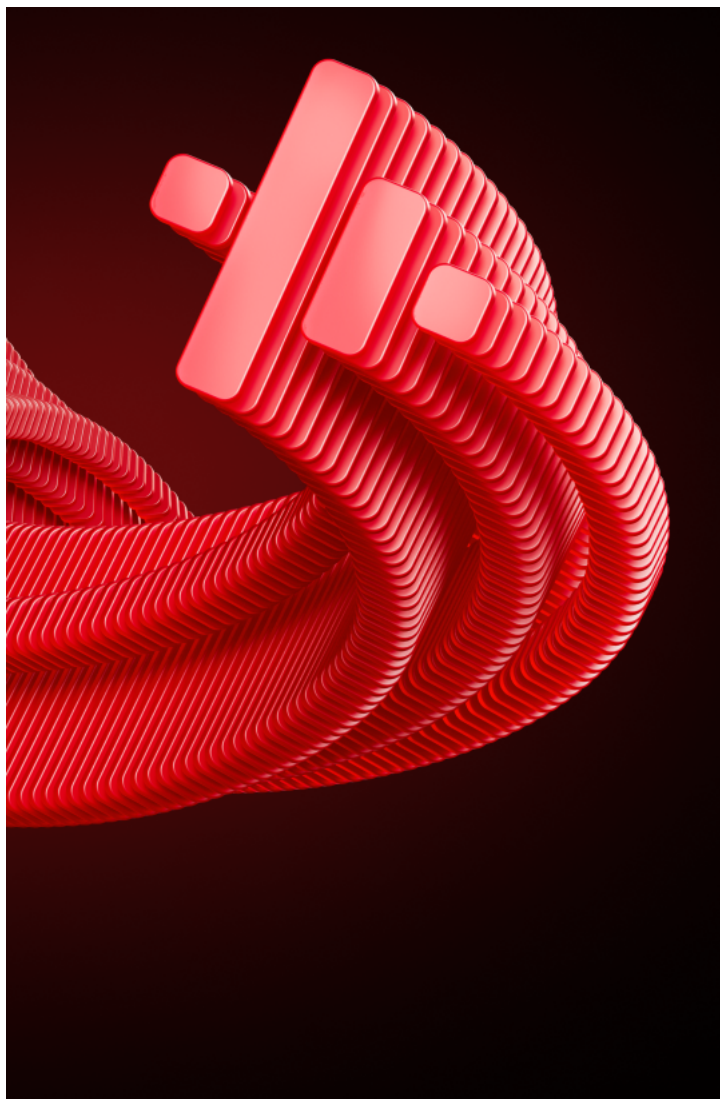


ExCobalt: GoRed, the hidden-tunnel technique

 global.ptsecurity.com/analytics/pt-esc-threat-intelligence/excobalt-gored-the-hidden-tunnel-technique



Contents

Introduction

While responding to an incident at one of our clients, the PT ESC CSIRT team discovered a previously unknown backdoor written in Go, which we attributed to a cybercrime gang dubbed ExCobalt.

ExCobalt focuses on cyberespionage and includes several members active since at least 2016 and presumably once part of the notorious Cobalt gang. Cobalt attacked financial institutions to steal funds. One of Cobalt's hallmarks was the use of the CobInt tool, something ExCobalt began to use in 2022.

Over the past year, the PT ESC has recorded attacks linked to ExCobalt and investigated related incidents at Russian organizations in the following sectors:

- Metallurgy
- Telecommunications
- Mining
- Information technology
- Government
- Software development

This article discusses ExCobalt's new tool, **GoRed**, how it evolved, and some of the tactics, techniques, and procedures that the group has used in its attacks.

The investigation begins

While investigating an incident recorded in March 2024 on one of our client's Linux hosts, we discovered a file named **scron**d, compressed with UPX (Ultimate Packer for eXecutables).

The data in an unpacked sample, written in Go, included package paths containing the substring **"red.team/go-red/"**. This fact suggests that this sample is a proprietary tool called **GoRed**.

After investigating the site, we were unable to find any significant links to malicious activity. Therefore, we can assume that the **red.team** domain found in the **GoRed** strings is a local repository with penetration testing tools, no longer actively used by its creators. All the information dates back to 2019, and the design is typical of many similar sites.

Figure 1. Internal packages

As for the **GoRed** backdoor, the following key features can be identified:

- Operators can connect to the backdoor and execute commands, similar to other C2 frameworks like Cobalt Strike or Sliver.
- **GoRed** uses the RPC protocol to communicate with its C2 server.
- Operators use DNS/ICMP tunneling, WSS, and QUIC to communicate with **GoRed**.
- **GoRed** can obtain credentials from compromised systems.
- It collects various types of information from compromised systems: details of active processes, host names, lists of network interfaces, file system structures, and so on.
- Operators use a variety of commands to conduct reconnaissance on the victim's network.
- **GoRed** serializes, encrypts, archives, and sends data it collects to a special server dedicated to storing compromised data.

For a complete technical description of **GoRed**, see the further section **"GoRed analysis"**.

First version of GoRed and other malicious tools we found

As we analyzed **GoRed**, we found that we had already come across several versions of the backdoor while responding to incidents at several of our clients earlier. For example, in July 2023, as we were investigating an incident at a certain company, we discovered several different tools inside the attackers' directories due to an error they had made, one of tools being the original version of **GoRed**.

Similarly, in an October 2023 incident, we found further tools inside public directories on the attackers' network. A short name and description of each tool we found during the investigations are presented in the table below.

2586	CVE-2022-2586: a Linux kernel vulnerability associated with pointer abuse in the <i>io_uring</i> function. This was used by the threat actor to escalate privileges and execute arbitrary code in the vulnerable system
3156.zip	CVE-2021-3156: a sudo vulnerability known as Baron Samedit. This allows a local user to execute arbitrary code with root privileges, bypassing standard security controls
4034	Pwnkit is a local privilege escalation exploit for CVE-2021-4034. This is a vulnerability in the pkexec utility. It allows local users to escalate their privileges to root, which can lead to complete compromise of the system
traceme	<u>Local privilege escalation exploit for CVE-2019-13272</u> . This is a vulnerability in the Linux kernel, in the ptrace component. It allows users with CAP_SYS_PTRACE privileges to escalate their privileges to root to execute arbitrary code in the system
bitrix.zip	This is an archive with vote_agent.php and html_editor_action.php files, containing RCE exploits for Bitrix. It was distributed via Telegram channels in June 2022, and also mentioned in connection with the May 2023 mass defacement of Russian websites (https://habr.com/ru/companies/ruvds/articles/739898/)
col	First version of the GoRed backdoor: 0.0.1
fs	fscan (https://github.com/shadow1ng/fscan/)
get_wp-commentin.txt	Obfuscated PHP file
run	FreeBSD file. Runs the specified executable file as root, same as the command sudo exec < specified file>
set	FreeBSD file. Sets root as file owner and grants rwx (read, write, execute) permissions to it, same as the commands chown < file> 0; chmod < file> 777
install	Installs a malicious module for Apache 2.4
install[drop]	Contains a dropper with a UPX-compressed binary file that is written to <i>/usr/local/games/w</i> . The file's privileges are then changed: chown root:root; chmod 4755

install[drop][upx] /usr/local/games/w	Tries to run the following command as root: (setuid(0); setgid(0); /bin/sh -c...)
k	Contains the following set of basic network utilities and replaces the process name with [kthr]:
curl	Stripped down version of curl
socksd	Proxy (socks4/5 with user+pass support)
shell	Bind shell
host	Resolves a host's name via the 8.8.8.8 DNS server
hash	MD5/SHA-1/SHA-256
gz	Inflate/deflate functionality
netcat	-
kit	Shell script to install kitsune . The MEGATSUNE variable was used instead of KITSUNE. amd64.rpm-bin.link and pkg.dpkg-source.info were used as the C2s
lock.zip	Part of the locker wiper ransomware repository with a read.me file in Russian inside
locker	Locker from lock.zip without the configuration files
m	Mettle (https://github.com/rapid7/mettle)
rev	Bind shell, shares code with shell from k , renames process to [kr]
sf	Binary for BSD
spark	Spark RAT
w.txt	PHP web shell: WSO (https://github.com/ndbrain/WSO)
wef	Variant of reverse SSH client (https://github.com/NHAS/reverse_ssh)
y.txt	PHP shell: p0wny-shell
knife/k	We found this file several months after the above-mentioned k , also in a public directory but in a different domain. It was also named k , and the functionality is similar, which suggests that this is simply the next version. This is a tool to control a compromised Linux server. It has extensive functionality; commands to execute are specified in the command line:
sync	No handler. Causes segfault if run
curl < url>	Loads content from specified URL
socksd	Starts a SOCKS server
netcat	No handler. Causes segfault if run
host < host>	Tries to resolve the host's name via the 8.8.8.8:53 DNS server
hash < filename>	Calculates the file's hash (MD5, SHA-1, SHA-256 are available)
gz < filename>	Compresses the file
shell < options> < IP address:port>	Remote shell. The command can be run with the following possible options: -l (listen to the specified address) or -c (connect to the specified address)

In addition to the tools obtained from the threat actors' server directories, ExCobalt used the following tools:

- Mimikatz
- ProcDump
- SMBExec
- Metasploit
- rsocx

Modified versions of standard utilities

Besides the above tools found in public directories, we came across modified versions of standard Linux utilities in several incidents, something we first saw in November 2023.

The modification of utilities serves two purposes:

1. Modified **ps** and **htop** hide an active **core** process of the **gsocket** module and related processes in terminal output.

Let's take **ps** as an example.

The *simple_readproc* function was modified.

The original code looks as follows:

Figure 2. Original **ps** code

The following code was added:

Figure 3. Code added to **ps**

The *is_target_proc* function checks for every process whether its name is equal to any of the hardcoded names of malicious processes:

Figure 4. The check inside *is_target_proc*

If the check returns *true*, the process will be hidden in the terminal output. Here's the list of the names of malicious processes hidden this way:

Figure 5. Names of malicious processes

2. Modified **ss** and **netstat** hide the active network connection of the **core** module of **gsocket** in terminal output.

Let's take **netstat** as an example.

The first code modification was made in the *main* function.

The original code looks as follows:

Figure 6. Original code of the *main* function of **netstat**

First, code was added to copy all C2s from the data section to a global structure:

Figure 7. Copying the C2s to a global structure

Second modification was in the *tcp_do_one* function, starting here:

Figure 8. Original code of the *tcp_do_one* function of **netstat**

Checks for the name of the malicious process and malicious connection were added to the code. If the check returns *true*, the process with one of the hardcoded names or network connections will be hidden:

Figure 9. Checking for the name of the malicious process and malicious connection

Relation to ExCobalt

In November 2023, we discussed [ExCobalt's attacks on Russian companies](#).

In that report, we mentioned the domain `lib.rpm-bin.link`, where upon directory enumeration we found many of the tools described above—including **col**, the first version of **GoRed**.

Also, in a March 2024 incident, we observed infected hosts that contacted the threat actor's servers: `get.rpm-bin.link` and `leo.rpm-bin.link`. Additionally, **GoRed** used a *static_TransportConfig* structure with the following C2s:

- `leo.rpm-bin.link`
- `sula.rpm-bin.link`
- `lib.rest`
- `rosm.pro`

In May 2023, [Bi.Zone researchers released an analysis of attacks by Sneaking Leprechaun](#), whose tools showed some overlap with the above-mentioned files found inside the public directories.

Furthermore, our colleagues at Rostelecom-Solar in May 2024 released a report on the Shedding Zmiy activity cluster, which also correlated with ExCobalt. [Case 7 in this report](#) described the same attack and a **GoRed** stealer sample with a C2 at `pkg.collect.net.in`. This sample was designated as Bulldog Backdoor in the report.

GoRed analysis

Before we proceed to analyzing the current version of **GoRed**, we will provide a retrospective analysis of its evolution.

Versions we found

All the versions we found are shown in the table below.

Version	Description
0.0.1	- Assumed to be the first one. - Collects information about the victim. - Source obfuscated with garble.
0.0.9	- Debug build: GoRed activity logging enabled. - Built-in configuration expanded. - Collects information about the victim. - Source obfuscated with garble.
0.0.9	- Built-in configuration expanded. - Collects information about the victim. - Source obfuscated with garble.
0.0.13	- Removed garble obfuscation. - Added control flow based on a command line interface. - Added reverse shell functionality via the WSS and DNS protocols. - Collects only processes. - Added a configuration structure absent in previous versions.
0.0.23-10-g4528ef3	- Added collection of network interfaces. - Added beacon mode (gecko command). - Added protocols for an operator to connect via DNS, ICMP, QUIC, WSS. - Added CBOR codec for RPC support. - Added proxy mode. - Implemented updating functionality. - Added file system monitoring (birdwatch command). - Added ICMP tunneling.
0.1.3-4-g68c293d	This version is mentioned only in the Solar report; we have not come across it.
0.1.3-62-g4843e53	- Expanded victim data collection feature (collector command). - Added transport configuration.
0.1.4	No changes found from previous version.

This article examines the current version, **0.1.4**.

Internal packages

First, we will describe the structure of the internal packages and their purpose to give you an understanding of **GoRed** functionality. In the backdoor's data, we have found the following package paths containing the substring "**red.team/go-red/**":

Package	Purpose
red.team/go-red/config/	Retrieval of the internal and transport configurations
red.team/go-red/bb/	Processing of an operator's commands
red.team/go-red/birdwatch/	Monitoring of the file system
red.team/go-red/gecko/	Protocol for communication between GoRed and its C2
red.team/go-red/backend/	Connecting to a data exfiltration server
red.team/go-red/collector/	Collecting system information
red.team/go-red/util/	Various auxiliary utilities
red.team/go-red/packer/	Data packing
red.team/go-red/proxy/	Proxy mode operation
red.team/go-red/revshell/	Reverse shell mode operation
red.team/go-red/dns/	DNS tunneling implementation
red.team/go-red/icmptunnel/	ICMP tunneling implementation

Let's now proceed to analysis proper. Here's a simplified diagram of the control flow to give you an understanding of how it works.

Figure 10. Diagram of the control flow

1. Start of execution

The control flow relies on the **command line interface (CLI)**. However, before passing control to the CLI, the backdoor initializes a number of commands, which are described below.

- **service** command
- **gecko** subcommand

Figure 11. Initializing the command and subcommand

The **service** command, intended for gaining persistence in the system, is initialized first. The **GoRed** CLI command structure looks as follows:

```
struct cli_Command
{
    string Name;
    _slice_string Aliases;
    string Usage;
    string UsageText;
    string Description;
    string ArgsUsage;
    string Category;
    PTR_cli_BashCompleteFunc BashComplete;
    PTR_cli_BeforeFunc Before;
    PTR_cli_AfterFunc After;
    PTR_cli_ActionFunc Action;
    PTR_cli_OnUsageErrorFunc OnUsageError;
    _slice_ptr_cli_Command Subcommands;
    _slice_cli_Flag Flags;
    cli_FlagCategories flagCategories;
    bool SkipFlagParsing;
    bool HideHelp;
    bool HideHelpCommand;
    bool Hidden;
    bool UseShortOptionHandling;
    string HelpName;
    _slice_string commandNamePath;
    string CustomHelpTemplate;
    cli_CommandCategories categories;
    bool isRoot;
    cli_separatorSpec separator;
};
```

In terms of command identification, the most helpful fields in this structure are the following:

- *Name*: command name
- *Usage*: command description
- *Action*: function called when the command is executed
- *Subcommands*: subcommands for the current command

Next, the structure of the CLI itself is initialized into the variable *app*.

Figure 12. Initializing the *app* structure for the CLI

The *app* structure looks as follows:

```

struct cli_App
{
    string Name;
    string HelpName;
    string Usage;
    string UsageText;
    string ArgsUsage;
    string Version;
    string Description;
    string DefaultCommand;
    _slice_ptr_cli_Command Commands;
    _slice_cli_Flag Flags;
    bool EnableBashCompletion;
    bool HideHelp;
    bool HideHelpCommand;
    bool HideVersion;
    cli_CommandCategories categories;
    cli_FlagCategories flagCategories;
    PTR_cli_BashCompleteFunc BashComplete;
    PTR_cli_BeforeFunc Before;
    PTR_cli_AfterFunc After;
    PTR_cli_ActionFunc Action;
    PTR_cli_CommandNotFoundFunc CommandNotFound;
    PTR_cli_OnUsageErrorFunc OnUsageError;
    PTR_cli_InvalidFlagAccessFunc InvalidFlagAccessHandler;
    time_Time Compiled;
    _slice_ptr_cli_Author Authors;
    string Copyright;
    io_Reader Reader;
    io_Writer Writer;
    io_Writer ErrWriter;
    PTR_cli_ExitErrHandlerFunc ExitErrHandler;
    map_string_interface_ Metadata;
    PTR_func_map_string_string ExtraInfo;
    string CustomAppHelpTemplate;
    string SliceFlagSeparator;
    bool DisableSliceFlagSeparator;
    bool UseShortOptionHandling;
    bool Suggest;
    bool AllowExtFlags;
    bool SkipFlagParsing;
    bool didSetup;
    cli_separatorSpec separator;
    _ptr_cli_Command rootCommand;
};

```

Here, too, the most informative fields for identifying commands are as follows:

- *Name*: current command name
- *Action*: function called
- *Commands*: subcommands for the current command

The *Commands* field for the *app* structure is also initialized.

Figure 13. Initializing the CLI context structure

Next, the *Logging* field is retrieved from the *embedded_Config* structure described in the section below. After this, the control flow moves to the CLI.

Figure 14. Transferring control to the CLI

2. Gaining persistence in the system

The first command to be executed is **service**. It achieves persistence in the system. It can be executed with the following options:

Option	Purpose
no-service	Simply proceeds to executing CLI commands
uninstall	Removes the service
restart	Restarts the service

If there are no options from the table above, it gains persistence as a service with the name it received as an argument. To maintain presence in the compromised system, it creates environment variables whose names begin with "BB", for example:

- BB_WS
- BB_QUIC
- BB_ICMP
- BB_DNS
- BB_START_DELAY

3. Initializing beacon mode

The control flow is then transferred to the **gecko** command. This command is the entry point for **GoRed** to run in beacon mode.

Figure 15. Execution options for the **gecko** command

It can be executed with the following options:

Option	Purpose
wss	Use the corresponding protocol for communication between the operator and GoRed
quic	
icmp	
dns	
background	Run the command in the background
start-delay	Add delay in communications with the C2

Depending on the protocol received as an option, the command fetches the C2 from the transport configuration whose structure is described in the section below. It then begins initializing the beacon functionality.

Figure 16. Initializing the beacon functionality

To identify the victim, the malware first generates an ID, which is an MD5 hash of the computer's MAC addresses and name, similarly to the case described in our [Hellhounds: Operation Lahat](#) article. The resulting hash sum is added to a field in the *client* structure, which stores all data required for communication with the C2.

Figure 17. Obtaining a victim ID

4. Establishing an initial connection

After **GoRed** is initialized, it needs to connect to its C2. The number of connection attempts is defined in the **backoff** package.

Figure 18. Setting the number of C2 connection attempts

The execution flow calls the function that registers the beacon functionality, after which the CLI commands are initialized.

Figure 19. Starting beacon functionality

Registration uses the RPC protocol. Data in the *model_Beacon* structure is sent to the server, and data in the *model_Auth* structure is used for authentication with the server.

```
struct model_RegisterBeaconRequest
{
    model_Beacon Beacon;
    model_Auth Auth;
};

struct model_Beacon
{
    string ID;
    string Hostname;
    string ClientIPs;
    _slice_string Tags;
    string OS;
    string Username;
};

struct model_Auth
{
    string Token;
    uuid_UUID ClientID;
    string Transport;
};
```

The fields in these structures have the following purposes:

Field	Purpose														
Beacon	The structure containing victim information														
	<table><tr><th>Field</th><th>Purpose</th></tr><tr><td>ID</td><td>Victim's ID</td></tr><tr><td>Hostname</td><td>Victim's hostname</td></tr><tr><td>ClientIPs</td><td>Victim's IP addresses</td></tr><tr><td>Tags</td><td>Victim's tag</td></tr><tr><td>OS</td><td>Victim's operating system</td></tr><tr><td>Username</td><td>Victim's username</td></tr></table>	Field	Purpose	ID	Victim's ID	Hostname	Victim's hostname	ClientIPs	Victim's IP addresses	Tags	Victim's tag	OS	Victim's operating system	Username	Victim's username
Field	Purpose														
ID	Victim's ID														
Hostname	Victim's hostname														
ClientIPs	Victim's IP addresses														
Tags	Victim's tag														
OS	Victim's operating system														
Username	Victim's username														
Auth	The structure containing authentication data														
	<table><tr><th>Field</th><th>Purpose</th></tr><tr><td>Token</td><td>Authentication token</td></tr><tr><td>ClientID</td><td>Client ID</td></tr><tr><td>Transport</td><td>Protocol being used</td></tr></table>	Field	Purpose	Token	Authentication token	ClientID	Client ID	Transport	Protocol being used						
Field	Purpose														
Token	Authentication token														
ClientID	Client ID														
Transport	Protocol being used														

After registering, **GoRed** runs the **birdwatch** command to monitor the file system.

Figure 20. Running **birdwatch** in the background

The execution flow then sets a **GoRed** beacon mode heartbeat period (for signaling to C2).

Figure 21. Setting a heartbeat period

The execution flow then runs a command to monitor the password file stored in `/etc/shadow/`.

Figure 22. Running **creds-watcher** in the background

Finally, the execution flow initializes all available commands and goes into heartbeat mode.

Figure 23. Entering heartbeat mode

5. Entering listen and execute mode

At the final stage of initialization, **GoRed** starts listening for the operator's commands that it previously initialized. It can execute both system and built-in commands. Commands can be set to run in the background.

Figure 24. Function to set a command to run in the background

6. Communications in beacon mode

GoRed uses the RPC protocol to communicate with its C2 in beacon mode.

Figure 25. RPC functionality

It registers a custom codec to communicate via RPC.

Figure 26. Custom RPC codec functionality

The registered codec serializes data with CBOR and encrypts with AES-256-GCM (*Secret* field in *embedded_Config*) when sending, and does the reverse when receiving.

Figure 27. Function that transforms data for exfiltration

Configurations

GoRed contains two configuration blocks: built-in and transport.

Built-in configuration

This is the configuration of **GoRed** itself. It is encoded in Base64 and serialized with **msgpack**.

Figure 28. Built-in configuration

For versions **0.0.23-10-g4528ef3** through **0.1.4**, the structure of the built-in configuration is as follows:

```
struct embedded_Config
{
    string Logging;
    string Token;
    uuid_UUID UserID;
    uuid_UUID ClientID;
    string ClientKey;
    string Version;
    _slice_string Tags;
    _slice_string Args;
    string Secret;
    _ptr_url_URL BackendAddress;
    _ptr_url_URL ProxyAddress;
};
```

The purposes of the built-in configuration fields are as follows:

Field	Purpose
Logging	Logging and log format: - No logging - To a .log file in a temporary directory - Directly to the operator's console
Token	Generated JWT for RPC
UserID	UUID for the <i>payload</i> field in the JWT (when using RPC)
ClientID	Unique identifier for the exfiltrated data of the victim
ClientKey	HS256 key needed to generate the JWT when exfiltrating data
Version	GoRed version
Tags	Victim's tag
Args	GoRed arguments
Secret	AES-256-GCM key for encrypting or decrypting the data transmitted or received via RPC
BackendAddress	Address of a dedicated server for data exfiltration
ProxyAddress	List of proxy addresses for data exfiltration

Configuration structure for versions **0.0.9** through **0.0.13**.

```
struct embedded_Config
{
    bool Debug;
    string Token;
    uuid_UUID ClientID;
    string ClientKey;
    string Version;
    _slice_string Tags;
    string Secret;
    string BasicAuthLogin;
    string BasicAuthPass;
    _ptr_url_URL BackendAddress;
    _ptr_url_URL ProxyAddress;
};
```

The fields in this version of the built-in configuration have the following purposes:

Field	Purpose
Debug	Logging and log format: • No logging • To a .log file in a temporary directory • Directly to the operator's console
BasicAuthLogin	Login for authentication when using the curl command
BasicAuthPass	Password for authentication when using the curl command

Built-in configuration for version **0.0.1**.

```
struct embedded_Config
{
    bool Debug;
    uuid_UUID ClientID;
    string ClientKey;
    string Version;
    _ptr_url_URL BackendAddress;
};
```

Golang script for getting built-in configuration fields:

```

package main

import (
    "encoding/base64"
    "fmt"
    "net/url"

    "github.com/google/uuid"
    "github.com/vmihailenco/msgpack/v5"
)

type embedded_Config struct {
    Logging      string
    Token        string
    UserID       uuid.UUID
    ClientID     uuid.UUID
    ClientKey    string
    Version      string
    Tags         []string
    Args         []string
    Secret       string
    BackendAddress* url.URL
    ProxyAddress* url.URL
}

const config = `...`

func main() {
    var item map[string]any
    data, _ := base64.StdEncoding.DecodeString(config)
    err := msgpack.Unmarshal(data, &item)
    if err != nil {
        panic(err)
    }
    fmt.Print("Logging: ")
    fmt.Println(item["Logging"])
    fmt.Print("Token: ")
    fmt.Println(item["Token"])
    fmt.Print("UserID: ")
    fmt.Println(uuid.UUID(item["UserID"].([]byte)))
    fmt.Print("ClientID: ")
    fmt.Println(uuid.UUID(item["ClientID"].([]byte)))
    fmt.Print("ClientKey: ")
    fmt.Println(item["ClientKey"])
    fmt.Print("Version: ")
    fmt.Println(item["Version"])
    fmt.Print("Tags: ")
    fmt.Println(item["Tags"])
    fmt.Print("Args: ")
    fmt.Println(item["Args"])
    fmt.Print("Secret: ")
    fmt.Println(item["Secret"])
    fmt.Print("BackendAddress: ")
    fmt.Println(string(item["BackendAddress"].([]byte)))
    fmt.Print("ProxyAddress: ")
    fmt.Println(item["ProxyAddress"])
}

```

Python script for getting built-in configuration fields:

```

import msgpack

import base64

s = base64.b64decode('...')

config = msgpack.unpackb(s, raw = False)

print(config)

```

Transport configuration

The transport configuration looks as follows:

Figure 29. Transport configuration

It has the following structure:

```
struct static_TransportConfig
{
    static_Transport Revsh;
    static_Transport RPC;
    static_Transport Proxy;
};

struct static_Transport
{
    _ptr_static_Address WS;
    _ptr_static_Address QUIC;
    _ptr_static_Address ICMP;
    _ptr_static_Address DNS;
    _ptr_static_Address TCP;
};

struct static_Address
{
    string Domain;
    string BackupIP;
    signed __int64 Port;
    string Proto;
};
```

The fields of the **static_TransportConfig** structure have the following purposes:

Field	Purpose
Revsh	Reverse shell connection addresses for an operator
RPC	Addresses for GoRed beacon mode RPC heartbeats
Proxy	Addresses for running GoRed in proxy mode

The fields of the *static_Address* structure have the following purposes:

Field	Purpose
Domain	Domain to connect to
BackupIP	IP to connect to if the domain cannot be resolved
Port	Connection port
Proto	Connection protocol

Communication protocols

GoRed has several protocols for communicating with the operator.

Protocol	Implementation
ws	Implements WebSocket connection
quic	Implements QUIC connection
icmp	Implements ICMP tunneling
dns	Implements DNS tunneling

DNS

DNS tunneling in **GoRed** can use Base64 or Base32. This option is selected during compilation.

Figure 30. Using Base32 for traffic tunneling

An example of a domain used in an attack is **8E1A4QB4OGA66RPJCHL72DJGCKRMIOR8CDN3EDJBDOOAEQ3FEDQ5UQB4OGA66RP.JCHL6EDJGCKRMIOR8CDN3EDJBDLJG.rosml**

Background commands

Background commands run continuously; some of them can be added to the background or removed, depending on the conditions in the table below.

Command	Description
birdwatch	Watches for new files inside directories. Runs in the background by default. • list: subcommand to print a list of paths monitored for new files. • unwatch: subcommand to stop monitoring paths for new files.
creds-watcher	Watches for passwords. Runs in the background by default.
revsh-host	Enables reverse shell mode. Runs in the background upon execution. • Sets WebSocket as the communication protocol between the operator and GoRed. • Sets QUIC as the communication protocol between the operator and GoRed. • Sets ICMP as the communication protocol between the operator and GoRed. • Sets DNS as the communication protocol between the operator and GoRed.
rev-proxy	Enables reverse proxy mode via SOCKS5. Runs in the background upon execution.
rev-fwd	Enables reverse port forwarding mode. Runs in the background upon execution.

Connecting in rev-proxy and rev-fwd mode

Before starting to act as a server, **GoRed** needs to initialize an embedded X.509 certificate, similarly to the case described in our [Hellhounds: Operation Lahat. Part 2](#) article.

Figure 31. Retrieving the certificate and host information

The backdoor also needs to collect information about the victim's host by executing the *CollectHostInfo* function shown above. This produces the structure presented below, except for the *Addr* field.

```
struct proto_HostInfo
{
    string Addr;
    string OS;
    string Username;
    string Hostname;
    _slice_string IPs;
};
```

The fields in the structure have the following purposes:

Field	Purpose
Addr	Address to connect to, passed to the command as a parameter
OS	Victim's operating system
Username	Victim's username
Hostname	Victim's hostname
IPs	Victim's IP addresses

A structure to identify **GoRed** as a server is initialized as follows:

Figure 32. Initializing the structure

The initialized structure looks as follows:

```
struct proto_BinInfo
{
    uuid_UUID ClientID;
    string Token;
    _slice_string Tags;
};
```

The fields in the structure have the following purposes:

Field	Purpose
ClientID	Identifies the victim
Token	Generated JWT for RPC
Tags	Victim's tag

Having received the *proto_HostInfo* and *proto_BinInfo* structures, **GoRed** uses their fields in a handshake message that it sends to the C2 at the address it gets from the transport configuration. The handshake message structure looks as follows:

```
struct proto_MsgGreeting
{
    proto_ConnectionMode Mode;
    string MachineID;
    proto_BinInfo BinInfo;
    _ptr_proto_HostInfo HostInfo;
};
```

The fields in the structure have the following purposes:

Field	Purpose
Mode	The following modes are supported: • 2: rev-proxy mode • 3: rev-fwd mode
MachineID	Victim's computer ID
BinInfo	Structure containing the GoRed configuration information
HostInfo	Structure containing victim information

The handshake sequence used to register **GoRed** as a server looks as follows:

Figure 33. Communications with the C2

In response to the handshake message, the server sends a message with the following structure:

```
struct proto_MsgGreetingResponse
{
    string Greeting;
    _ptr_proto_BinInfo BinInfo;
    _ptr_proto_HostInfo HostInfo;
};
```

If the *Greeting* field contains the string "welcome", the connection is considered successful, and **GoRed** starts running in server mode; if not, the connection cannot be established.

Issued commands

The operator uses the following commands to communicate with **GoRed**:

Command	Description
upload	Exfiltrates files. Takes a file path as the argument
download	Downloads files onto the infected computer. Takes a file path as the argument
bg-list	Displays a list of internal background commands
bg-stop	Cancels an internal background command. Takes a command ID as an argument. Can be used to stop all background commands

Command	Description
stealth	Automatically sets the heartbeat frequency within a larger range for greater stealth. on: subcommand that enables this mode with the time range 0×9D29229E000—0×105EF39B2000 off: subcommand that disables this mode with the time range 0×12A05F200—0×45D964B800
emit-period	Gets or sets a heartbeat frequency manually
conn-providers	Gets the communication protocols available in the current version of GoRed
info	Gets victim information: - UserID - Logging - Tags - Version - Time
collect	Collects and exfiltrates system information. See the section below for more details
bb-update	Updates GoRed . Sends a GET request to the URL passed as the argument and restarts GoRed with the restart argument
bb-ps	Gets the status of the process passed as the argument
bb-cat	Reads the file passed as the argument
bb-find	Searches for files passed as the argument
bb-ls	Displays the contents of the directory passed as the argument
bb-mkdir	Creates the directory passed as the argument
bb-pwd	Returns the full path of the current directory
bb-rm	Deletes the file passed as the argument
bb-wc	Collects information about the file passed as the argument: - Number of words - Number of lines - Number of characters - Number of bytes
bb-nmap	Scans the network. Takes a host IP as the argument
bb-ping	Pings an external host. Takes an IP as the argument
bb-wget	Gets files via HTTP. Takes two arguments: the source URL and the output filename specified after the —output option
bb-curl	Similar to curl , but with limited functionality

collect

Since this command collects system information and data for subsequent exfiltration, we decided to describe it in more detail:

Option	Purpose
local-archive	Selects a compression algorithm: tar or gzip
skip-trees	Skips collecting file system structure information
skip-files	Skips collecting files
exec-timeout	Sets a time period for collecting files

Collected information

An example of the code used to collect information is shown below.

Figure 34. Example of the code used to collect information

A complete list of collected information is provided in the table below.

File	Contents
------	----------

File	Contents
processes.json	List of processes
envvars.json	List of environment variables
host.json	Information about the processor, RAM, installed OS, user name, group name
network_interfaces.json	List of network interfaces
netstats.json	List of active network connections
*.txt	Files that will be collected depending on the values of the fields in the model_CollectionConfig structure
hardware.json	Hardware information
trees.json	File system structure

Exfiltration

Before the data is sent, it is serialized with **msgpack** and encrypted with AES-256-GCM (*Secret* field in *embedded_Config*).

Figure 35. Example of collected data preparation

Next, after archiving the data, the backdoor sends it with a POST request to a URL generated by appending `/api/collection-result` to the *BackendAddress* field in *embedded_Config*.

Figure 36. Data exfiltration

It is also possible to update the *model_CollectionConfig* structure by sending a GET request to a URL generated by appending `/api/config` to the *BackendAddress* field in *embedded_Config*.

\

Figure 37. Configuration update

The *model_CollectionConfig* structure is a configuration for the **collect** command and has the following fields:

```
struct model_CollectionConfig
{
    _slice__slice_string Commands;
    _slice_string Files;
    _slice_string TreePaths;
};
```

The fields in the structure have the following purposes:

Field	Purpose
Commands	bb_files_* command
Files	Files
TreePaths	Paths

Conclusion

ExCobalt continues to demonstrate a high level of activity and determination in attacking Russian companies, constantly adding new tools to its arsenal and improving its techniques. Not only is it developing new attack methods, but it's also actively improving its existing tools, such as the **GoRed** backdoor.

ExCobalt is apparently aiming for more sophisticated and productive methods of hacking and cyberespionage, seeing how **GoRed** has been acquiring new capabilities and features. These include expanded functionality for collecting victim data and increased secrecy both inside the infected system and in communications with C2 servers.

In addition, ExCobalt demonstrates flexibility and versatility by supplementing its toolset with modified standard utilities, which help the group to easily bypass security controls and adapt to changes in protection methods. The use of modified utilities is a sign that the members of the group have an in-depth understanding of the weaknesses of companies they attack, while leveraging vulnerabilities helps them to pursue sophisticated attacks on their targets.

Overall, the evolution of ExCobalt and its toolset, including **GoRed** and the modified utilities, highlights the need for organizations and cybersecurity professionals to continuously improve detection and protection techniques to combat this group as well as other similar cyberthreats.

Authors:

Vladislav Lunin, Senior Information Security Threat Researcher, PT ESC

Alexander Badayev, Information Security Threat Researcher, PT ESC

IOCs

File indicators

File	MD5	SHA-1
		GoRed
col	6ea3feb1888ce02e3d0d2857b5ef71c4	c5540ec2ec79a21f07b0d793cc36b0:
coll.exe	64db61efc8acf370b91110b6f93d4dce	a81373d92d798418109552fb91d4c4
63f6de3c86de55172b147b947f29c808.virus	63f6de3c86de55172b147b947f29c808	5a504869350a4bdbcd22b09dbe7b0
c.upx	d3cd9d9bad6450e8fd4fd2e972639c69	a190448a0c01a6e58610de27d022cc
cad5cb82baccd1f28e381e5c924f204a.virus	cad5cb82baccd1f28e381e5c924f204a	81861a853216f78219dd8cb0b4717d
bb.exe	6f6e7fe49a8d5696f389e202d3b8c7e2	1d784e6c7d12fb7730895f21e4bfd3c
coll.exe	b5dc9a67f76fa18784b51fd3c5b9607c	de243b57b087f5d1cde50db1949aa3
revsh.exe	caf68b393d56548074b9434564cb0625	680cb0a25e4a5148f5a1f7d3b75fad4
-	b747c05888caf380edf6b2baab142272	ef50067027e27bea188023fa6a8ce9f
dttcodexgigas.4f6164321d10c7a54a54398ccc7b11c1e7390e38	0385b0f83dbfc99c243ff066e3fe3cb2	4f6164321d10c7a54a54398ccc7b11c
7dc1e49f1664af70d85d31af70f29071	7dc1e49f1664af70d85d31af70f29071	1981f9a1d885c0ccb2d1f5910765a52
Scada.Connect.Win.ConnectionImpl.dll	fc3b7f47958f6c1c6a93a2f2f970734c	8030f2430234426ab3bdc8cdd995be
crond	fad11b841d84bbe33248719341b298d3	fd7532d2a42dd3ba26a1a3453698b8
sron	c02bee46d6a7a46f54eabe003fec897	58d03630792f287184177660d9fd841
scron	ad5c0363e7e28c69007f891fbc3dd030	3dd9bd38a8f8166b1af25cb523a9a6f
Utilities we found during our research		
/dev/shm/.gs-0/gsn	b3a07b9f99f8d36bda871b63d55afb01	7e3d46ce5aa7345d8b84e61453233f
check.zip	c1f3f6efb9ef18268eb3b841065e6554	ca9a2e18119ac348962e2112c66812
gsn_x86_64-alpine.tar.gz	e210c26d26a1395d9bc1de21fe1b2975	4ba1ae554f2cfeecf250ba5a258a4ff
xlswifi	376531d8a3a19016aa64d80dec23d980	0f621d371782f8e610c630f942a8951
xlswired	fcc1ad58da960c5780a66fcc24c6c2fa	928e4e776e82645fe14a53e2ad62b5
3156.zip	489fbca25049e5fab9dca10541e33214	91eeab83ddcd82a77804f2e5572d84
4034	a5fa43f822b6dd88298371232d49c597	1aa5b4deae98f707b0a529d97fd8e71
bitrix.zip	d08bef69aee69d91b8cd0315175f665c	ada92c3a38e227aa8d42b4886e036c
fs	89ae36448f1922870f1a09c29f17c775	3b1329e81739b1ea6acbb4ec4dff11f
get_wp-commentin.txt	46eb5fa7c75cc29d89f3e48be26bbd46	36ef757aa3eedc3ec22b56d60931c
install	848faa5839487c4331cb2a1146811f23	f67dbe68fc11139b719fec11784247cf
k	0cda2ee10f5b8e9a241ef3e7e352752d	6ffe11b31443bd9cef4928aa3f29b11c
kit	2cad1092a2828a33df2156a3a97d7cf1	27dd8d144d0ac3af9f4ad3df8a060d8

File	MD5	SHA-1
lock.zip	a2ff5b0bc0782560090574c992ccf995	97a3ead87af829f77dacfa23ab2786b
locker	d3064fe5d8a402b26099fcdbaeaedef1	f07e31056001ccc26be75772c9a2f3e
m	fbf3f02b37b10bde868fed9d7b750fd8	7c27d25dbc01958724fd55f0fadf966e
rev	9b6122f1b4f6513c22b50ef05e881f38	6559a9eda3b8164e0c8926b4b7178c
sf	bc421b337fc639749528f2e756239269	d75faee2f8ec90a69354a2c033f20e1
spark	76cc921e5b26a0720db213479bff1ea2	f640f70d1b65b0bfc8bcbf5261f3cdc8
w.txt	3500760bc3e69102e01d256637f5f4a8	6ff2821bef28476341b75b67d9c9f2df
wef	b7735e157273a013f26515f0c969b093	5e79ffbbafdddeb2d85c8fe835b07eed
y.txt	166a248f264fbf11998c86e8b384e47a	9de84bd7118dee80f5b309ddbc46dc

Modified legitimate utilities

ps	4a04baf3c65581bcd14fbaf58aa6860b	352a62abc61c93fdb08f6f4201326f14
netstat	83b8aa078be2a0a5ca0ebf1968989a4b	a16120cca64e0c9a73f02975691e46
ps	b7db832b2598c83b7b077ce603a3ff73	1af6946263f4f548ffc510c9f68378a4
ss	415d091f42fc62e8dfb6f8bb5ce641c5	ad6653a7ee1bcb9590f5da12cf46d8f
htop	cded33e0e37e14bbf7cac53c4e305ece	1fc930a59587fd9faf7536add47d92de
netstat	eda9ca5f9405b5e2d004a4ba5c0dcd16	1d4c0b3c74ddacf7459743cc60dd2a1
ps	d215a54c581ab62079389c852d9ef84f	7e0a4c53bf3dfcb08993231539986a2
ss	ffc418b222c08f3071ff53cde4acb22e	2683dcce7fc3886f8305030b128103f

Network indicators

pkg.collect.net.in
leo.rpm-bin.link
lib.rest
45.146.7.26
rosm.pro
193.37.71.75
sula.rpm-bin.link
45.146.7.16
get.upd-rkn.net
rls.upd-rkn.net
94.131.113.95
mtp.upd-rk.net
get.upd-rk.net
src.setup.mom
45.147.200.165
wired.setup.mom
188.127.225.231
get.rpm-bin.link

75.119.130.76
135.125.107.221
45.87.247.239
ci.upd-rkn.net
base.upd-rkn.net
mtp.upd-rkn.net
bot.upd-rkn.net
narwhal.rpm-bin.link
unicorn.rpm-bin.link
ci.rpm-bin.link
ops.rpm-bin.link
lib.rpm-bin.link
rhl.rpm-bin.link
source.rpm-bin.link
chifa.rpm-bin.link
amd64.rpm-bin.link
get.setup.mom
trust.setup.mom

File signatures

```

rule PTESC_apt_linux_UA_Excobalt__RiskTool__PsPatched{
  strings:
    $s = "For more details see ps(1)."
```

\$readproc_patched = {48 8B BB A8 02 00 00 48 ?? ?? 74 0DE8 ?? ?? ?? ?? 85 ?? 0F 85 81 05 00 00 48 8B BB 28 03 00 00 48 ?? ?? 74 0D E8 ?? ?? ?? ?? 85 C0 0F 85 68 05 00 00 41 8B 54 24 50 48 89 D8 85 D2 74 13 83 7B 04 02 0F 84 52 05 00 00 83 ?? ?? 0F 84 49 05 00 00}

```

  condition:
    ( uint32be ( 0 ) == 0x7f454c46 ) and ( all of them )
}

rule PTESC_apt_linux_UA_Excobalt__RiskTool__NetstatPatched{
  strings:
    $s = "usage: netstat [-vWeenNcCF] [ ] -r"
```

\$main_patched = {48 8D 05 ?? ?? ?? ?? 4A 8B ?? ?? 0F B6 [1-2] 8D 50 ?? 80 FA ?? 76 ?? 84 C0 74 ?? 66 0F EF C0 31 F6}

```

  condition:
    ( uint32be ( 0 ) == 0x7f454c46 ) and ( all of them )
}

rule PTESC_apt_linux_UA_Excobalt__RiskTool__SsPatched{
  strings:
    $s = "Usage: ss [ OPTIONS ]"
```

\$main_patched = {48 63 FD 48 8D 05 ?? ?? ?? ?? 4C 89 EE 83 C5 ?? 48 C1 E7 ?? BA ?? ?? ?? ?? 48 01 C7 E8 ?? ?? ?? ?? 83 FB ?? 0F 9E C0 83 FD ?? 0F 9E C2 48 83 C3 ?? 20 D0 0F 85 ?? ?? ?? ?? }

```

  condition:
    ( uint32be ( 0 ) == 0x7f454c46 ) and ( all of them )
}

rule PTESC_apt_linux_UA_Excobalt__RiskTool__HtopPatched{
  strings:
    $s = "/.htoprc"
```

\$readproc_patched = {48 89 DF E8 ?? ?? ?? ?? 48 8B 7B ?? 48 85 FF 74 ?? E8 ?? ?? ?? ?? 85 C0 75 ?? 48 8B 7B ?? 48 85 FF 74 ?? E8 ?? ?? ?? ?? 85 C0 75 ?? }

```

  condition:
    ( uint32be ( 0 ) == 0x7f454c46 ) and ( all of them )
}

rule PTESC_apt_multi_UA_Excobalt__Backdoor__GoRed__Obf{
  strings:
    $s1 = ").Parse" ascii nocase
    $s2 = ").MarshalBinary" ascii nocase
    $s3 = ").UnmarshalBinary" ascii nocase
    $s4 = ").Request" ascii nocase
    $s5 = ").Connect" ascii nocase
    $s6 = ").MarshalJSON" ascii nocase
    $s7 = ").Hash" ascii nocase
    $s8 = ").Add" ascii nocase
    $s9 = ").Close" ascii nocase
    $s10 = "JWT" ascii nocase
    $s11 = "config" ascii nocase
    $s12 = "GET" ascii nocase
    $s13 = "POST" ascii nocase
    $s14 = "a5674391" ascii nocase
    $c1 = {0F B6 54 0C ? ? 0F B6 74 0C ? ? 29 D6 40 88 74 0C ? ? 48 FF ? ? 48 83 F9 ? ? 7C ? ? }
    $c2 = {0F B6 54 04 ? ? 0F B6 74 04 ? ? 31 D6 40 88 74 04 ? ? 48 FF ? ? 66 0F 1F 44 00 ? ? 48 83 F8 ? ? 7C ? ? }
    $c3 = {0F B6 54 0C ? ? 0F B6 74 0C ? ? 01 F2 88 54 0C ? ? 48 FF ? ? 48 83 F9 ? ? 7C ? ? }
```

```

  condition:
    ((uint32be(0) == 0x7f454c46) or (uint16be(0) == 0x4d5a)) and (all of them)
}

rule PTESC_apt_multi_UA_Excobalt__Backdoor__GoRed{
  strings:
    $s1 = "embedded.GetConfig" ascii nocase
    $s2 = "common.runInBackground" ascii nocase
    $s3 = "common.run" ascii nocase
    $s4 = "common.RunCommand" ascii nocase
    $s5 = "revsh" ascii nocase
    $s6 = "dns" ascii nocase
    $s7 = "ws" ascii nocase
    $s8 = "dump" ascii nocase
    $s9 = "shell" ascii nocase
    $s10 = "files" ascii nocase
```

```

  condition :
    ((uint32be(0) == 0x7f454c46) or (uint16be(0) == 0x4d5a)) and (all of them)
}

```

ID	Name	Description
Reconnaissance		
T1595.002	Active Scanning: Vulnerability Scanning	ExCobalt used tools such as fscan to scan the system for vulnerabilities
Resource Development		
T1583.001	Acquire Infrastructure: Domains	ExCobalt registered domains and used these in its attacks
T1583.002	Acquire Infrastructure: DNS Server	ExCobalt used purchased DNS servers for tunneling
T1587.003	Develop Capabilities: Digital Certificates	ExCobalt created self-signed digital certificates for mutual TLS authentication for its GoRed backdoor
Initial Access		
T1199	Trusted Relationship	ExCobalt infiltrated an infrastructure and further developed the attack by taking advantage of a previously compromised contractor
T1195.001	Supply Chain Compromise: Compromise Software Dependencies and Development Tools	ExCobalt infected a component used to build the target company's legitimate software
Execution		
T1059.003	Command and Scripting Interpreter: Windows Command Shell	ExCobalt used the Spark RAT, which leveraged cmd.exe for command execution
T1059.004	Command and Scripting Interpreter: Unix Shell	ExCobalt used shell scripts, such as an sh script to install kitsune
T1059.006	Command and Scripting Interpreter: Python	ExCobalt used Python scripts for C2 communications
T1106	Native API	ExCobalt used the GoRed backdoor, which leveraged built-in Windows/Linux API calls to execute system commands
Persistence		
T1053.003	Scheduled Task/Job: Cron	ExCobalt used the GoRed backdoor, which leveraged cron to gain persistence in the system
T1505.003	Server Software Component: Web Shell	ExCobalt used various web shells to gain persistence
T1136.001	Create Account: Local Account	ExCobalt created accounts in compromised systems
Privilege Escalation		
T1068	Exploitation for Privilege Escalation	In its attacks, ExCobalt exploited the following vulnerabilities: CVE-2022-2586, CVE-2021-3156, CVE-2021-4034, CVE-2019-13272, CVE-2022-27228, CVE-2021-44228, CVE-2021-40438, CVE-2023-3519, BDU:2023-05857, and CVE-2019-12725
Defense Evasion		
T1140	Deobfuscate/Decode Files or Information	ExCobalt used various tools to deobfuscate and decode files, including the Spark RAT, which relied on a custom XOR algorithm to decrypt the payload
T1027.002	Obfuscated Files or Information: Software Packing	ExCobalt used the Spark RAT packaged with Enigma Protector to conceal its contents
T1027	Obfuscated Files or Information	ExCobalt used the GoRed backdoor whose earlier versions were obfuscated with garble
T1601.001	Modify System Image: Patch System Image	ExCobalt used modified ps , netstat , ss , and htop utilities that hid gsocket network connections by removing them from the list of returned connections
T1070.004	Indicator Removal: File Deletion	ExCobalt removed files from the file system using the rm command

ID	Name	Description
Credential Access		
T1003.008	OS Credential Dumping: /etc/passwd and /etc/shadow	ExCobalt used the GoRed backdoor to obtain passwords from /etc/shadow
T1003.001	OS Credential Dumping: LSASS Memory	ExCobalt used tools for dumping the system process lsass.exe
Discovery		
T1082	System Information Discovery	ExCobalt used various tools to obtain a system's host name, keyboard layout, and language information
T1614.001	System Location Discovery: System Language Discovery	ExCobalt used various tools to obtain system language information
T1033	System Owner/User Discovery	ExCobalt used various tools to obtain information about users
T1087.001	Account Discovery: Local Account	ExCobalt used the GoRed backdoor to receive information about the victim's account
T1083	File and Directory Discovery	ExCobalt used various tools, including the GoRed backdoor, to scan the file system
T1046	Network Service Discovery	ExCobalt used the GoRed backdoor to scan the gomap package, an nmap -like Go module
T1057	Process Discovery	ExCobalt used various tools, including the GoRed backdoor, to obtain information about active processes
Lateral Movement		
T1021.004	Remote Services: SSH	ExCobalt navigated the network via SSH, installing gsocket in the process
T1021.002	Remote Services: SMB/Windows Admin Shares	ExCobalt used the SMBExec utility
T1021.001	Remote Services: Remote Desktop Protocol	ExCobalt used RDP to connect
T1563.001	Remote Service Session Hijacking: SSH Hijacking	ExCobalt intercepted the contents of outgoing SSH sessions
Collection		
T1560.001	Archive Collected Data: Archive via Utility	ExCobalt used the knife utility to archive files using the gz < filename> command
T1560.002	Archive Collected Data: Archive via Library	ExCobalt used the GoRed backdoor to archive data for exfiltration via the gzip and tar Go packages
T1074	Data Staged	ExCobalt used the GoRed backdoor to store collected data in an archive before exfiltrating
Command And Control		
T1071.001	Application Layer Protocol: Web Protocols	ExCobalt used the Spark RAT, which relied on HTTP POST requests to communicate with its C2 server to get commands, and GoRed , which leveraged WS for C2 communications
T1132.001	Data Encoding: Standard Encoding	ExCobalt used various tools, including the Spark RAT, to encrypt messages sent to the C2 server in Base64
T1071.004	Application Layer Protocol: DNS	ExCobalt used the GoRed backdoor, which leveraged DNS tunneling for C2 communications
T1572	Protocol Tunneling	ExCobalt used the GoRed backdoor, which utilized DNS and ICMP protocol tunneling
T1132.002	Data Encoding: Non-Standard Encoding	ExCobalt used the GoRed backdoor, which leveraged Base32 to tunnel DNS traffic
T1573.001	Encrypted Channel: Symmetric Cryptography	ExCobalt used the GoRed backdoor, which leveraged AES-256-GCM to encrypt exfiltrated data
T1090.001	Proxy: Internal Proxy	ExCobalt used the GoRed backdoor, which could act as a proxy server

ID	Name	Description
T1095	Non-Application Layer Protocol	ExCobalt used the GoRed backdoor, which leveraged RPC, QUIC, and ICMP tunneling to communicate with its C2 server
Exfiltration		
T1041	Exfiltration Over C2 Channel	ExCobalt used various tools, including the Spark RAT, to transmit data over the C2 channel
T1048.001	Exfiltration Over Alternative Protocol: Exfiltration Over Symmetric Encrypted Non-C2 Protocol	ExCobalt used the GoRed backdoor, which leveraged HTTPS to exfiltrate data encrypted by AES-256-GCM
T1020	Automated Exfiltration	ExCobalt used the GoRed backdoor to automatically exfiltrate data to the threat actor's domain
T1567	Exfiltration Over Web Service	ExCobalt used the GoRed backdoor to exfiltrate data to the threat actor's server. The server acted as an exfiltration mechanism rather than a C2 server
Impact		
T1485	Data Destruction	ExCobalt used the locker wiper ransomware to encrypt the system
T1486	Data Encrypted for Impact	

Verdicts by Positive Technologies products

PT Sandbox

apt_multi_UA_Excobalt__Backdoor__GoRed
 apt_multi_UA_Excobalt__Backdoor__GoRed__Obf
 apt_linux_UA_Excobalt__RiskTool__PsPatched
 apt_linux_UA_Excobalt__RiskTool__NetstatPatched
 apt_linux_UA_Excobalt__RiskTool__HtopPatched
 apt_linux_UA_Ruh8__Dropper__kit
 tool_multi_ZZ_WebShell__Backdoor__Generic__PHP
 tool_multi_ZZ_WebShell__Backdoor__PHP__Obfuscated
 tool_multi_ZZ_Gsocket__NetTool

MaxPatrol SIEM

Unix_Boot_Modify (mod probe)
 Unix_System_Information_Discovery
 Unix_Connect_from_Suspicious_Dir
 Unix_Suspicious_Command

PT NAD

BACKDOOR [PTsecurity] GoRed Request sid: 10011377
 BACKDOOR [PTsecurity] GoRed Exfiltration sid: 10011378
 ATTACK AD [PTsecurity] Anonymous SMB connect to IPC share sid: 10005876, 10005877
 TOOLS [PTsecurity] .gs.thc.org domain resolve. Probably activity gsocket sid: 10009306
 TOOLS [PTsecurity] gsocket server activity sid: 10009305
 SUSPICIOUS [PTsecurity] Suspicious communication using UA go-external-ip & External IP Check sid: 10011380

POLICY [PTsecurity] HTTP Host header RFC2616 violation sid: 10011339

POLICY [PTsecurity] Unsuccessful SMB Port Scan External Network sid: 10001310, 10003655