueExA(phkResult, "PRODUCTVERSION", 0, &Type, Data, &cbData);
  v1 = (const char *)Data;
}
RegCloseKey(phkResult);
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\TrendMicro\\AMSP", 0, 0x20119u, &hKey) )
{
  cbData = 1;
  Type = 1024;
  RegQueryValueExA(hKey, "TMFBE_GUID", 0, &cbData, v10, &Type);
  v1 = (const char *)v10;
}
RegCloseKey(hKey);
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\360Safe\\Liveup", 0, 0x20119u, &v4) )
{
  cbData = 1;
  Type = 1024;
  RegQueryValueExA(v4, "mid", 0, &cbData, v9, &Type);
  v1 = (const char *)v9;
}
RegCloseKey(v4);
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\McAfee\\Endpoint\\AV", 0, 0x20119u, &v6) )
{
  cbData = 1;
  Type = 1024;
  RegQueryValueExA(v6, "ProductVersion", 0, &cbData, v11, &Type);
```

Figure 19: Detection of specific anti-virus products.

```
LOBYTE(v10[0]) = 0;
sub_401990("Mozilla/4.0(compatible;MSIE8.0;WindowsNT6.0;Trident/4.0)", (void *)0x38, v10);
v13 = 0;
v8 = 15;
v7 = 0;
v10[10] = (int)v6;
LOBYTE(v6[0]) = 0;
sub_401990("http:[redacted]/anki/abuky.php", (void *)0x30, v6);
v13 = -1;
sub_4023F0(v6[0], (int)v6[1], (int)v6[2], (int)v6[3], v7, v8, v9, v10[0]);
```

No space

Figure 20: Specific User-Agent and C2 server.

Relationship between APT actors and malware

Figure 21 shows the relationship between various APT actors and pieces of malware. As mentioned earlier, most malware that uses the KCP protocol is associated with APT41, and gokcpdoor is also suspected to be associated with this group. However, as described in the last section, we have found gokcpdoor along with malware used by the Tick actor, and for this reason we believe it is related to Tick. (For a summary of attribution, see Appendix 2.)

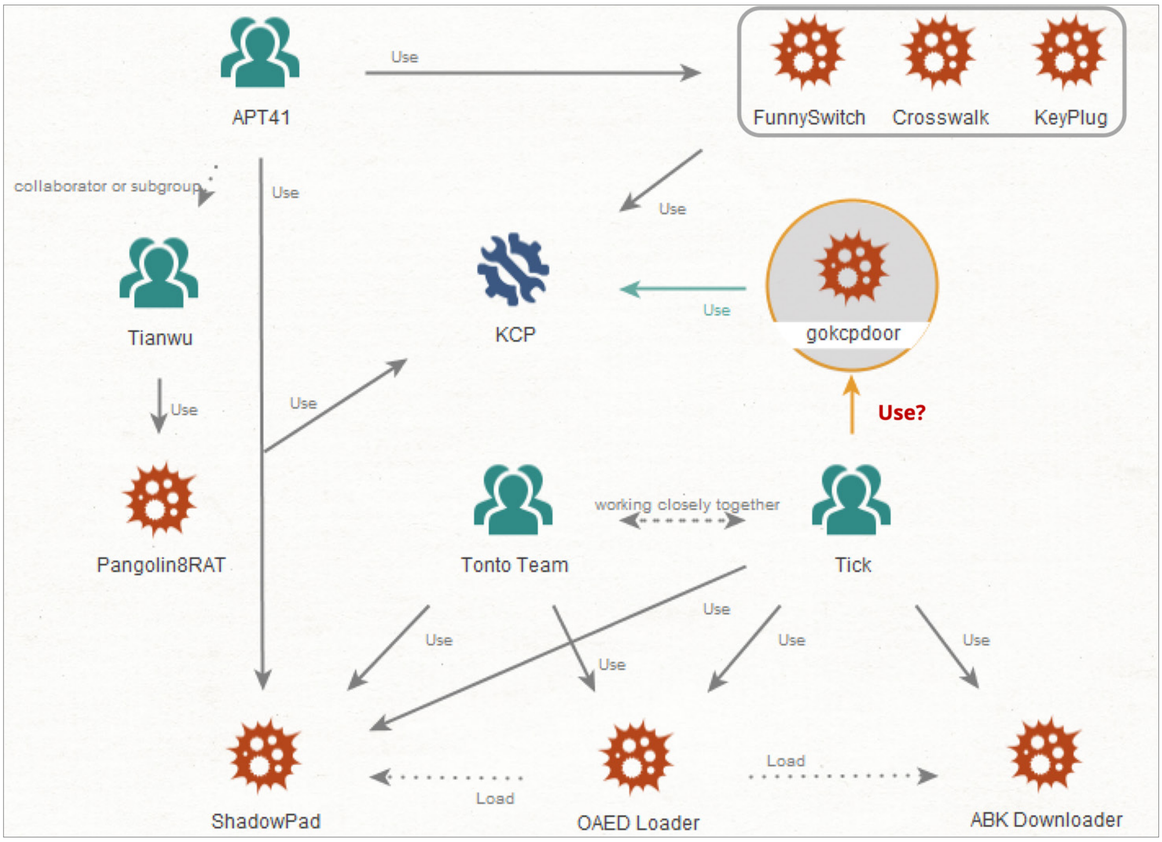

Figure 21: Overview of attribution (APT actors and malware).

COUNTERMEASURES

In this section we discuss measures that could be used to prevent and detect gokcpdoor and similar threats.

KCP traffic detection

KCP traffic can be difficult to identify. For this reason, it is important for network security products to check unknown UDP traffic. KCP traffic also has some characteristics that can be found using a *Splunk* query, as shown in Figure 22. (Please refer to Appendix 3 for *Splunk Steam* settings.) In addition, it is possible to analyse suspicious UDP traffic using KCP dissector [10], as shown in Figure 23.

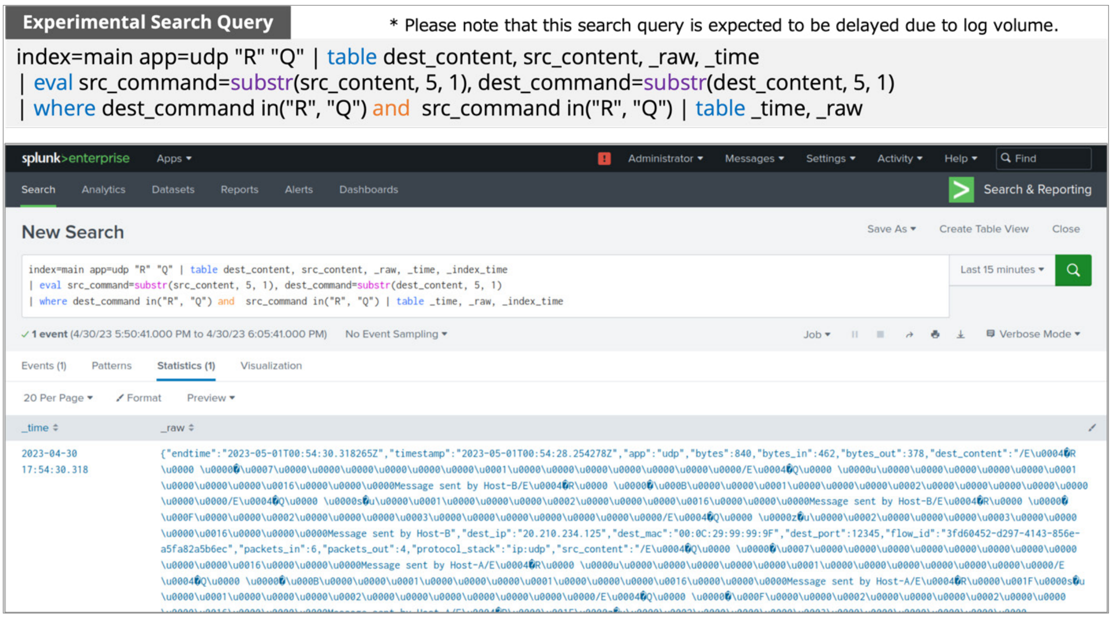

Figure 22: Searching for KCP traffic on Splunk.

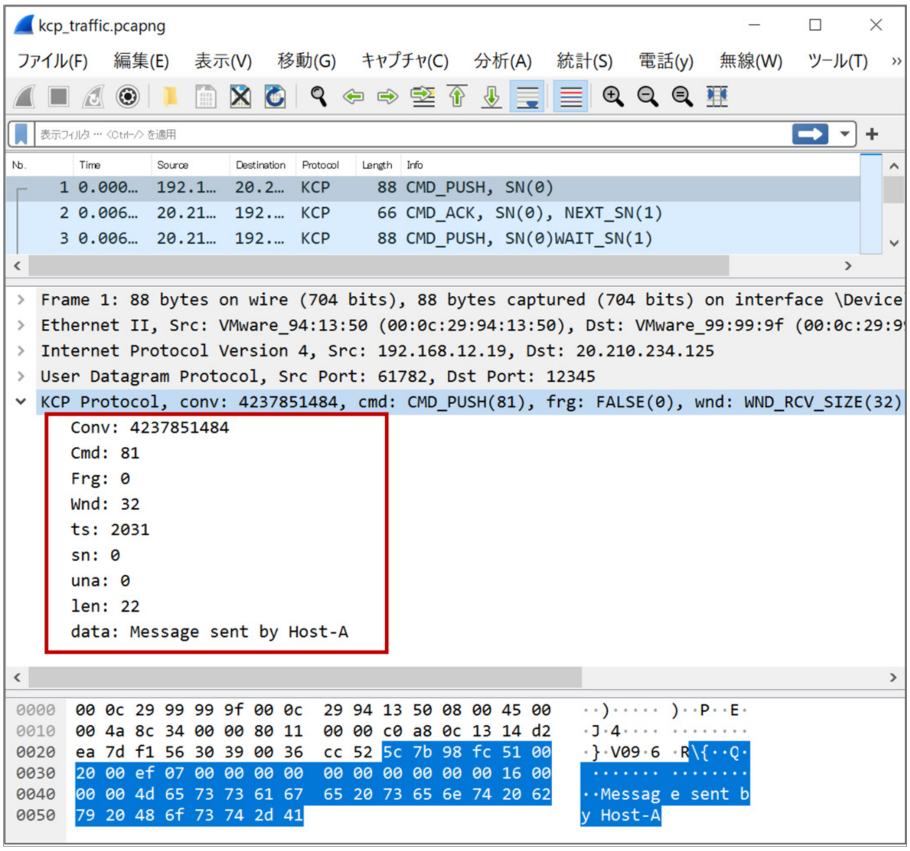

Figure 23: Example of using cfadmin-cn/kcp_dissector [10] for Wireshark to analyse suspicious UDP traffic.

Detection of gokcpdoor

Gokcpdoor can be detected and prevented using the following methods:

- Using a YARA rule (see Figure 24)
- Using *Autoruns* to check suspicious AutoStart Extensibility Points (ASEPs)[1]
- Using *Sysmon* to check the recording of Create Process and Network Connect events (Figure 25)
- Using EDR products to check execution of shell commands can be traced by process tree (Figure 26)

```
rule gokcpdoor {

meta:
   description = "Detects gokcpdoor malware"
   author = "LAC Co., Ltd."

    strings:
      $str1 = "gokcpdoor" ascii
      $str2 = "exec_lin.go" ascii
      $str3 = "exec_win.go" ascii
      $str4 = "syscmds/ps_linux.go" ascii
      $str5 = "syscmds/ps_windows.go" ascii
      $str6 = "target.go" ascii

    condition:
     (4 of ($str*)) and filesize > 2MB
}
```

* We recommend deliberate testing and tuning prior to implementation in any production system

Figure 24: Example YARA rule of gokcpdoor malware.

[1] In the case we analysed the APT actor had registered gokcpdoor as a service to implement persistence mechanisms.

```
Event SYSMONEVENT_NETWORK_CONNECT
    RuleName: -
    UtcTime: 2023-04-04 00:24:31.662
    ProcessGuid: {c6bde458-6e3f-642b-c0cb-5b0000000000}
    ProcessId: 26849
    Image: /home/test/Desktop/gokcpdoor
    User: -
    Protocol: udp
    Initiated: false
    SourceIsIpv6: true
    SourceIp: 0:0:0:0:0:0:0:0
    SourceHostname: -
    SourcePort: 0
    SourcePortName: -
    DestinationIsIpv6: true
    DestinationIp: 0:0:0:0:0:0:0:0
    DestinationHostname: -
    DestinationPort: 10054
    DestinationPortName: -
```

Figure 25: Example logs (Network Connect) of Sysmon Linux after gokcpdoor has been executed.

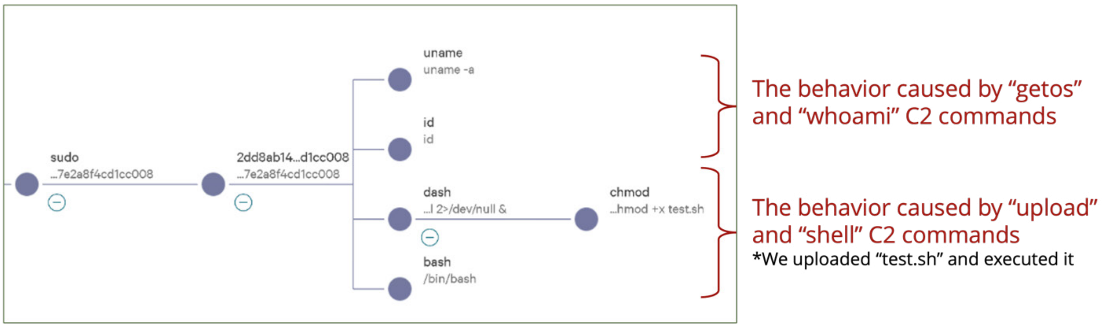

Figure 26: Example of CrowdStrike Falcon graphs process tree.

CONCLUSION

Gokcpdoor is a backdoor malware coded in Golang using KCP protocol for C2 communication. Attack vectors using the KCP protocol are on the rise and may be used more and more in the future.

We have shared some detection and prevention methods to protect against this and similar threats.

We have also suggested a possible relationship with the China-based APT actors Tick or APT41, but attribution is difficult.

We plan to continue to investigate APT actors using gokcpdoor and provide updated information that will help security researchers and defenders.

REFERENCES

[1] KCP. https://github.com/skywind3000/kcp.

[2] Kcp go. https://pkg.go.dev/github.com/xtaci/kcp-go.

[3] Positive Technologies. Higaisa or Winnti? APT41 backdoors, old and new. 14 January 2021. https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new/.

[4] Yeh, S.; Chang, L. The next-gen PlugX/ShadowPad? A dive into the emerging China-nexus modular trojan, Pangolin8RAT. Black Hat Asia 2022. https://i.blackhat.com/Asia-22/Thursday-Materials/AS-22-LeonSilvia-NextGenPlugXShadowPad.pdf.

[5] Kaspersky. PseudoManuscrypt: a mass-scale spyware attack campaign. 16 December 2021. https://ics-cert.kaspersky.com/publications/reports/2021/12/16/pseudomanuscrypt-a-mass-scale-spyware-attack-campaign/.

[6] Brown, R.; Ta, V.; Bienstock, D.; Ackerman, G.; Wolfram, J. Does This Look Infected? A Summary of APT41 Targeting U.S. State Governments. Mandiant. 8 March 2022. https://www.mandiant.com/resources/blog/apt41-us-state-governments.

[7] KCP.cs. https://github.com/qchencc/kcp-dotnet/blob/master/Source/Network/KCP.cs.

[8] nao_sec. An Overhead View of the Royal Road. 29 January 2020. https://nao-sec.org/2020/01/an-overhead-view-of-the-royal-road.html.

[9] Macnica Networks and TeamT5. APT Threat Landscape in Japan 2020. 21 May 2021. https://www.macnica.co.jp/business/security/manufacturers/files/mpressioncss_ta_report_2020_5_en.pdf.

[10] KCP dissector. https://github.com/cfadmin-cn/kcp_dissector.

APPENDIX 1: OSS LIBRARY LISTS

Table 5 lists the Golang OSS libraries used by gokcpdoor.

OSS Libraries (GitHub)	Description
klauspost/Reedsolomon	Provides Reed-Solomon Erasure Coding
klauspost/cpuid	Gets information about related CPU
templexxx/cpu	Gets information about related CPU
templexxx/xorsimd	Provides XOR code engine
pkg/errors	Provides simple error handling primitives
tjfoc/gmsm	Provides Chinese cryptographic algorithm
txthinking/x	Provides some network utilities function
txthinking/runnergroup	Ends concurrency reliably
patrickmn/go-cache	Provides in-memory cache function
xtaci/kcp-go	Provides KCP connection
	Provides KCP session implemented by UDP
txthinking/socks5	Provides SOCKS5 implemented for client
	Provides UDP support for SOCKS5
BishopFox/Sliver	Provides API for finding and listing processes
	Provides 'netstat' command function
digibib/tcpforward	Provides forward TCP traffic
1lann/udp-forward	Provides forward UDP traffic

Table 5: List of OSS libraries.

APPENDIX 2: DIAMOND MODEL

Figure 27 shows the Diamond Model for the gokcpdoor campaign.

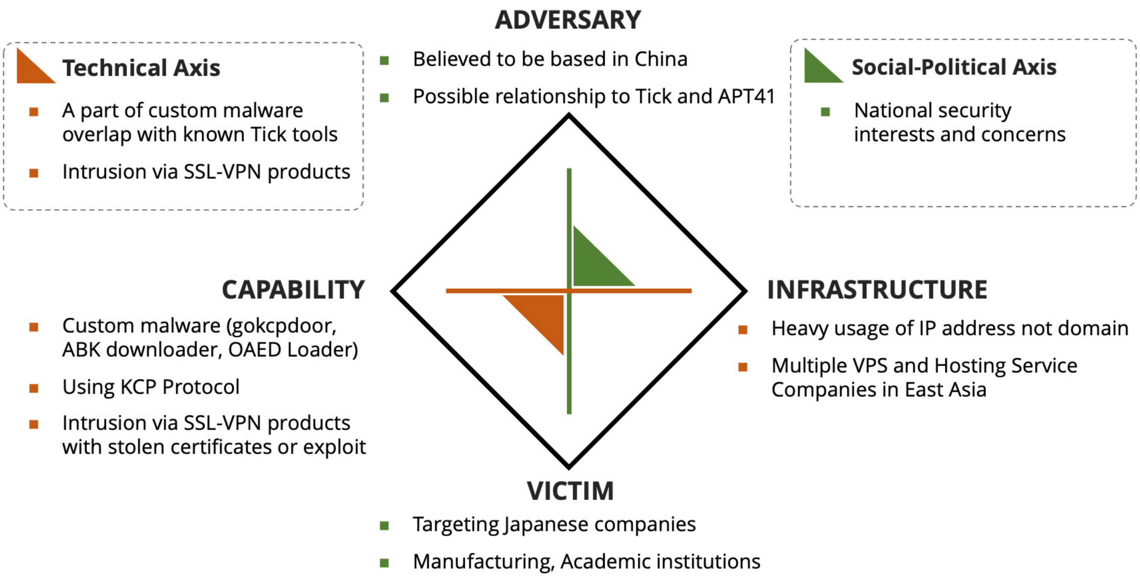

Figure 27: Diamond Model of this campaign.

APPENDIX 3: SPLUNK STREAM SETTINGS

To search KCP traffic within *Splunk*, you need to enable UDP traffic capture and content recording in the *Splunk Stream* app, as shown in Figure 26. We recommend estimating the amount of logs before setting these up in production.

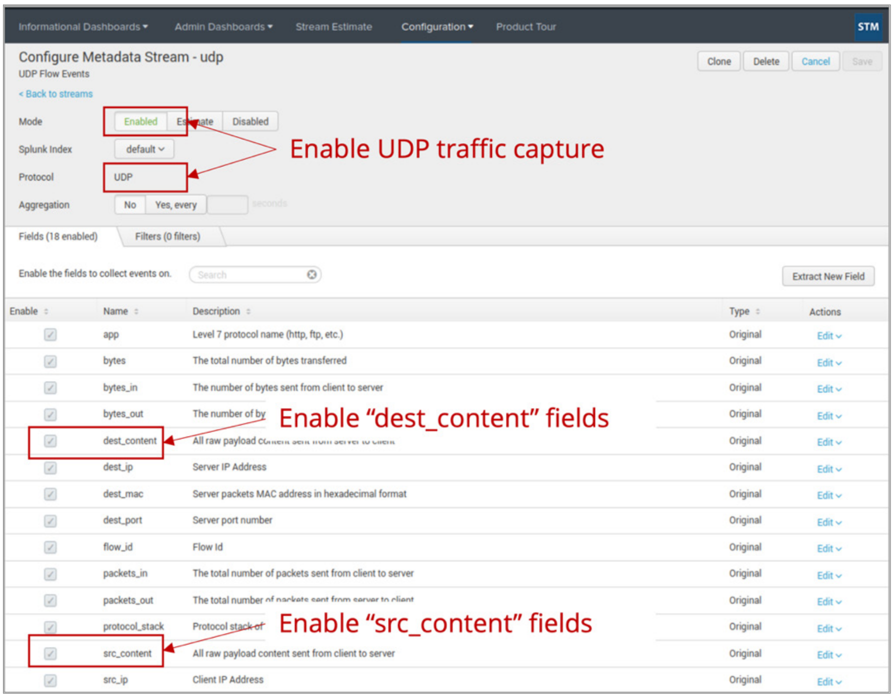

Figure 26: Splunk Stream settings.

APPENDIX 4: INDICATORS OF COMPROMISE (IOCs)

The following files were analysed for this paper.

Indicator	Type	Context
86f02e9f344a8e8009e59ecae934a780	MD5	ABK Downloader
d85c9b3d49b1af482c384a4253c16e28ae65a0f5	SHA1	
61eb25a6e6457087232de7ce7cd7b6cd9926e10674487c9e55b9a3fa54748b4c	SHA256	
Mozilla/4.0(compatible;MSIE8.0;WindowsNT6.0;Trident/4.0)	User-Agent	
a6f4a5ec66b7c5f275e793be02885543	MD5	gokcpdoor for Linux
bdb3db1013b16cb64b3f8156eae621054fa334bf	SHA1	
2dd8ab1493a97e0a4416e077d6ce1c35c7b2d8749592b319a7e2a8f4cd1cc008	SHA256	

Table 6: Samples related to this campaign.