

VoidLink: The Cloud-Native Malware Framework

 research.checkpoint.com/2026/voidlink-the-cloud-native-malware-framework

pedrod@checkpoint.com

January 13, 2026



Key takeaways

- **VoidLink** is an advanced malware framework made up of custom loaders, implants, rootkits, and modular plugins designed to maintain long-term access to **Linux systems**. The framework includes multiple **cloud-focused capabilities and modules**, and is engineered to operate reliably in **cloud and container environments** over extended periods.
- VoidLink's architecture is extremely flexible and **highly modular**, centered around a **custom Plugin API** that appears to be inspired by Cobalt Strike's Beacon Object Files (BOF) approach. This API is used in more than **30+ plug-in modules** available by default.
- VoidLink employs **multiple Operational Security (OPSEC) mechanisms**, including **runtime code encryption**, **self-deletion upon tampering**, and **adaptive behavior based on the detected environment**, alongside a range of **user-mode and kernel-level rootkit capabilities**.
- The framework appears to be **built and maintained by Chinese-affiliated developers** (exact affiliation remains unclear) and is **actively evolving**. Its overall design and thorough documentation suggest it is intended for commercial purposes.
- The developers demonstrate a high level of technical expertise, with strong proficiency across multiple programming languages, including Go, Zig, C, and modern frameworks such as React. In addition, the attacker possesses in-depth knowledge of sophisticated operating system internals, enabling the development of advanced and complex solutions.

VoidLink – a Cloud-First Malware Framework

In December 2025, Check Point Research identified a small cluster of previously unseen Linux malware samples that appear to originate from a Chinese-affiliated development environment. Many of the binaries included debug symbols and other development artifacts, suggesting we were looking at in-progress builds rather than a finished, widely deployed tool. The speed and variety of changes across the samples indicate a framework that is being iterated upon quickly to achieve broader, real-world use.

The framework, internally referred to by its original developers as VoidLink, is a cloud-first implant written in Zig and designed to operate in modern infrastructure. It can recognize major cloud environments and detect when it is running inside Kubernetes or Docker, then tailor its behavior accordingly. VoidLink also harvests credentials associated with cloud environments and standard source code version control systems, such as Git, indicating that software engineers may be a potential target, either for espionage activities or possible future supply-chain-based attacks.

VoidLink’s feature set is unusually broad. It includes rootkit-style capabilities (LD_PRELOAD, LKM, and eBPF), an in-memory plugin system for extending functionality, and adaptive stealth that adjusts runtime evasion based on the security products it detects, favoring operational security over performance in monitored environments. It also supports multiple command-and-control channels, including HTTP/HTTPS, ICMP, and DNS tunneling, and can form P2P/mesh-style communication between compromised hosts. In the latest samples, most components appear to be close to completion, alongside a functional C2 server and a dashboard front end integrated into a single ecosystem.

The framework’s intended use remains unclear, and as of this writing, no evidence of real-world infections has been observed. The way it is built suggests it may ultimately be positioned for commercial use, either as a product offering or as a framework developed for a customer.

Command and Control Panel

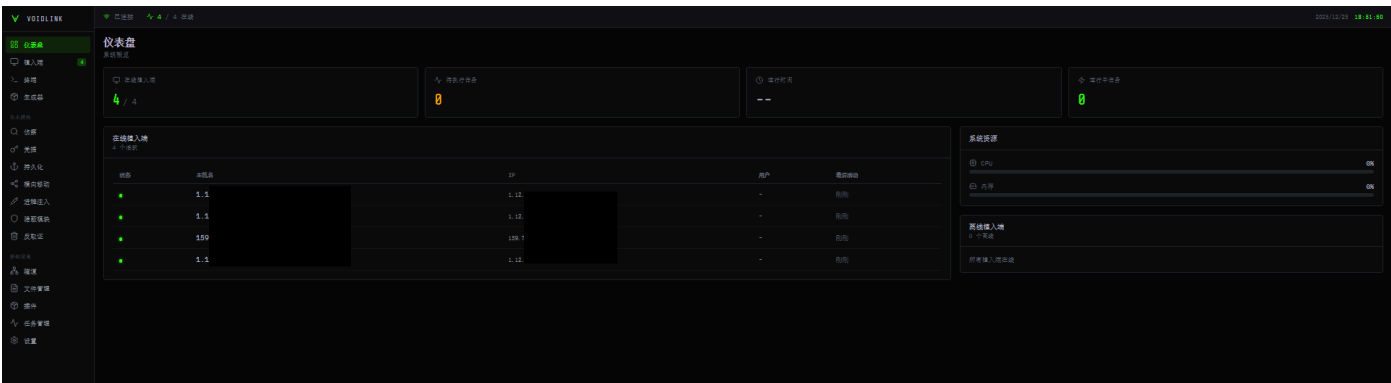


Figure 1 – Main Panel

To best manage an attack, VoidLink ships with a web-based dashboard that provides the operator with complete control over the running agents, implants, and plugins. This interface is localized for Chinese-affiliated operators, but the navigation follows a familiar C2 layout: a left sidebar groups pages into **Dashboard**, **Attack**, and **Infrastructure**. The Dashboard section covers the core operator loop (agent manager, built-in terminal, and an implant builder). In contrast, the Attack section organizes post-exploitation activity such as reconnaissance, credential access, persistence, lateral movement, process injection, stealth, and evidence wiping.

Dashboard	Attack	Infrastructure
Implants	Reconnaissance	Tunneling
Terminal	Credentials	File Management
Builder	Persistence	Plugin Management
	Lateral Movement	Task Management
	Process Injection	Set Up
	Hidden Modules	
	Wipe Evidence	

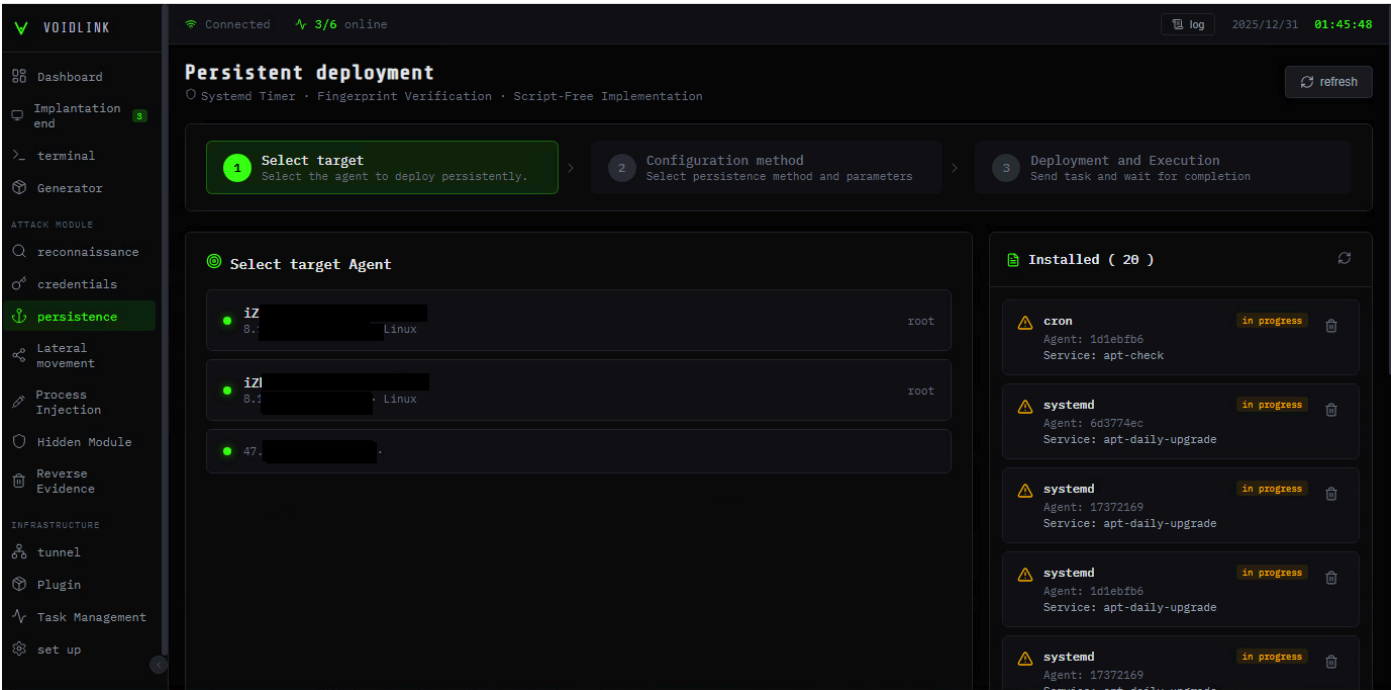


Figure 2 – Persistence Panel (Translated)

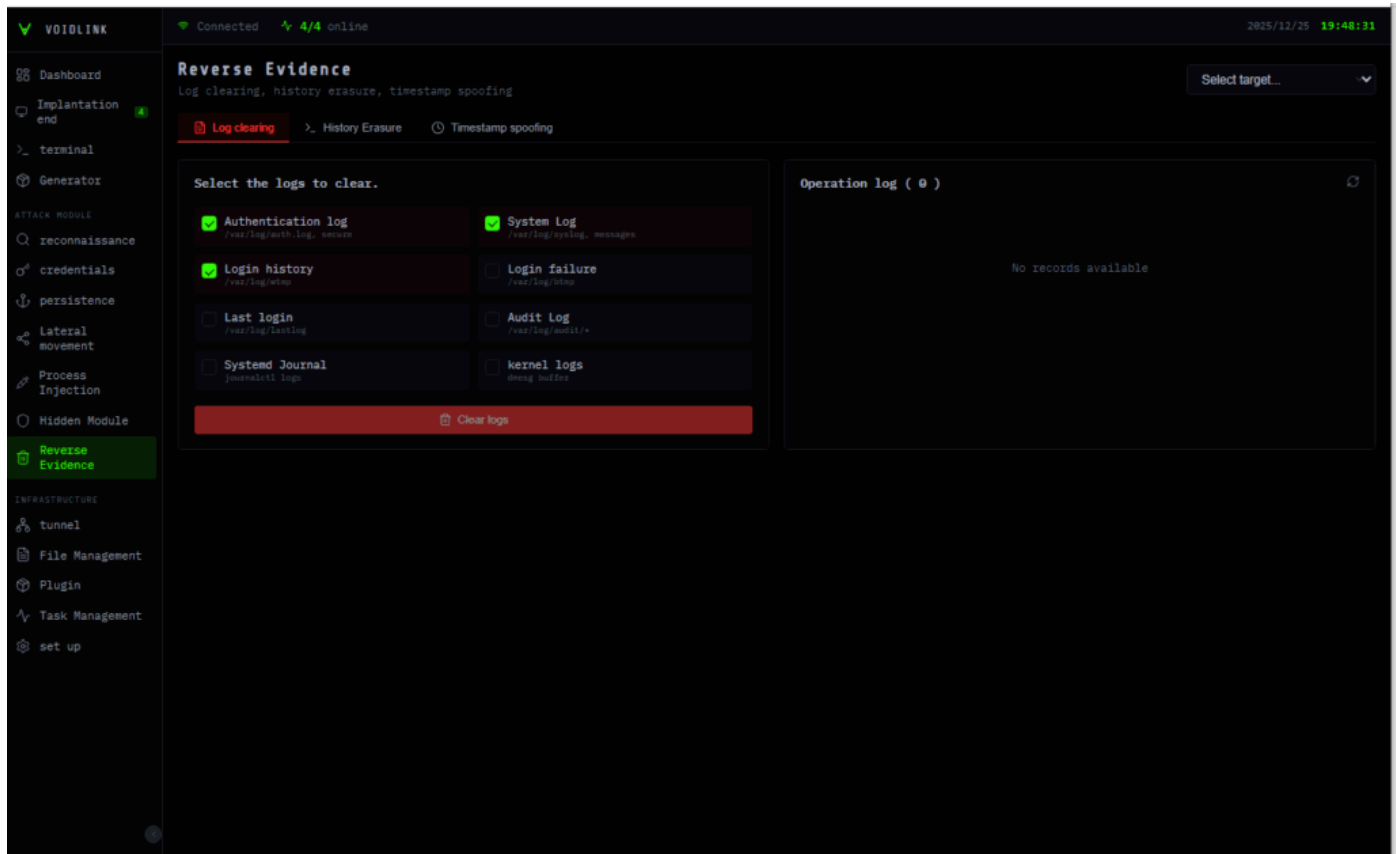


Figure 3 – Wipe Evidence Panel (Translated)

The **Generator** panel acts as the build interface for VoidLink, enabling the threat actor to generate additional, customized implant variants on demand. From this screen, the operator can select the desired capability set and tune the overall evasion posture. It also exposes operational parameters such as the implant’s heartbeat or beaconing interval, allowing the actor to balance responsiveness against stealth by controlling how frequently the implant checks in and executes tasks. All these parameters can also be changed at runtime.

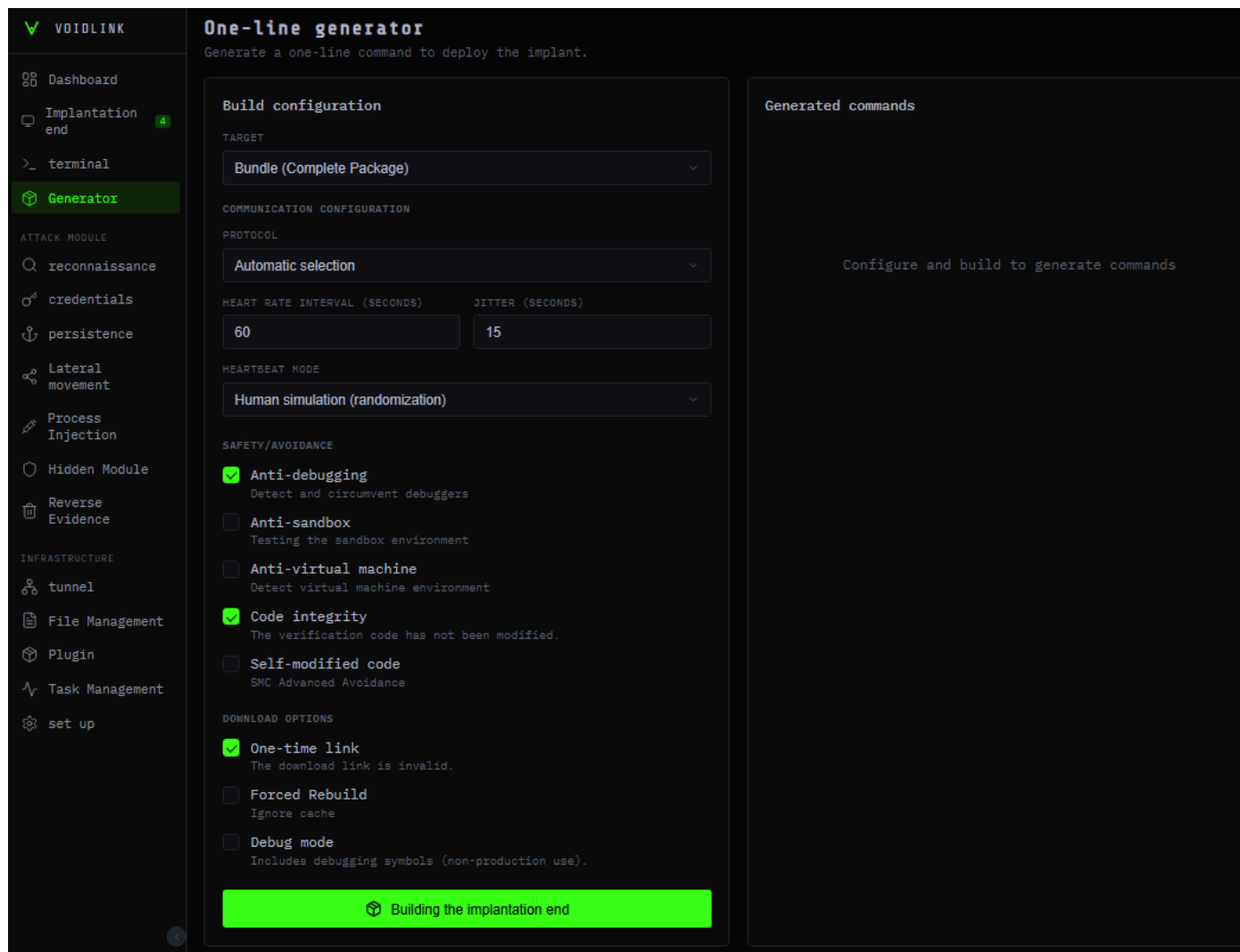


Figure 4 – Builder Panel (Translated)

The most interesting component of the dashboard is the plugin management panel. It allows the operator to deploy selected modules to victims and to upload custom modules. At the time of our research, 37 plugins were available, organized into several categories: Tools, Anti-Forensics, Reconnaissance, Containers, Privilege Escalation, Lateral Movement, and “Others” (see “Plugin System” below).

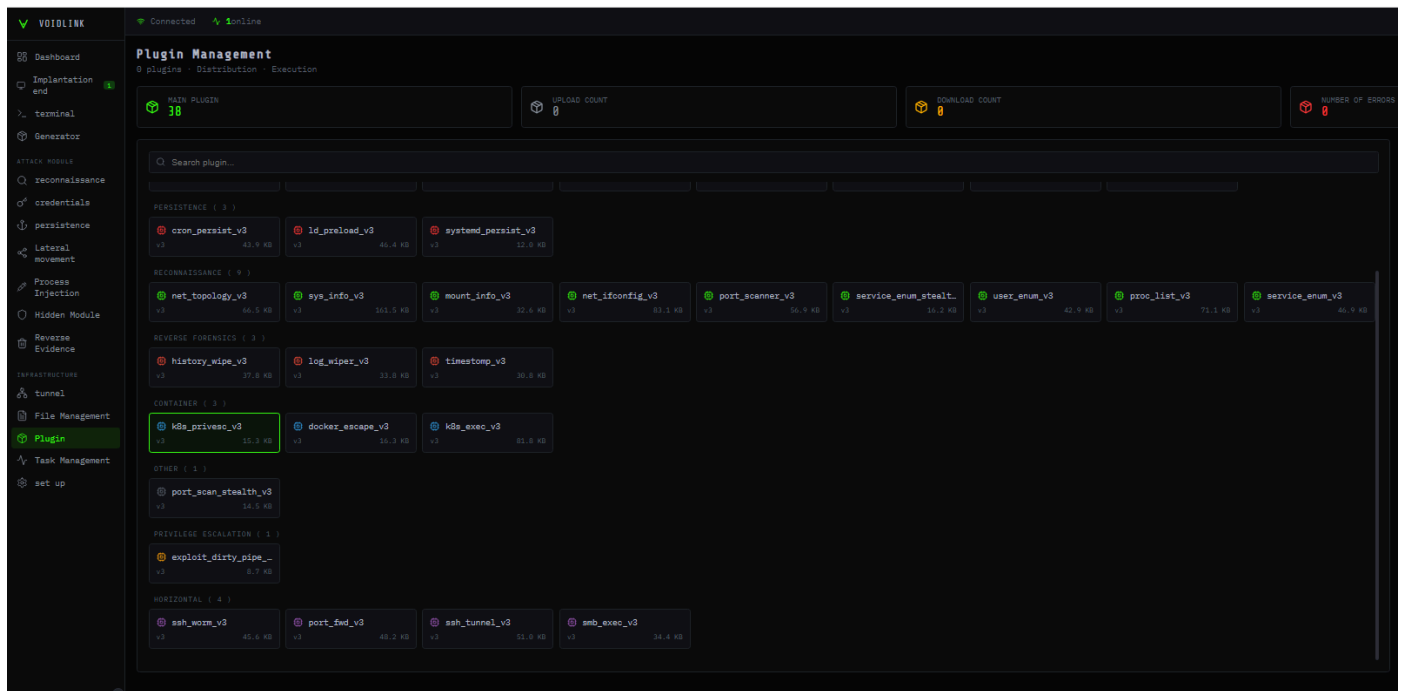


Figure 5 – Plugins Panel

Technical Overview

VoidLink is an impressive piece of software, written in Zig for Linux, and it is far more advanced than typical Linux malware. At its base, it features a conventional core that maintains implant stability. The core manages global state, communications, and task execution. This well-designed core hosts several features on top that make the malware a full-fledged C2 framework.

VoidLink is delivered through a two stage loader, where the final implant has core modules embedded, but external code can be downloaded at runtime as plugins:

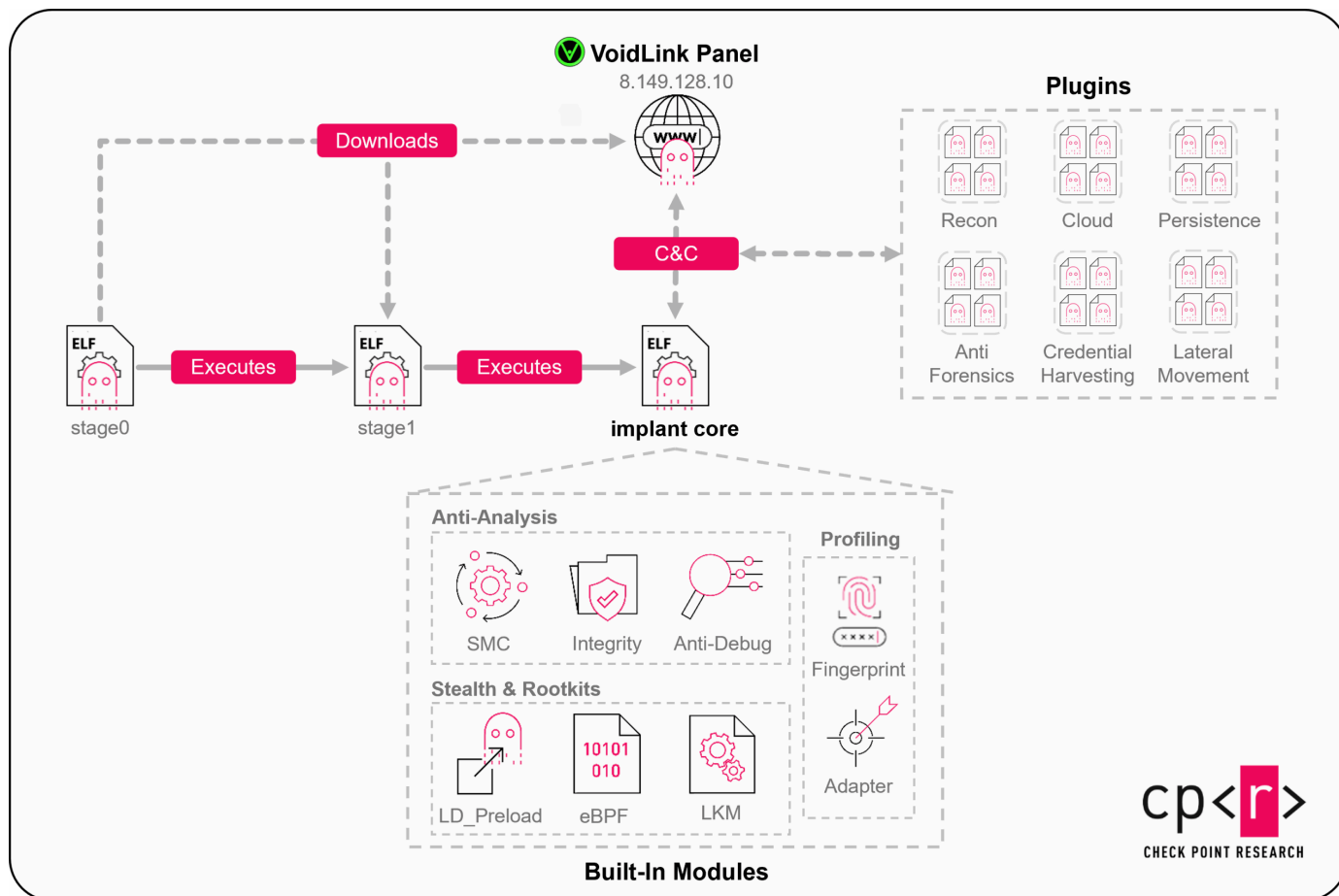


Figure 6 – VoidLink High Level Overview

Cloud-First Tradecraft

VoidLink is a cloud-first Linux implant. Once a machine is infected, it surveys the compromised system and can detect which cloud provider the infected machine is running under. Currently, VoidLink can detect AWS, GCP, Azure, Alibaba, and Tencent, with plans to add detections for Huawei, DigitalOcean, and Vultr. For all these cloud providers, VoidLink queries additional information on instance metadata using the respective vendor's API.

```

anyerror __cdecl adapter_cloud_detect_CloudDetector_fetchAWSMetadata(adapter_cloud_detect_CloudDetector *self)
{
    unsigned __int64 v1; // rsi
    __u8 v2; // r8
    __u8 v3; // r8
    __u8 v4; // r8
    u8 *path_4; // [rsp+38h] [rbp-48h]

    adapter_cloud_detect_CloudDetector_readCloudInitData(self);
    if ( *(_QWORD *) (v1 + 16) )
        return 0;
    path_4 = adapter_cloud_detect_CloudDetector_readEnvVar(self, (__u8) __PAIR128__("AWS_REGION", v1)).ptr;
    if ( path_4 )
    {
        adapter_cloud_detect_CloudMetadata_setRegion(
            (adapter_cloud_detect_CloudMetadata *) self,
            (__u8) __PAIR128__((unsigned __int64) path_4, v1 + 16));
    }
    else
    {
        v2.len = (usize) "/latest/meta-data/placement/region";
        v2.ptr = (u8 *) 80;
        adapter_cloud_detect_CloudDetector_fetchMetadataHTTP(
            self,
            (__u8) __PAIR128__("169.254.169.254", v1),
            0xFu,
            v2,
            (__u8) 0x22uLL);
        v3.ptr = (u8 *) 80;
        v3.len = (usize) "/latest/meta-data/instance-id";
        adapter_cloud_detect_CloudDetector_fetchMetadataHTTP(
            self,
            (__u8) __PAIR128__("169.254.169.254", v1),
            0xFu,
            v3,
            (__u8) 0x1DuLL);
        v4.ptr = (u8 *) 80;
        v4.len = (usize) "/latest/meta-data/instance-type";
        adapter_cloud_detect_CloudDetector_fetchMetadataHTTP(
            self,
            (__u8) __PAIR128__("169.254.169.254", v1),
            0xFu,
            v4,
            (__u8) 0x1FuLL);
    }
    return 0;
}

```

Figure 7 – Querying AWS metadata

In addition to cloud detection, it collects vast amounts of information about the infected machine, enumerating its hypervisor and detecting whether it is running in Docker container or a Kubernetes pod.

To ease data exfiltration, privilege escalation, and lateral movement in containerized environments, several post-exploitation modules are implemented—from automated container escapes over secret extraction to dedicated lateral movement commands.

Ultimately, the goal of this implant appears to be stealthy, long-term access, surveillance, and data collection.

Plugin Development API

In addition to the core modules and commands, the VoidLink framework offers an extensive development API, similar to (and likely inspired by) Cobalt Strike and its Beacon API. The API is set up during the malware's initialization by creating an export table that contains all available APIs.

```

v110 = plugin_export_table_ExportTable_add(
    export_table,
    (__u8)__PAIR128__("beacon_net_accept", v9),
    (void *)0x11,
    (u32)plugin_export_table_impl_beacon_net_accept);
if ( v110 )
{
    Thread_Mutex_unlock((Thread_Mutex *)export_table);
    builtin_returnError();
    return v110;
}
else
{
    v109 = plugin_export_table_ExportTable_add(
        export_table,
        (__u8)__PAIR128__("beacon_get_proc_addr", v10),
        (void *)0x14,
        (u32)plugin_export_table_impl_beacon_get_proc_addr);
    if ( v109 )
    {
        Thread_Mutex_unlock((Thread_Mutex *)export_table);
        builtin_returnError();
        return v109;
    }
    else
    {
        v108 = plugin_export_table_ExportTable_add(
            export_table,
            (__u8)__PAIR128__("beacon_file_read", v11),
            (void *)0x10,
            (u32)plugin_export_table_impl_beacon_file_read);
        if ( v108 )
        {
            Thread_Mutex_unlock((Thread_Mutex *)export_table);
            builtin_returnError();
            return v108;
        }
        else
    }
}

```

Figure 8 – Development API Export Table

When developing a VoidLink plugin, a developer can then reference these APIs to e.g. read files, create socket connections, execute files, resolve routines from shared objects or log to the C2 console. The whole API operates on direct syscalls, bypassing *libc* hooks.

Adaptive Stealth

Upon launch, VoidLink enumerates installed security products and hardening measures, including Linux EDRs and kernel hardening technologies. However, this information is not only returned to the operator but used to calculate a **risk score** for the environment and suggest an **evasion strategy**, which is then used in other modules to influence their behavior, so that, for example, a port scan is executed more slowly and with greater control in an environment where monitoring is implemented, and the risk is comparatively high. This pattern of adaptive stealth is one of VoidLink's core principles and is applied throughout the framework.

```
stealth_edr_detector_EdrType::none = 0x0,  
stealth_edr_detector_EdrType::falco = 0x1,  
stealth_edr_detector_EdrType::sysdig = 0x2,  
stealth_edr_detector_EdrType::sentinel_one = 0x3,  
stealth_edr_detector_EdrType::crowdstrike = 0x4,  
stealth_edr_detector_EdrType::carbon_black = 0x5,  
stealth_edr_detector_EdrType::palo_alto = 0x6,  
stealth_edr_detector_EdrType::ossec = 0x7,  
stealth_edr_detector_EdrType::wazuh = 0x8,  
stealth_edr_detector_EdrType::auditd = 0x9,  
stealth_edr_detector_EdrType::sysmon_linux = 0xA,  
stealth_edr_detector_EdrType::unknown = 0xB,
```

Figure 9 – Detected EDRs

Rootkit Modules

Another noteworthy component is a monitor that helps VoidLink blend in with normal system activity. It builds a profile of host behavior by reading machine telemetry (CPU, memory, network, and active processes), parsing it, and creating adaptive intervals for communication with the C2, with constraints such as working hours and low-activity times.

A stealth module integrates advanced concealment techniques, including kernel-level techniques. It maintains a family of rootkits tailored to multiple kernel versions. It couples them with eBPF programs that can hook sensitive paths without requiring a traditional LKM on newer, locked-down systems. VoidLink handles rootkit deployment once again, based on the environment in which it runs, and chooses the right rootkit to deploy accordingly. Depending on the Kernel version and supported features, the following rootkits are chosen:

- LD_PRELOAD: When the “kernel” flag is disabled, OR the kernel version is < 4.0
- eBPF: For a Kernel version ≥ 5.5 with eBPF support
- LKM: Kernel ≥ 4.0

```

// Select rootkit deployment method depending on environment
v10 = stealth_stealth_manager_StealthManager_selectBestMethod(self);
switch ( (v10 + 4) & 7 )
{
    case 0:
        v8 = stealth_stealth_manager_StealthManager_activateLdPreload(self);
        if ( !v8 )
            goto LABEL_7;
        builtin_returnError();
        result = v8;
        break;
    case 1:
        v7 = stealth_stealth_manager_StealthManager_activateProcHide(self);
        if ( !v7 )
            goto LABEL_7;
        builtin_returnError();
        result = v7;
        break;
    case 2:
    case 3:
        debug_FullPanic__function__defaultPanic____memcpyAlias_0();
        return result;
    case 4:
        return 0;
    case 5:
    case 6:
        v6 = stealth_stealth_manager_StealthManager_activateEbpF(self, method);
        if ( !v6 )
            goto LABEL_7;
        builtin_returnError();
        result = v6;
        break;
    case 7:
        v9 = stealth_stealth_manager_StealthManager_activateLkm(self);
        if ( v9 )

```

Figure 10 – Rootkit deployment depending on environment

Using the rootkits, the implant can selectively hide its processes, files, and network sockets, as well as hide the rootkit modules themselves.

Command and Control

At the network level, VoidLink attempts to make outbound network connections appear legitimate; several modules conceal the traffic. There is a layer responsible for HTTP camouflage, which attempts to make requests appear legitimate.

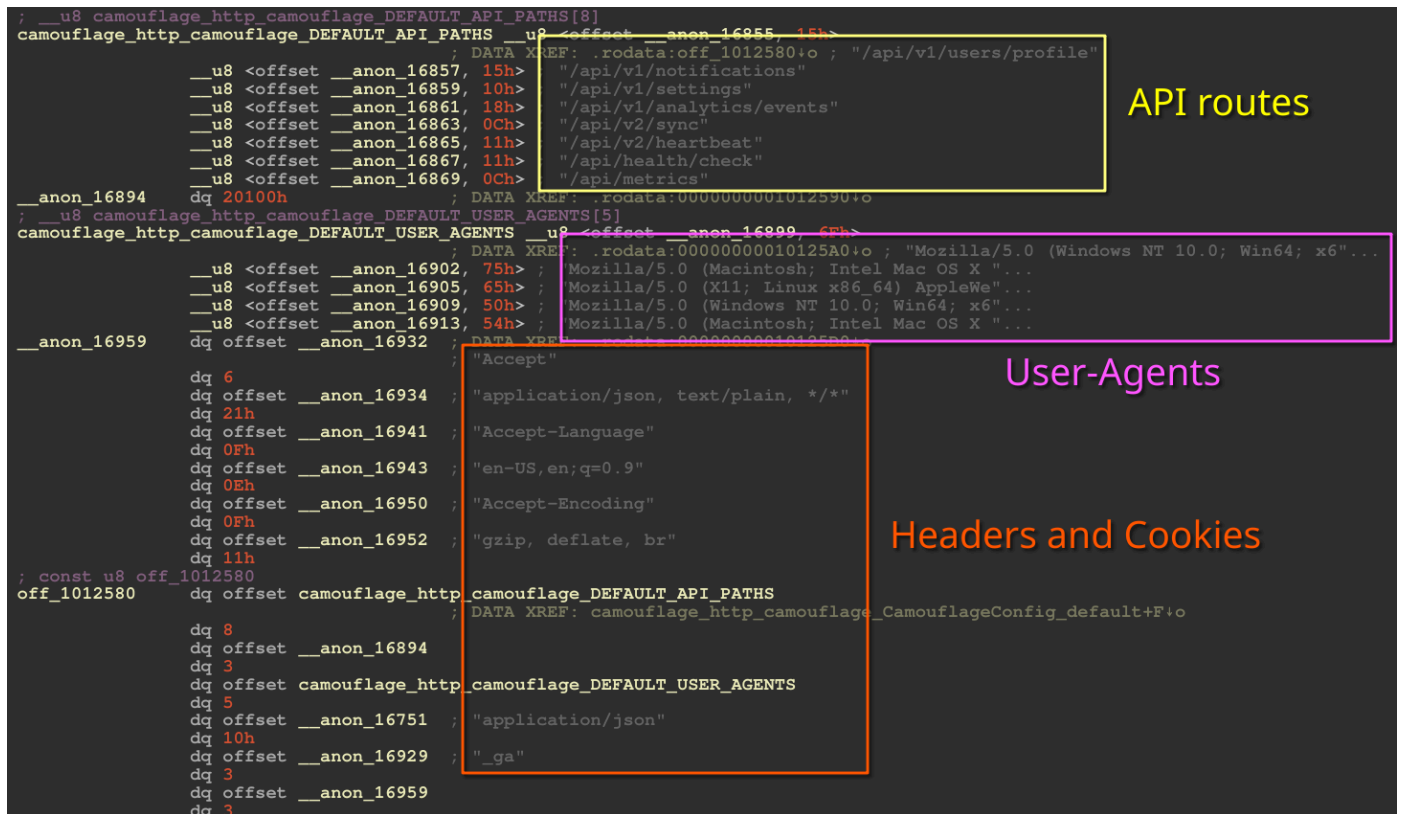


Figure 11 – HTTP camouflage configuration

Requests, as well as exfiltrated files, can be hidden in various ways, including via PNG-like blobs, standard website content (JS/CSS/HTML), or by mimicking API traffic. VoidLink supports multiple transport protocols: HTTP/1.1, HTTP/2, WebSocket, DNS, and ICMP. All are managed through a protocol dubbed VoidStream by the developers. VoidStream handles encryption and message parsing for all of the previously mentioned protocols.

While not fully implemented, analyzed samples also contain methods for mesh C2—a peer-to-peer networking method in which infected machines form a mesh network, routing packets in-between each other without needing outbound internet access.

Anti-Analysis

VoidLink deploys several anti-analysis mechanisms. In addition to anti-debugging techniques, VoidLink detects various debuggers and monitoring tools. VoidLink also runs runtime integrity checks to identify potential hooks and patches. Additionally, a self-modifying code option decrypts protected code regions at runtime and encrypts them while not in use, evading runtime memory scanners. If VoidLink detects any type of tampering, it deletes itself.

Anti-forensic modules ensure that any traces left by VoidLink are also deleted. The malware cleans command histories, login records, system logs, and dropped files, all while ensuring that files are not only unlinked from the file system but also overwritten with random data to prevent forensic recovery.

Plugin System

VoidLink's plugin system effectively expands its framework, evolving from an implant to a fully featured post-exploitation framework. Again, similar to Cobalt Strike and its Beacon Object Files, plugins come as (ELF) object files that are loaded at runtime and are executed in-memory.

The plugins available by default cover various categories:

Recon	Detailed system and environment profiling, user and group enumeration, process and service discovery, filesystem and mount mapping, and mapping of local network topology and interfaces.
Cloud	Kubernetes and Docker discovery and privilege-escalation helpers, container escape checks, and probes for misconfigurations that allow attackers to break out of pods or containers into the underlying host or cluster.
Credential Harvesting	Multiple plugins to harvest credentials and secrets, including SSH keys, git credentials, local password material, browser credentials and cookies, tokens, and API keys in environment variables or process arguments, and items stored in the system keyring.
Utilities and lateral movement	Post-exploitation tooling includes file management, interactive and non-interactive shells, port forwarding and tunneling, and an SSH-based worm that attempts to connect to known hosts and spread laterally.
Persistence	Persistence Plugins that establish persistence via native mechanisms like dynamic linker abuse, cron jobs, and system services.
Anti-forensics	Components that wipe or edit logs and shell history based on keywords and perform timestomping of files to disrupt forensic timelines.

Together, these plugins sit atop an already sophisticated core implementation, enriching VoidLink's capabilities beyond cloud environments to developer and administrator workstations that interface directly with those cloud environments, turning any compromised machine into a flexible launchpad for deeper access or supply-chain compromise. The appendix lists all plugins we analyzed, with a summarized description of each.

Conclusion

VoidLink is a rapidly developing Linux command and control framework, tailored towards modern cloud environments with a focus on stealth. The sheer number of features and its modular architecture show that the authors intended to create a sophisticated, modern and feature-rich framework. VoidLink aims to automate evasion as much as possible, profiling an environment and choosing the most suitable strategy to operate in it. Augmented by kernel mode tradecraft and a vast plugin ecosystem, VoidLink enables its operators to move through cloud environments and container ecosystems with adaptive stealth.

While the larger part of the malware landscape targets Windows, the Linux platform is often an underlooked target by both malware developers and defenders. The creation of a framework dedicated to the Linux platform, and more specifically, cloud environments, shows that these platforms are a valid target for threat actors.

Although it is not clear if the framework is intended to be sold as a legitimate penetration testing tool, as a tool for the criminal underground, or as a dedicated product for a single customer, defenders should proactively secure their Linux, cloud, and container environments and be prepared to defend against advanced threats such as VoidLink.

Protections

Check Point Threat Emulation and Harmony Endpoint provide comprehensive coverage of attack tactics, file types, and operating systems, and protect against the attacks and threats described in this report.

Indicators of Compromise

Stage 0	70aa5b3516d331e9d1876f3b8994fc8c18e2b1b9f15096e6c790de8cdadb3fc9
Stage 1	13025f83ee515b299632d267f94b37c71115b22447a0425ac7baed4bf60b95cd
VoidLink Implants	05eac3663d47a29da0d32f67e10d161f831138e10958dcd88b9dc97038948f69 15cb93d38b0a4bd931434a501d8308739326ce482da5158eb657b0af0fa7ba49 6850788b9c76042e0e29a318f65fceb574083ed3ec39a34bc64a1292f4586b41 6dcfe9f66d3aef1efd7007c588a59f69e5cd61b7a8eca1fb89a84b8ccef13a2b 28c4a4df27f7ce8ced69476cc7923cf56625928a7b4530bc7b484eec67fe3943 e990a39e479e0750d2320735444b6c86cc26822d86a40d37d6e163d0fe058896 4c4201cc1278da615bacf48deef461bf26c343f8cbb2d8596788b41829a39f3f

Appendix

Plugin name	Description
ssh_harvester_stealth_v3.o	Plugin that collects SSH private keys and configuration data from the host for later use or exfiltration.
ssh_harvester_v3.o	Plugin that collects SSH private keys and configuration data from the host for later use or exfiltration.
port_scan_stealth_v3.o	Basic port scanner that checks a specified target for open network ports.
port_scanner_v3.o	Basic port scanner that checks a specified target for open network ports.
sys_info_v3.o	Information-gathering plugin that collects general system details such as host identifiers, OS information, and environment data.
ssh_worm_v3.o	Self-propagating SSH worm module that iterates over known hosts and attempts credential-based logins in a throttled, low-profile manner.
term_ptty_v3.o	Interactive backdoor, allows remote command execution and file read or write operations.
file_mgr_v3.o	File manager plugin that lets the operator browse directories and upload, download, create, rename, or delete files and folders.
simple_test.o	Test plugin, does nothing.
port_fwd_v3.o	Port-forwarding module that sets up tunnels to expose internal services on the compromised host to the operator.
k8s_privesc_v3.o.b	Kubernetes privilege-escalation helper that looks for ways to break out of a pod or raise privileges and reports any viable paths.
systemd_persist_v3.o	Persistence plugin that creates or modifies systemd service definitions so the implant is launched automatically on startup.
docker_escape_v3.o	Container-escape module that checks and attempts known Docker breakout techniques, sending any interesting results back to the C2.
cron_persist_v3.o	Persistence component that installs or alters cron jobs for persistence.
hello_plugin_v3.o	Lightweight reconnaissance plugin that records basic runtime details such as process ID, user ID, hostname, and current directory.

mimipenguin_lite_v3.o	Credential-harvesting module that inspects running processes and their arguments to extract passwords or other sensitive secrets.
browser_stealer_v3.o	Browser data theft plugin that targets Chrome and Firefox to extract stored credentials, cookies, and similar artifacts.
log_wiper_v3.o	Log-cleaning tool that searches common log files for specific keywords and removes matching entries to obscure activity.
timestomp_v3.o	Timestomping module that alters file and directory timestamps to disrupt forensic reconstruction of events.
service_enum_stealth_v3.o	Stealthy service-enumeration plugin that probes open ports to identify the underlying services, banners, and version details.
proc_list_v3.o	Process discovery component that enumerates running processes and collects detailed information about each one.
keyring_dump_v3.o	Keyring-dumping module that extracts secrets stored in the system keyring.
mount_info_v3.o.b	Filesystem reconnaissance plugin that lists mounted disks and volumes.
user_enum_v3.o	Account-enumeration module that gathers information about local users, groups, and their relationships on the system.
k8s_exec_v3.o	Kubernetes discovery plugin that queries the cluster to enumerate accessible namespaces, pods, and related resources.
ld_preload_v3.o	Persistence mechanism that abuses dynamic linker hooks such as LD_PRELOAD so the implant code runs when targeted programs start.
hello_plugin.o	Basic info-gathering plugin that captures minimal runtime context like process ID, user, hostname, and working directory.
passwd_dump_v3.o	Credential-dumping module that reads local account databases to collect usernames and associated password hashes.
history_wipe_v3.o	History-manipulation tool that removes shell history entries matching a pattern and can disable future history logging.
env_vars_v3.o	Environment-variable scanner that searches exported variables for secrets such as API keys, access tokens, and credentials.
net_topology_v3.o	Network reconnaissance module that uses ARP and routing information to map nearby hosts and overall network topology.
ssh_tunnel_v3.o	Tunneling plugin that establishes SSH-based tunnels to route traffic through the compromised host.

exploit_dirty_pipe_v3.o	Privilege-escalation module that implements a Dirty Pipe, compiled for MAC with unused code, wouldn't work.
net_ifconfig_v3.o	Network configuration enumerator that collects interface information, IP addresses, and related network settings.
service_enum_v3.o	Service-enumeration module that connects to discovered ports to identify services, capture banners, and gather version information.

[BACK TO ALL POSTS](#)