Malware Spotlight: Linodas aka DinodasRAT for Linux - Check Point Research

By etal

Published: 2024-03-31 · Archived: 2026-04-05 14:09:55 UTC

Introduction

In recent months, Check Point Research (CPR) has been closely monitoring the activity of a Chinese-nexus cyber espionage threat actor who is focusing on Southeast Asia, Africa, and South America. This activity significantly aligns with the insights the Trend Micro researchers publicly shared in their comprehensive analysis of a threat actor called Earth Krahang. This actor's toolset notably includes a cross-platform backdoor named DinodasRAT, also known as XDealer, which was also observed previously in attacks by the Chinese threat actor LuoYu.

The Windows version of this malware was thoroughly analyzed by ESET, but its Linux counterpart has not garnered much public interest. In this blog post, we share our full technical analysis of the latest Linux version (v11) of DinodasRAT, which we track as Linodas. It appears to be more mature than the Windows version, with a set of capabilities tailored specifically for Linux servers. In addition, the latest version introduces a separate evasion module to hide any traces of malware in the system by proxying and modifying the system binaries' execution.

While we finalized this blog spot, a technical analysis of an older V10 version of the Linux RAT was published by researchers from Kaspersky. Although it overlaps with our findings to some extent, our report provides additional information on the advancements in the latest version (v11) of the Linux RAT.

Dinodas Origins

Several hints indicate that DinodasRAT was initially based on the open-source project called SimpleRemoter, a remote access tool based in turn on the Gh0st RAT, but with several additional upgrades. Similarities between SimpleRemoter and an older version of Dinodas RAT include the usage of the same zlib library version 1.2.11, and overlaps in the code, such as the OS version detection function which is nearly identical:

Figure 1 – Similarities in the OS version detection function between the Dinodas sample (left) and the open-source code.

While we can't say definitively that DinodasRAT developers reused the entire source code, it is clear that they were inspired by it at least regarding the C2 command functionality which shows significant similarities between the two RATs.

We also observed additional open-source code usage in the DinodasRAT code from another repository created by the same developer. In this case, the DinodasRAT authors used functionality related to handling the INI files.

The final example of reusing open-source code is the developers' choice for encryption. Instead of implementing their own method, they decided to go with the encryption used in QQ.

Linodas as a Separate Code Base

Each sample of the cross-platform DinodasRAT embeds a string containing the backdoor's internal version. Inside the Linux samples (that we track as **Linodas)**, we observed the following strings that reflect the backdoor evolution:

Mark	First seen	Hashes
Linux_%s_%s_%u_V7	July 2021	3d93b8954ed1441516302681674f4989bd0f20232ac2b211f4b601af0fcfc13b bf830191215e0c8db207ea320d8e795990cf6b3e6698932e6e0c9c0588fc9eff
Linux_%s_%s_%u_V10	Jan 2023	15412d1a6b7f79fad45bcd32cf82f9d651d9ccca082f98a0cca3ad5335284e45
Linux_%s_%s_%u_V11	Nov 2023	6302acdfce30cec5e9167ff7905800a6220c7dda495c0aae1f4594c7263a29b2 ebdf3d3e0867b29e66d8b7570be4e6619c64fae7e1fbd052be387f736c980c8e (embedded module)

The earliest Linux version we retrieved was first seen in the wild in July 2021. However, this version is numbered internally as v7, which indicates that the development of the malware started earlier. In comparison, at the same time the Windows branches of the backdoor included `Rin_%s_%s_%u_V6` and `Win_%s_%s_%u_V6` versions.

While Linodas shares logic with the Windows variant, it also adds its own set of behaviors designed specifically for Linux servers. The authors appear to be proficient in Linux, as their choice to support the OS was not just a simple #ifdef variant of a Windows RAT but a different project with a separate code base and possibly a different development team. Looking at the latest Linodas version (v11), we can also observe a Windows sample communicating with the same C2 server `update.microsoft-setting[.]com`:

Version	Operating System	Hash
Linux_%s_%s_%u_V11	Linux	6302acdfce30cec5e9167ff7905800a6220c7dda495c0aae1f4594c7263a29b2
Win_%s_%s_%u_V10	Windows	57f64f170dfeaa1150493ed3f63ea6f1df3ca71ad1722e12ac0f77744fb1a829

Two samples that have different internal versions may indicate that there are two different development teams, or at least two backdoors in different development stages communicating with the same C2 server. The Linux and Windows versions have overlapping command IDs, seamlessly supporting the same malware functionality for different operating systems.

The implant is installed on Linux servers as a way for the threat actors to gain an additional foothold in the network. Most of the samples we found have the name ntfsys, apparently attempting to masquerade as a system or driver file related to NTFS.

In our technical analysis, we split the logic of the RAT into several parts for easier comprehension.

Initial Startup

Linux is different from Windows; persistence is different than it is in Windows, execution permissions need to be ensured, and so on. The Linux backdoor handles those functions quite well. Once the backdoor is executed, it verifies if it's the first run by checking if it received two arguments: the letter `d`, and the calling daemon process ID. If those arguments are absent, the backdoor calls the daemon function and establishes persistence on the system. It then re-runs itself again properly: it gets the process ID and the self-execution path, and executes the command `[SELF_PATH] d [SELF_PID]` through the `system` function.

Persistence methods

The persistence process is quite extensive and covers multiple Ubuntu versions and RedHat distributions. It first checks for the current OS version by reading the files `/proc/version` and `etc/lsb-release` and parsing the output. Then, based on the gathered data, it achieves persistence by one of the following methods:

Method 1 (Ubuntu) – rc.local enabled via systemd
The malware first checks if the file `/lib/systemd/system/rc.local.service` doesn't exist and then proceeds to write the following string into it:

Description=/etc/rc.local Compatibility

ConditionFileIsExecutable=/etc/rc.local

ExecStart=/etc/rc.local start

[Unit] Description=/etc/rc.local Compatibility ConditionFileIsExecutable=/etc/rc.local After=network.target [Service] Type=forking ExecStart=/etc/rc.local start TimeoutSec=0 RemainAfterExit=yes

```
[Unit]
Description=/etc/rc.local Compatibility
ConditionFileIsExecutable=/etc/rc.local
After=network.target

[Service]
Type=forking
ExecStart=/etc/rc.local start
TimeoutSec=0
RemainAfterExit=yes
```

It then creates the following symlink `/lib/systemd/system/rc.local.service` → `/etc/systemd/system/`. Next, it checks if the file `/etc/rc.local` exists, and if it does, it adds the following string to it:

#!/bin/bash [SELF_FILE_PATH] exit 0

```
#!/bin/bash
[SELF_FILE_PATH]
exit 0
```

It then makes the `/etc/rc.local` file executable by calling the command `chmod 777` on it, and then validates that the persistence was written correctly into it, ensuring the self path is the actual self file path. Then, it changes the following INI fields in the `/lib/systemd/system/rc.local.service` file:

WantedBy=multi-user.target

[Service] RemainAfterExit=no [Install] WantedBy=multi-user.target Alias=rc-local.service

```
[Service]
RemainAfterExit=no

[Install]
WantedBy=multi-user.target
Alias=rc-local.service
```

Method 2 (Red Hat) – init.d script
The backdoor first runs the command `where chkconfig` parses the output and tries to execute `chkconfig`. If the command is found and runs correctly, it adds it to the `PATH` environment variable and proceeds to do the actual persistence. It checks if the file `/etc/init.d/[SELF_FILE_NAME]` doesn't exist, then proceeds to write the following string into it:

Provides: [SELF_FILE_NAME]

Required-Start: $local_fs $network

Required-Stop: $local_fs

Short-Description: [SELF_FILE_NAME] service

Description: [SELF_FILE_NAME] service daemon

#!/bin/sh ### BEGIN INIT INFO # Provides: [SELF_FILE_NAME] # Required-Start: $local_fs $network # Required-Stop: $local_fs # Default-Start: 2 3 4 5 # Default-Stop: 0 1 6 # Short-Description: [SELF_FILE_NAME] service # Description: [SELF_FILE_NAME] service daemon ### END INIT INFO [SELF_FULL_ATH] restart

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          [SELF_FILE_NAME]
# Required-Start:    $local_fs $network
# Required-Stop:     $local_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: [SELF_FILE_NAME] service
# Description:       [SELF_FILE_NAME] service daemon
### END INIT INFO
[SELF_FULL_ATH] restart
```

If the file wasn't created, it writes the same data to the file `/etc/ch.sh` and executes the command `mv /etc/ch.sh /etc/init.d/[SELF_FILE_NAME]` then runs the command `chmod 777` on the created file and executes it. Then, it begins validating the persistence by running the command `chkconfig --list | grep [SELF_FILE_NAME]`. If the result doesn't contain `6:`, it runs the following two commands:

chkconfig --add [SELF_FLE_NAME]

chkconfig zentao [SELF_FLE_NAME]

chkconfig --add [SELF_FLE_NAME] chkconfig zentao [SELF_FLE_NAME]

```
chkconfig --add [SELF_FLE_NAME]
chkconfig zentao [SELF_FLE_NAME]
```

Method 3 (Red Hat) – rc.local
Persistence is done through the `/etc/rc.d/rc.local` file. file. If the file exists, the backdoor checks if the self-path is inside the contents, and if not, adds itself to the file with the string `\n[SELF_PATH]\n`.

Core Logic

After the backdoor is run properly with 2 arguments, it proceeds to the main logic. First, it performs a set of checks such as the root privilege, its path/folder, and the calling daemon process ID, and saves them all in global variables. It then changes its file timestamp using the following command: `touch -d \"2010-09-08 12:23:02\" [SELF_FILE_PATH]`. This timestamp is also set to other backdoor-related files such as the config files when accessing them. This enables the files to "blend in" with the other files in the filesystem.

Next, it reads a config file based on the hardcoded string `/etc/.netsc.conf`. The config files for all the samples are usually hidden files. From the config, it attempts to read a field called `imei` under the `para` section. This field represents the bot ID generated for the infected machine. If this field doesn't exist in the config, a unique machine ID is generated based on machine parameters in the following way:

1. Get the machine's MAC address by running an `ifconfig` or `ip` command and extract the MAC address from the output. This is OS distribution-dependent.
2. Run the command `dmidecode`, which dumps the system's SMIBIOS.
3. Combine the MAC address and the output from `dmidecode` and perform md5sum.
4. Generate a random number.
5. Generate a timestamp based on the current time.

All of those fields are combined in the following string: `Linux_[TIMESTAMP]_[MD5SUM]_[RANDOM NUMBER]_V11`. For example: `Linux_20240310_11cb06d0bf454c3708a3658c2601ea16_40459_V11`.

After generation, the bot ID is saved to the config file, and the config file timestamp is also changed in a way similar to how it's done for the executable.

Next, the backdoor does basic system enumeration to get the distribution, exact OS version, and system architecture. All those values are saved in global variables and are used later when the backdoor contacts the C2 for the first time. The hardcoded C2 address (in the latest version, `update.microsoft-setting[.]com:443`) is parsed and saved in a global struct, and the malware reads two more config fields, `mode`, and `checkroot`, from the `para` section. The `mode` field serves as the type of C2 communication to use, TCP or UDP. The `checkroot` indicates if the backdoor should be monitoring logged-in users. After the initial configuration, the backdoor proceeds to create several threads which are used for monitoring and cleanup, and initiates the connection to the C2 server.

Monitoring/Cleanup Threads

The backdoor creates five threads tasked with system monitoring, helper module download, and cleanup of old reverse shell connections..

Thread #1: Logged-in user monitor. If the `mode` field in the config or the global variable for `mode` is set to 1, meaning the connection type is TCP, this thread monitors logged-in users using the who command. It parses its output and closes the C2 connection if the logged-in IP is not a local IP or the C2 IP.

Thread #2: C2 connection status monitoring. Each time a valid request is made to the C2, the time field in the global C2 connection struct is updated. If half an hour passed since the last request to the C2, the thread closes the connection to the C2.

T**hread #3: Filter module download and setup.** The thread first verifies that the module wasn't already downloaded (through the use of a global variable). If it wasn't, the thread performs the following steps:

1. Check if the file `[SELF_PATH].so6` exists and calculate an md5 hash of its content.
2. Send an encrypted request to the C2, requesting the md5 hash of a module available on the C2 server, and compare the received md5 hash with the existing one.
3. If the hash is different, make another request to the C2 and save a newly received file with the same name.
4. Read data from the file `/usr/lib/libsysattr.so`, likely dropped at an earlier stage of infection. This file should contain a set of values separated by the character `|` and represents a set of instructions for executables to be replaced.
5. Locate in the system each file specified in the instructions file and backup it with a name in the following format: `[FILE_NAME].a`. Then replace the original file with the newly received so6 filter module and make it executable.

All these steps allow the threat actors to wrap certain system executables to control their output, which is detailed later in the dedicated "Filter module" section.

Thread #4: Logged-in users monitor and logging. If a user is logged in to the Linux machine and its IPv4 is not a local IPV4 or one of the C2s, its details are logged and sent to the C2 server.

Thread #5: Reverse shell old sessions cleanup. This thread monitors reverse shell sessions. If there is a reverse shell session that was not active for the last 3599 seconds (almost one hour), the session is removed.

C2 Communication

Before the C2 communication starts, two global structs are checked: one contains the config value indicating whether to use TCP or UDP when connecting to the C2, and the other one indicates whether the C2 communication should cease if there are logged-in users. If any of those checks fail, communication with the C2 is not initiated. If the checks pass, the backdoor parses the C2 address (host and port) and resolves a C2 domain to IPv4 if needed.

Next, the malware sets up a socket in TCP or UDP mode, based on the configuration, and attempts to connect to the C2. If the connection is successful, a thread to parse C2 commands is spawned. The C2 commands supported in the latest version of the backdoor are detailed in the next section.

After this initial connection to the C2 server is set and the C2 command thread is created, an endless loop is executed which is responsible for sending a heartbeat to the C2 server. The heartbeat contains the following values combined and separated by `\t`:

- Distribution info
- System architecture

- The string “root”
- Constant value 0xC
- UDP packet length 800
- Self path

If the request to the C2 failed, the connection to the C2 is reestablished, and the heartbeat string is generated and sent again.

Supported C2 Commands

Similar to the Windows backdoor, the Linux backdoor supports a wide range of capabilities. In the following table, we outline all of them and indicate whether they exist in the Windows version of the backdoor.

Id	Description	Arguments	Exists in Windows Version
0x02	List files and directories under a specified folder	Folder name	✔️
0x03	Delete files or directories	List of files or directories separated by &&	✔️
0x05	Send files to the C2	Request ID, file list separated by &&	✔️
0x06	Stop the “send files” command	–	✔️
0x08	Download a file from the C2 and execute it (if a flag is enabled)	Execute flag, Filename, File data, all separated by \t	✔️
0x09	Stop the “download file” command	–	✔️
0x0E	Update C2 URL	List of C2 URLs separated by \t	✔️
0x0F	Enumerate logged-in users	–	
0x11	Enumerate running processes	–	✔️
0x12	Kill process	Process ID to kill	✔️
0x13	Enumerate running services	–	✔️
0x14	Start/Stop service	Service name, action type, separated by \t. Action type can be one of the following: 1 – start service 0 – stop service 2 – stop and delete service	✔️

Id	Description	Arguments	Exists in Windows Version
0x18	Execute a process and send the response back	Process path to execute	✔️
0x19	Make a file executable and execute it	Receive maybe multiple files to execute separated by \t	✔️
0x1A	Start/Stop/Get State for HTTP proxy	Integer value	
0x1B	Reverse shell start	–	✔️
0x1C	Reverse shell restart	–	
0x1D	Reverse shell close	–	
0x1E	Write to reverse shell	Binary values split by \0x01\0x02	
0x27	Rename/copy/move file	Action type, file path, new path	✔️
0x28	Send “ok” to the C2	–	✔️
0x2B	Change proxy connection type	Integer value	✔️
0x2C	Set proxy type	Integer value	✔️
0x2D	Change file transfer mode	Integer, Integer value	✔️
0x2E	Self-fully uninstall, remove persistence, kill the parent daemon, and exit	–	✔️
0x31	Update a global integer value used as the UDP packet length	Integer value	✔️
0x32	Read a file and return its contents	file path, max bytes to read	
0x33	Write data to a file	file path, bytes to write in hex string	
0x34	Collect user activity through various files such as: /var/run/utmp /var/log/wtmp /var/log/lastlog	–	
0x35	Parse and send the /usr/lib/libsysattr.a file	–	
0x36	Parse data and write to /usr/lib/libsysattr.a file	Fields separated by \t	

The Filter Module

As mentioned previously, in Linodas v11, Thread #3 is responsible for downloading an additional module which replaces any specified binaries in the system. None of the previous versions support this functionality.

The filter module we received from the actor-controlled C&C server by running `ntfsys` (v11) is saved as `ntfsys.so6` (sha256: `ebdf3d3e0867b29e66d8b7570be4e6619c64fae7e1fbd052be387f736c980c8e`).

Figure 2 – File detections for the Filter module when first seen in the wild in November 2023.

As described earlier, the installation thread substituted certain binaries in the system with the filter module. The module purpose is to proxy the execution of these binaries and control their output. This is how it's done step by step:

Figure 3 – A diagram of the execution of a system binary "wrapped" by the filter module modifying its output in real time.

1. The module starts every time the system tries to use the replaced binary, with or without arguments.
2. When executed, the module checks if the config file in `/usr/lib/libsysattr.a` exists. This separate configuration file is not downloaded together with the module and is likely placed by the threat actors into the server using the Linodas reverse shell. From `para` section of the configuration file, the module loads two fields, `ip` and `name`.
3. The module combines all of the arguments it received, separates them with spaces, and checks if the original binary with the name `[SELF_PATH].a` If it doesn't, it outputs the `bash shell` and exits. If the file exists, it executes it with the arguments received and appends string `2>&1` which merges the error output with the standard output, so both can be manipulated or viewed together.

4. While the module executes the subprogram, it reads chunks of the output and splits them by line. Each line goes through a filtering process, where the line containing any of the values for `ip` or `name` from the config is ignored. If the line passes the filtering, it is printed.

The threat actors are likely using this module as a poor man's "rootkit", allowing them to filter any values, such as IP, username, process name, or other artifacts, from various information-gathering binaries such as `who`, `netstat`, `ps`, etc. This enables them to hide the presence of Linodas and its artifacts from any monitoring efforts by vigilant victims.

Conclusion

In this blog post, we analyzed the Linux version of DinadasRAT, used by Chinese-nexus APT threat actors and observed since at least 2021. While there are multiple similarities with the Windows version, the Linux malware indicates a separate and independent development branch that introduces auxiliary modules with separate configuration files, and additional C2 commands focused on establishing and controlling reverse shells, collecting user activity from the logs, and manipulating local file content.

The complexity and capabilities of Linodas highlight the continued emphasis by threat actors on targeting Linux servers both as a method for maintaining presence and as a pivot point within compromised networks. This approach likely exploits the typically lower level of security protocols and solutions usually installed on Linux boxes, allowing the attackers to extend their foothold and remain undetected for longer periods.

Protections

Check Point Customers remain protected against attacks detailed in this report while using Check Point Harmony Endpoint and Threat Emulation:

- Backdoor.Win.OperationJacana.A
- Trojan.Wins.Jacana.ta.*
- Backdoor_Linux_DinodasRAT_*

IOCs

Type	Value
SHA256	3d93b8954ed1441516302681674f4989bd0f20232ac2b211f4b601af0fcfc13b
SHA256	bf830191215e0c8db207ea320d8e795990cf6b3e6698932e6e0c9c0588fc9eff
SHA256	15412d1a6b7f79fad45bcd32cf82f9d651d9ccca082f98a0cca3ad5335284e45
SHA256	98b5b4f96d4e1a9a6e170a4b2740ce1a1dfc411ada238e42a5954e66559a5541
SHA256	a2c3073fa5587f8a70d7def7fd8355e1f6d20eb906c3cd4df8c744826cb81d91
SHA256	ebdf3d3e0867b29e66d8b7570be4e6619c64fae7e1fbd052be387f736c980c8e
SHA256	6302acdfce30cec5e9167ff7905800a6220c7dda495c0aae1f4594c7263a29b2