

An In-Depth Analysis of a Two-Stage Botnet

✕ blog.xlab.qianxin.com/rustduck-en

Wang Hao

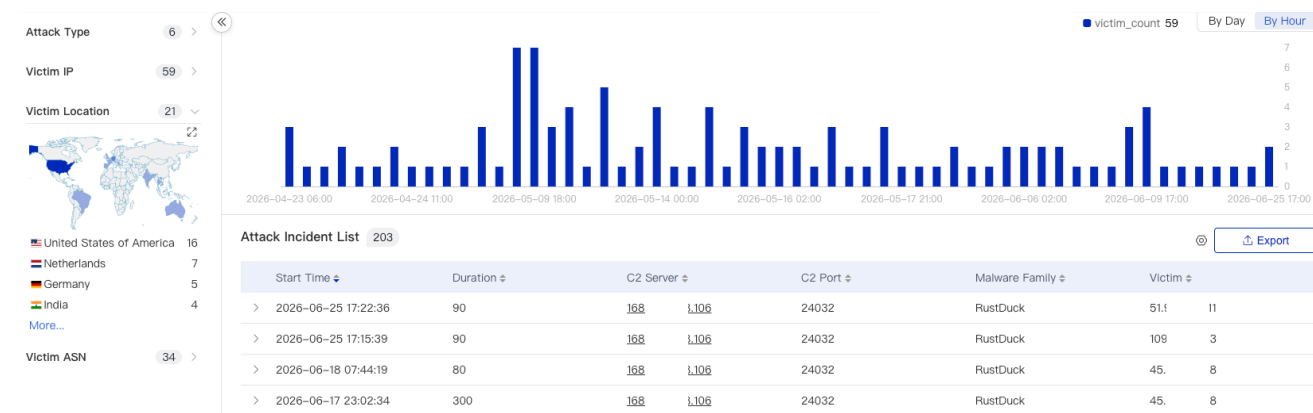
June 30, 2026



Overview

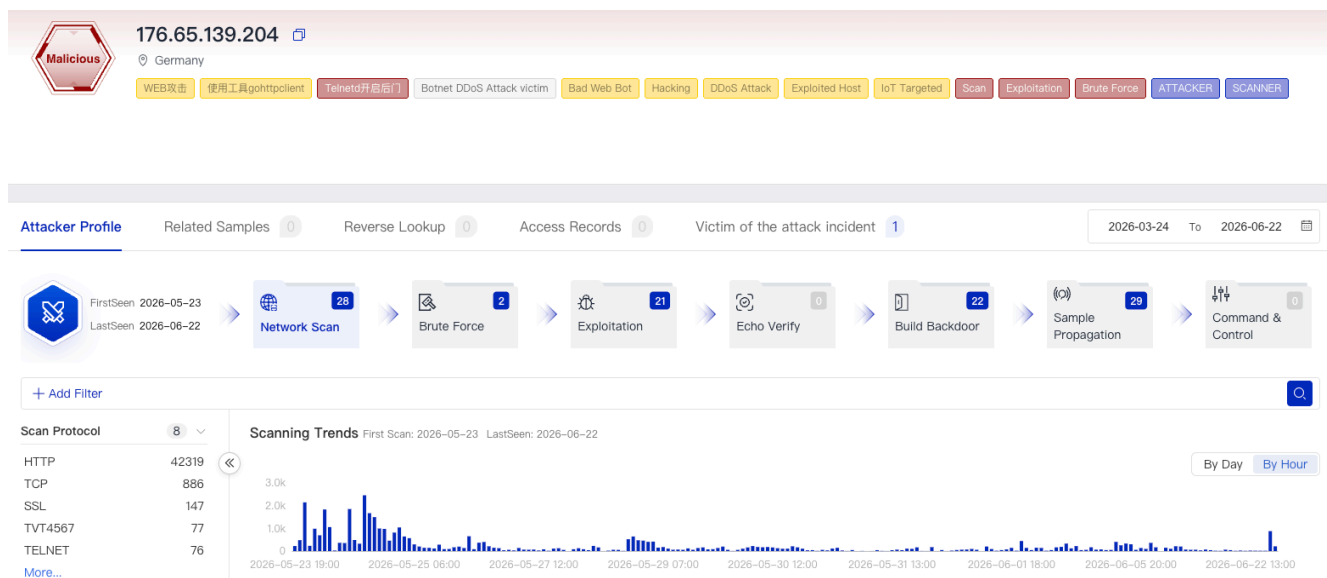
Since February 2026, the XLAB large-scale network threat perception system has detected a new malware family active in cyberspace that adopts a **Loader + Core (two-stage loading)** architecture. Currently, the family has spawned multiple variants, with the main core functionality being the execution of large-scale Distributed Denial-of-Service (DDoS) attacks. It also possesses strong cross-platform adaptability and continuous evolution capabilities.

Although the family's current activity level and influence in DDoS attacks are not yet comparable to some mainstream botnets, its speed of technological evolution deserves significant attention. Research has found that the family is undergoing a comprehensive technological transition from **C to Rust**, and its anti-defense and traffic encryption techniques are also iterating rapidly. Based on the shift of its technology stack from C to Rust and the characteristic of early core payloads encrypting three duckdns C2 domains, we have named it RustDuck.



Attack Incident Trend

In terms of sample propagation, the spread chain of this family mainly covers IoT devices, web applications, and enterprise infrastructure. The attack methods primarily involve weak password brute-forcing (Telnet/SSH) and the exploitation of various RCE vulnerabilities, including device vulnerabilities in Android ADB, TVT API, Ruijie, TP-Link, ZTE, and others, as well as web/component vulnerabilities such as ThinkPHP, Jenkins, YARN. It also combines several historical CVEs (CVE-2025-29635, CVE-2017-17215, CVE-2018-8007, CVE-2024-1781) to expand the attack surface. Overall, it presents a combined propagation characteristic of "weak passwords + IoT vulnerabilities + Web RCE", capable of covering routers, cameras, Android terminals, and server environments, enabling large-scale automated intrusion and payload delivery. Currently, over 20 IPs have been observed participating in spreading the RustDuck botnet, with the most active implant source IP being: 176.65.139[.]204

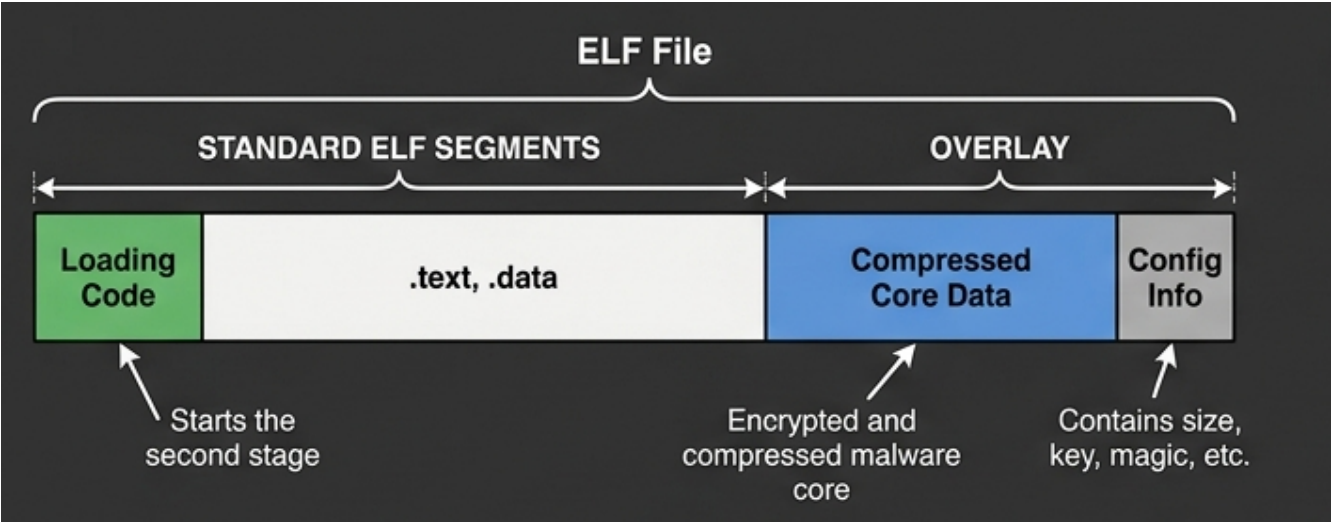


Sample Analysis

1. Evolution and Clustering Analysis of the Loader

Loader-stage samples typically adopt a streamlined three-part design: **Loading Code**, **Compressed Data (Compressed Core)**, and **Configuration Information (Config)**.

Its classic file layout is as follows: the loading code resides in the code segment at the beginning of a standard ELF file, while the core compressed data and configuration information are appended as overlay data at the end of the file.



By performing structured reverse engineering on the configuration information (Config) at the file tail, we can clearly cluster the family's Loader into the following four evolutionary stages:

Stage	SHA1 (first 8 chars)	Config Size	Decryption Algorithm	Decompression Algorithm	Magic Verification Feature
Loader Variant 1	8315f650	16 bytes	LCG + XOR	LZ4	Dynamic verification (ROL4 + XOR)
Loader Variant 2	6aa791c7	33 bytes	Xoshiro128 + XOR	BLZ	Introduces dynamic constants

Stage	SHA1 (first 8 chars)	Config Size	Decryption Algorithm	Decompression Algorithm	Magic Verification Feature
Loader Variant 3	4d11bd49	48 bytes	Standard XOR	LZ4	Fixed plaintext "ASHPCK\x01\x00"
Loader Variant 4	d39a3ee9	32 bytes	ChaCha20	LZ4	Fixed plaintext "iEMPCK\x02\x00\x00"

1. Loader Variant 1

SHA1: 8315f650e9e4f67c00277b076ab304eed23db47d

- **Config Size:** 16 bytes (equally divided into 4 fields, each 4 bytes)
- **Memory Layout:**

```

+-----+-----+-----+-----+
|  Key (4B)  | Compress_Size(4B) | Decompress_Size(4B) | Magic (4B) |
+-----+-----+-----+-----+
```

- **Magic Verification Algorithm:** Introduces cyclic left shift (ROL) and XOR combined verification:

```
compress_size ^ decompress_size ^ ROL4(key, 13) ^ 0x5A3C9E7F == magic
```

Encryption/Decryption and Decompression: Uses a Linear Congruential Generator (LCG) to generate a pseudo-random number sequence for XOR decryption, followed by **LZ4** decompression.

2. Loader Variant 2

SHA1: 6aa791c76b3107fca9d57b7ecea8f46d97d83738

- **Config Size:** 33 bytes

- **Memory Layout:**

```
+-----+-----+-----+-----+
-----+
|  Key (16B)  | Compress_Size(4B) | Decompress_Size(4B) |  Magic (8B)  |
Noise_Size(4B)|
+-----+-----+-----+-----+
-----+
```

- **Encryption/Decryption and Decompression:** Upgraded to **Xoshiro128 + XOR** decryption, with the decompression algorithm changed to **BLZ**.
- **Countermeasure Characteristics:** This variant introduces multiple hard-coded constants to obfuscate the decryption and Magic verification process. Since the constants vary dynamically across different samples, it is extremely difficult for security researchers to perform static batch decryption.

3. Loader Variant 3

| **SHA1:** 4d11bd496da82d15b3ed13050f414e44f5a892d4

- **Config Size:** 48 bytes
- **Memory Layout:**

```
+-----+-----+-----+-----+
-----+
| Compress_Size(4B) | Decompress_Size(4B) |          Key (32B)          |
Magic (8B) |
+-----+-----+-----+-----+
-----+
```

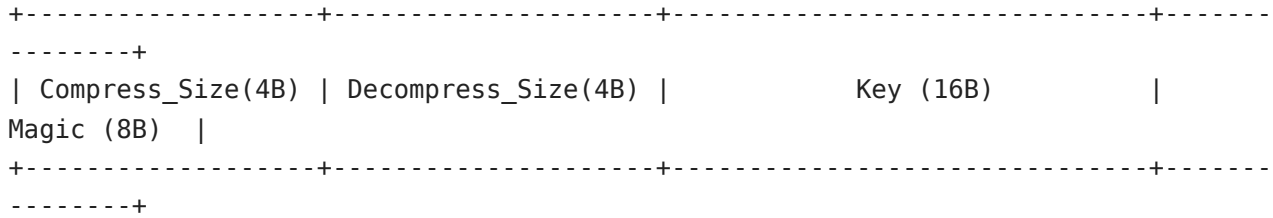
- **Magic Verification:** Fixed to the plaintext string "ASHPCK\x01\x00".
- **Encryption/Decryption and Decompression:** Reverts to standard **XOR** decryption and **LZ4** decompression, moving towards standardization.

4. Loader Variant 4

| **SHA1:** d39a3ee96be6b8f5238cb1253514ab55c88f714c

- **Config Size:** 32 bytes

- **Memory Layout:**



- **Magic Verification:** Fixed to the plaintext string "iEMPk\x02\x00\x00".
- **Encryption/Decryption and Decompression:** Introduces the high-strength **ChaCha20** stream cipher, with decompression still using **LZ4**.

2. Evolution of the Core Stage

With the shift to the Rust programming language, the family's Core (core module) exhibits extremely high engineering complexity in key derivation, anti-analysis, and communication protocols.

1. Key Generation and Encryption Algorithms

In terms of key derivation, the new variants uniformly introduce the **HKDF-SHA256** algorithm, and two coexisting key generation sources have been observed:

1. **UTC Time-Based Dynamic Key:** Synchronized with the clock, the key is forcibly updated every 10 minutes to counter replay attacks and long-term traffic auditing.
2. **Asymmetric Key Exchange:** Employs the **Curve25519 (Noise_IK_25519)** architecture to achieve forward secrecy.

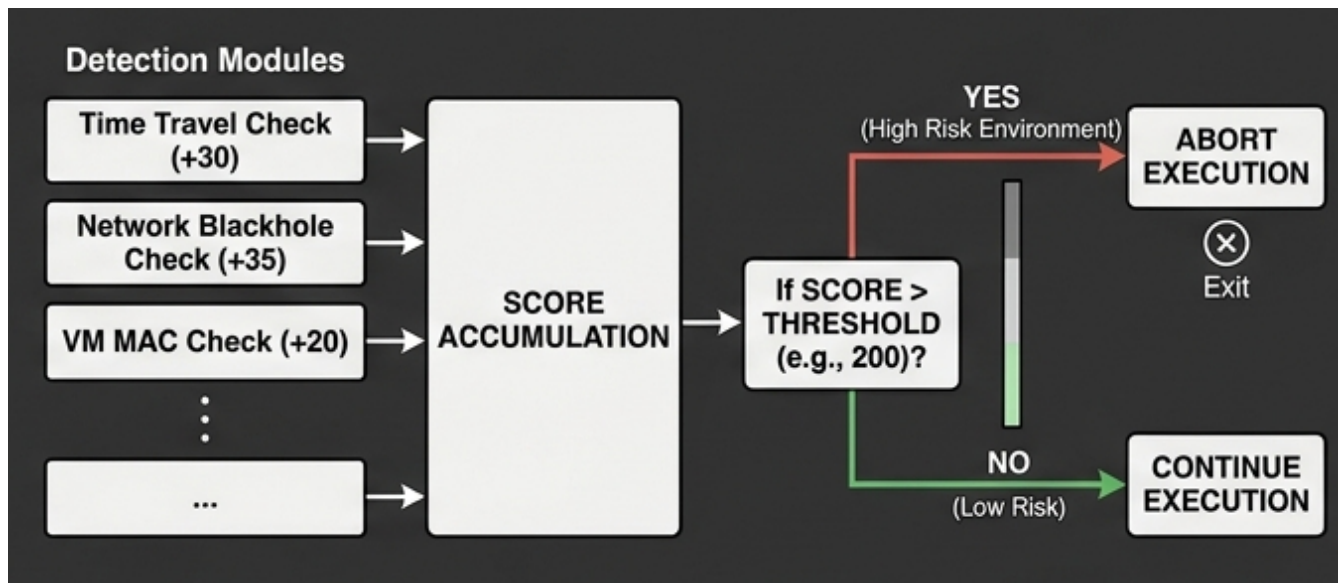
In terms of symmetric encryption/decryption algorithms, the evolution has split into two branches:

- **Branch A:** Uses a permuted (Permutation transformed) **Ascon128** lightweight encryption algorithm throughout.
- **Branch B:** Adopts a hybrid encryption scheme. Uses **ChaCha20-Poly1305** during the handshake phase, and switches to **AES-GCM** once the handshake is compliant.

2. Anti-Debugging and Environment Checks

To thwart automated sandbox analysis and dynamic debugging by reverse engineers, the Core module incorporates a **dynamic weight scoring mechanism**. During runtime, the

software iterates through various environment checks. When the **accumulated risk score exceeds a preset threshold**, the program automatically erases traces and exits.



The specific detection items and weight assignments are as follows:

Detection Technique	Risk Weight	Detection Principle and Countermeasure Logic
Analysis Tools Check	100	Scans the process list, finding packet capture tools (wireshark, tcpdump, etc.) and reverse engineering tools (gdb, ida, frida, x64dbg, etc.).
Debug Check	100	Reads /proc/self/status and checks if TracerPid is not 0.
Lib Check	100	Parses /proc/self/maps and checks for the presence of injection or debugging libraries such as frida, asan, ubsan, libdl-inject.
SHA256 Checksum	100	Verifies the SHA256 signature appended at the end of the file. If the file has been modified by a reverse engineer or breakpoints have been set (modifying opcode), it refuses to run.
Honeypot Check	50	Checks for honeypot feature files in key system paths, such as Cowrie (/etc/cowrie/cowrie.cfg) or Dionaea.
Env Check	40	Checks environment variables (environ) for sensitive keywords such as sandbox, malware, virus, sample.

Detection Technique	Risk Weight	Detection Principle and Countermeasure Logic
Network Blackhole	35	Asynchronously connects to the reserved test IP 192.0.2.1. If the connection succeeds within 0.5s, it indicates being in a deceptive sandbox environment that fakes Internet connectivity.
Time Travel Check	30	Dual clock source cross time difference verification. Compares system time and monotonic time before and after usleep to prevent sandbox time acceleration or reverse-engineering breakpoint stalling .
Hardware Check	25	Reads DMI information and SCSI device interface, searching for virtualization hardware keywords such as virtualbox, vbox, bochs.
VM MAC Check	20	Iterates through network interface MAC addresses, checking if they belong to the specific OUI prefixes of VBox (08:00:27), VMware, or Parallels.
PID Density Check	10	Checks if the total number of processes in the current system is less than 5, filtering out extremely clean minimalist emulator environments.

Key Anti-Debugging Code Implementation Reference:

```
// 1. Time Travel Check: Dual clock source cross verification
int check_timing_safetly() {
    struct timeval tv1, tv2;
    struct timespec ts1, ts2;

    gettimeofday(&tv1, NULL);
    clock_gettime(CLOCK_MONOTONIC, &ts1);

    usleep(500000); // deliberately suspend for 0.5 seconds

    gettimeofday(&tv2, NULL);
    clock_gettime(CLOCK_MONOTONIC, &ts2);

    long v2 = (tv2.tv_sec - tv1.tv_sec) * 1000 + (tv2.tv_usec - tv1.tv_usec) / 1000;
    long v3 = (ts2.tv_sec - ts1.tv_sec) * 1000 + (ts2.tv_nsec - ts1.tv_nsec) /
1000000;
```

```

    // Must actually sleep for more than 300ms, and the system error between the two
    clock sources must be within 1000ms
    if (v2 > 299 && v3 > 299 && abs(v3 - v2) <= 1000) {
        return 1; // Environment is normal
    }
    return 0; // Time anomaly: Possibly fast-forwarded by a sandbox, or halted by a
    reverse-engineering breakpoint
}

// 2. Network Blackhole Check: Network blackhole/sandbox blind connection detection
int check_network_blackhole() {
    int fd = socket(AF_INET, SOCK_STREAM, 0);
    if (fd < 0) return 0;

    fcntl(fd, F_SETFL, fcntl(fd, F_GETFL, 0) | O_NONBLOCK); // set non-blocking

    struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(80) };
    addr.sin_addr.s_addr = inet_addr("192.0.2.1"); // RFC 5737 test reserved IP
    (normally absolutely unreachable)

    connect(fd, (struct sockaddr *)&addr, sizeof(addr));

    fd_set writeset;
    FD_ZERO(&writeset);
    FD_SET(fd, &writeset);
    struct timeval tv = { .tv_sec = 0, .tv_usec = 500000 }; // asynchronous wait for
    0.5 seconds

    int res = select(fd + 1, NULL, &writeset, NULL, &tv);
    close(fd);

    // If timeout and not writable, the IP matches the expected "blackhole" state
    (passes check)
    if (res <= 0 || !FD_ISSET(fd, &writeset)) {
        return 1;
    }
    return 0; // Unexpectedly connected, indicating being in a forged sandbox that
    takes over all traffic
}

```

3. Communication Protocol

The new variant's network communication protocol deeply references the IK pattern of the Noise protocol framework. Relying on the client's hardcoded server static public key and the ephemeral public key generated at runtime for ECDH, session keys are derived. Additionally, the protocol introduces a global MsgID across all phases, which is used for message

sequence verification and participates in rolling generation of new keys. This design cuts off the possibility of decrypting traffic on the network side using plaintext keys.

The entire communication lifecycle is strictly divided into two phases:

Phase A: Handshake/Verification Phase

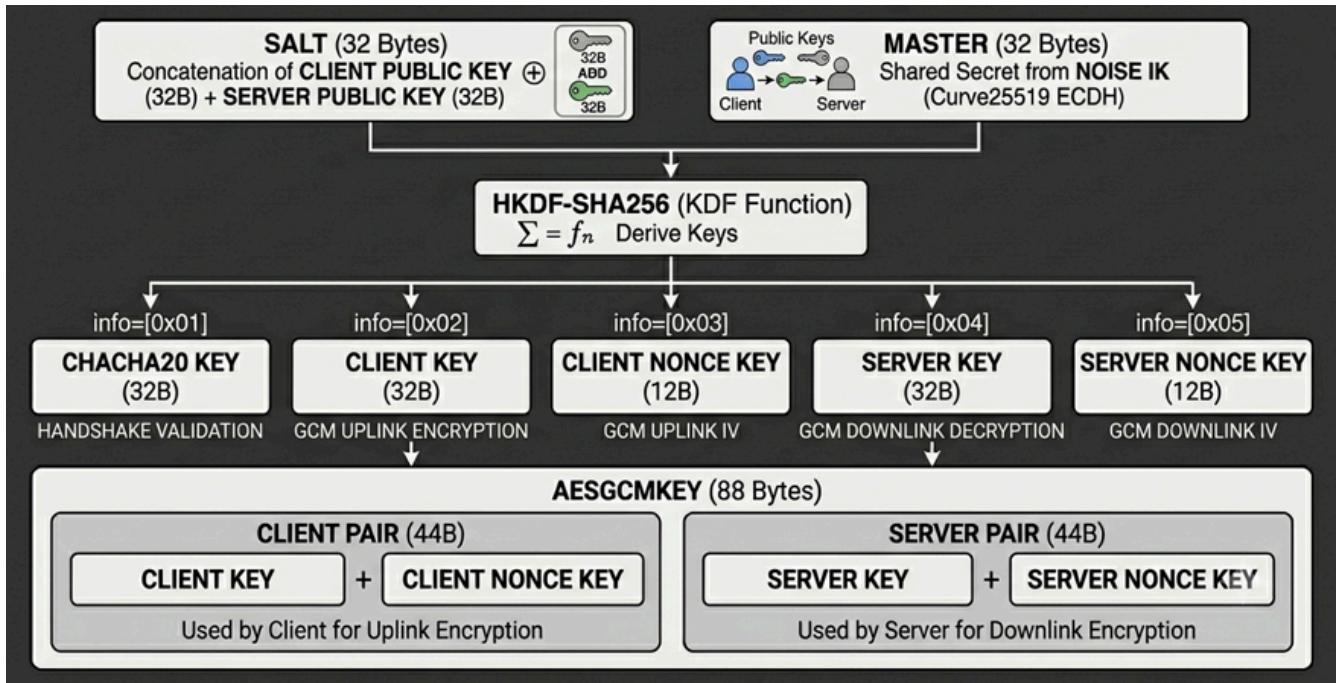
Message Encapsulation Format:

[illegible]

Transport Encryption: ChaCha20

Handshake Interaction Sequence:

1. **Key Exchange:** The client uploads a 32-byte client ephemeral public key; then receives a 32-byte public key sent by the server.
2. **KDF Derivation:** Both parties use their own private keys and the counterparty's public key to complete a modified key exchange, taking the shared secret as master and the public key concatenation as salt, and derive the subsequent chacha20Key and aesGCMKey through **HKDF-SHA256**:



3. **Four-Step Compliance Verification:** Based on chacha20Key + HMAC - SHA256, strict identity confirmation is established:

- **login (0xa0):** The client generates 16 bytes of random ciphertext and reports the current architecture, number of CPU cores, and memory size.
- **verify (0xa1):** The server returns a random message, along with the `login_hmac` for the login phase.
- **confirm (0xa2):** After successful verification, the client sends the `verify_hmac` and the local unique `botid` (a 64-byte hex string) to the server.
- **ack (0xa3):** The server sends an acknowledgement packet, officially announcing the end of the handshake phase.

Transport Encryption: AES-GCM scheme.

To maximize the ability to counter man-in-the-middle (MITM) attacks, this phase employs an **independent key scheme for uplink and downlink separation** during data encryption and decryption. The previously derived 88-byte aesGCMKey is precisely split into four parts:

- **Client sending (uplink) traffic:** Encrypted using `clientKey` and `clientNonceKey`.
- **Server receiving (downlink) commands:** Decrypted using `serverKey` and `serverNonceKey`.

Command Map:

msgType (Hex/Dec)	Command Name	Behavior Description
3 / 8	Attack	Launch DDoS attack task (supports various mixed flood attacks)
9	Stop Attack	Emergency stop of the current DDoS attack
10	Update	Remotely fetch new variant samples and perform hot update
11	Get Status	Report the current controlled host's alive status and resource usage
14	Update C2	Dynamically deliver and switch to new infrastructure C2 domains/IPs

Loader SHA1

```
8315f650e9e4f67c00277b076ab304eed23db47d
6aa791c76b3107fca9d57b7ecea8f46d97d83738
4d11bd496da82d15b3ed13050f414e44f5a892d4
d39a3ee96be6b8f5238cb1253514ab55c88f714c
```

C2 Domain

```
gayporn.twilightparadox.com
bigniggadick.ignorelist.com
```

ilovefemboy.mooo.com
igmc.duckdns.org
qewqewqewqtq.duckdns.org
qewqewqewqtqthree.duckdns.org
qewqewqewqtqtwo.duckdns.org

disciplinenahidwin.st
criminalcloudflare.online
dhdsjsdjxc.duckdns.org
fcfrfxrfrsfs5f.duckdns.org