

The HoneyMyte APT evolves with a kernel-mode rootkit and a ToneShell backdoor

By Noushin Shabab

Published: 2025-12-29 · Archived: 2026-05-09 02:01:38 UTC

Overview of the attacks

In mid-2025, we identified a malicious driver file on computer systems in Asia. The driver file is signed with an old, stolen, or leaked digital certificate and registers as a mini-filter driver on infected machines. Its end-goal is to inject a backdoor Trojan into the system processes and provide protection for malicious files, user-mode processes, and registry keys.

Our analysis indicates that the final payload injected by the driver is a new sample of the ToneShell backdoor, which connects to the attacker's servers and provides a reverse shell, along with other capabilities. The ToneShell backdoor is a tool known to be used exclusively by the HoneyMyte (aka Mustang Panda or Bronze President) APT actor and is often used in cyberespionage campaigns targeting government organizations, particularly in Southeast and East Asia.

The command-and-control servers for the ToneShell backdoor used in this campaign were registered in September 2024 via NameCheap services, and we suspect the attacks themselves to have begun in February 2025. We've observed through our telemetry that the new ToneShell backdoor is frequently employed in cyberespionage campaigns against government organizations in Southeast and East Asia, with Myanmar and Thailand being the most heavily targeted.

Notably, nearly all affected victims had previously been infected with other HoneyMyte tools, including the ToneDisk USB worm, PlugX, and older variants of ToneShell. Although the initial access vector remains unclear, it's suspected that the threat actor leveraged previously compromised machines to deploy the malicious driver.

Compromised digital certificate

The driver file is signed with a digital certificate from **Guangzhou Kingteller Technology Co., Ltd.**, with a serial number of **08 01 CC 11 EB 4D 1D 33 1E 3D 54 0C 55 A4 9F 7F**. The certificate was valid from August 2012 until 2015.

We found multiple other malicious files signed with the same certificate which didn't show any connections to the attacks described in this article. Therefore, we believe that other threat actors have been using it to sign their malicious tools as well. The following image shows the details of the certificate.

Guangzhou Kingteller Technology Co.,Ltd.

Identity: Guangzhou Kingteller Technology Co.,Ltd.

Verified by: VeriSign Class 3 Code Signing 2010 CA

Expires: 27/08/15

▼ Details

Subject Name

C (Country): CN
ST (State): Guangdong
L (Locality): Guangzhou
O (Organisation): Guangzhou Kingteller Technology Co.,Ltd.
OU (Organisation Unit): Digital ID Class 3 - Microsoft Software Validation v2
CN (Common Name): Guangzhou Kingteller Technology Co.,Ltd.

Issued Certificate

Version: 3
Serial Number: 08 01 CC 11 EB 4D 1D 33 1E 3D 54 0C 55 A4 9F 7F
Not Valid Before: 2012-08-27
Not Valid After: 2015-08-27

Certificate Fingerprints

SHA1: 1F 33 28 56 26 E8 1F 48 45 93 6C 7A 2A 55 A4 6F 6D AD A6 51
MD5: 12 1A A7 02 F3 56 4B 06 B1 39 DD D8 0C A9 F8 1B

Technical details of the malicious driver

The filename used for the driver on the victim's machine is **ProjectConfiguration.sys**. The registry key created for the driver's service uses the same name, **ProjectConfiguration**.

The malicious driver contains two user-mode shellcodes, which are embedded into the .data section of the driver's binary file. The shellcodes are executed as separate user-mode threads. The rootkit functionality protects both the driver's own module and the user-mode processes into which the backdoor code is injected, preventing access by any process on the system.

API resolution

To obfuscate the actual behavior of the driver module, the attackers used dynamic resolution of the required API addresses from hash values.

The malicious driver first retrieves the base address of the **ntoskrnl.exe** and **fltmgr.sys** by calling **ZwQuerySystemInformation** with the **SystemInformationClass** set to **SYSTEM_MODULE_INFORMATION**. It then iterates through this system information and searches for the desired DLLs by name, noting the **ImageBaseAddress** of each.

Once the base addresses of the libraries are obtained, the driver uses a simple hashing algorithm to dynamically resolve the required API addresses from **ntoskrnl.exe** and **fltmgr.sys**.

The hashing algorithm is shown below. The two variants of the seed value provided in the comment are used in the shellcodes and the final payload of the attack.

```
func hashString(str []byte) uint64 {  
    var seed uint64 = 131313 // Seed values used in other modules: 1313131, 13131313  
    var hash uint64 = 0  
  
    for i := 0; i < len(str); i++ {  
        hash = hash*seed + uint64(str[i])  
    }  
  
    return hash  
}
```

Protection of the driver file

The malicious driver registers itself with the Filter Manager using **FltRegisterFilter** and sets up a pre-operation callback. This callback inspects I/O requests for **IRP_MJ_SET_INFORMATION** and triggers a malicious handler when certain **FileInformationClass** values are detected. The handler then checks whether the targeted file object is associated with the driver; if it is, it forces the operation to fail by setting **IOStatus** to **STATUS_ACCESS_DENIED**. The relevant **FileInformationClass** values include:

- **FileRenameInformation**
- **FileDispositionInformation**
- **FileRenameInformationBypassAccessCheck**
- **FileDispositionInformationEx**
- **FileRenameInformationEx**
- **FileRenameInformationExBypassAccessCheck**

These classes correspond to file-delete and file-rename operations. By monitoring them, the driver prevents itself from being removed or renamed – actions that security tools might attempt when trying to quarantine it.

Protection of registry keys

The driver also builds a global list of registry paths and parameter names that it intends to protect. This list contains the following entries:

- **ProjectConfiguration**
- **ProjectConfiguration\Instances**
- **ProjectConfiguration Instance**

To guard these keys, the malware sets up a **RegistryCallback** routine, registering it through **CmRegisterCallbackEx**. To do so, it must assign itself an *altitude* value. Microsoft governs altitude assignments for mini-filters, grouping them into Load Order categories with predefined altitude ranges. A filter driver with a low numerical altitude is loaded into the I/O stack below filters with higher altitudes. The malware uses a hardcoded starting point of **330024** and creates altitude strings in the format **330024.%l**, where *%l* ranges from 0 to 10,000.

The malware then begins attempting to register the callback using the first generated altitude. If the registration fails with **STATUS_FLT_INSTANCE_ALTITUDE_COLLISION**, meaning the altitude is already taken, it increments the value and retries. It repeats this process until it successfully finds an unused altitude.

The callback monitors four specific registry operations. Whenever one of these operations targets a key from its protected list, it responds with **0xC0000022 (STATUS_ACCESS_DENIED)**, blocking the action. The monitored operations are:

- **RegNtPreCreateKey**
- **RegNtPreOpenKey**
- **RegNtPreCreateKeyEx**
- **RegNtPreOpenKeyEx**

Microsoft designates the **320000–329999** altitude range for the *FSFilter Anti-Virus* Load Order Group. The malware's chosen altitude exceeds this range. Since filters with lower altitudes sit deeper in the I/O stack, the malicious driver intercepts file operations before legitimate low-altitude filters like antivirus components, allowing it to circumvent security checks.

Finally, the malware tampers with the altitude assigned to **WdFilter**, a key Microsoft Defender driver. It locates the registry entry containing the driver's altitude and changes it to **0**, effectively preventing WdFilter from being loaded into the I/O stack.

Protection of user-mode processes

The malware sets up a list intended to hold protected process IDs (PIDs). It begins with 32 empty slots, which are filled as needed during execution. A status flag is also initialized and set to 1 to indicate that the list starts out empty.

Next, the malware uses **ObRegisterCallbacks** to register two callbacks that intercept process-related operations. These callbacks apply to both **OB_OPERATION_HANDLE_CREATE** and **OB_OPERATION_HANDLE_DUPLICATE**, and both use a malicious pre-operation routine.

This routine checks whether the process involved in the operation has a PID that appears in the protected list. If so, it sets the **DesiredAccess** field in the **OperationInformation** structure to 0, effectively denying any access to the process.

The malware also registers a callback routine by calling **PsSetCreateProcessNotifyRoutine**. These callbacks are triggered during every process creation and deletion on the system. This malware's callback routine checks whether the parent process ID (PPID) of a process being deleted exists in the protected list; if it does, the malware removes that PPID from the list. This eventually removes the rootkit protection from a process with an injected backdoor, once the backdoor has fulfilled its responsibilities.

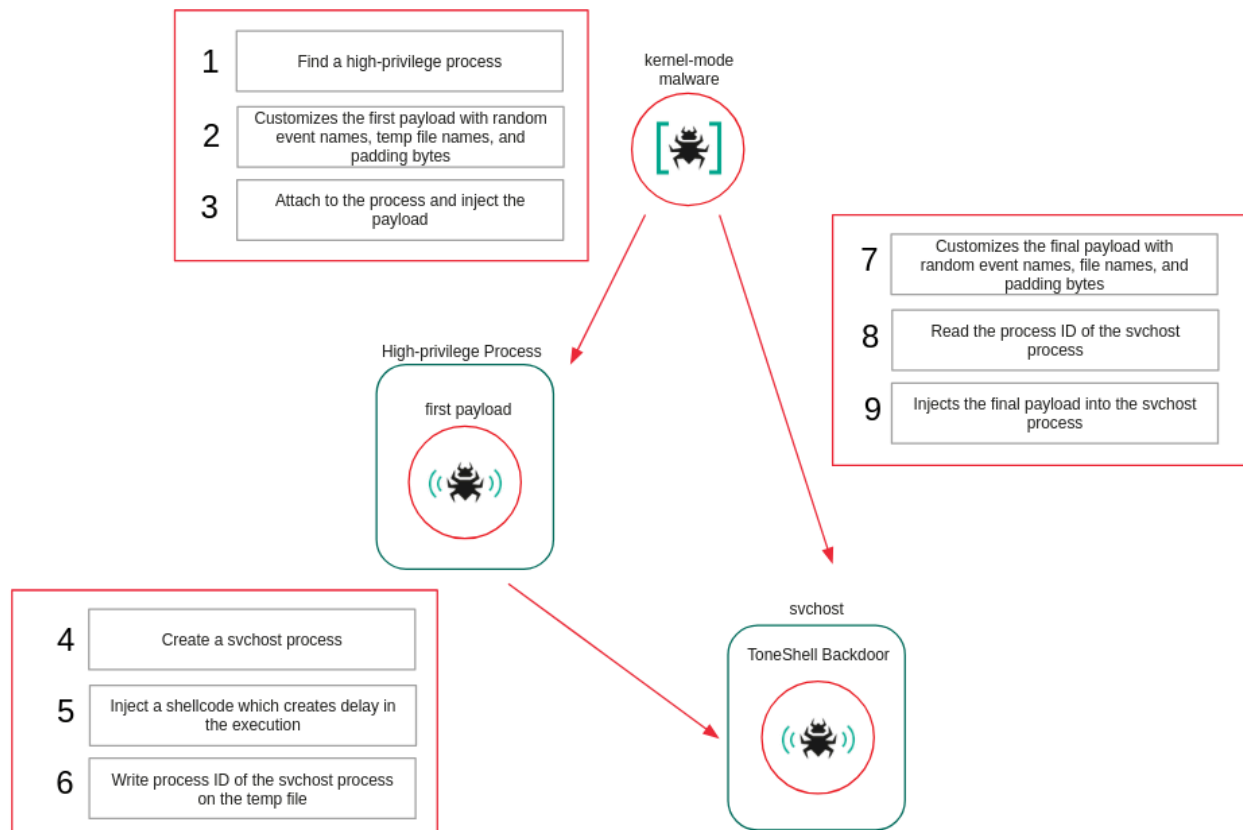
Payload injection

The driver delivers two user-mode payloads.

The first payload spawns an *svchost* process and injects a small delay-inducing shellcode. The PID of this new *svchost* instance is written to a file for later use.

The second payload is the final component – the ToneShell backdoor – and is later injected into that same *svchost* process.

Injection workflow:



The malicious driver searches for a high-privilege target process by iterating through PIDs and checking whether each process exists and runs under `SeLocalSystemSid`. Once it finds one, it customizes the first payload using random event names, file names, and padding bytes, then creates a named event and injects the payload by attaching its current thread to the process, allocating memory, and launching a new thread.

After injection, it waits for the payload to signal the event, reads the PID of the newly created *svchost* process from the generated file, and adds it to its protected process list. It then similarly customizes the second payload (ToneShell) using random event name and random padding bytes, then creates a named event and injects the payload by attaching to the process, allocating memory, and launching a new thread.

Once the ToneShell backdoor finishes execution, it signals the event. The malware then removes the *svchost* PID from the protected list, waits 10 seconds, and attempts to terminate the process.

ToneShell backdoor

The final stage of the attack deploys **ToneShell**, a backdoor previously linked to operations by the HoneyMyte APT group and discussed in earlier reporting (see [Malpedia](#) and [MITRE](#)). Notably, this is the first time we've seen ToneShell delivered through a **kernel-mode loader**, giving it protection from user-mode monitoring and benefiting from the rootkit capabilities of the driver that hides its activity from security tools.

Earlier ToneShell variants generated a 16-byte GUID using `CoCreateGuid` and stored it as a host identifier. In contrast, this version checks for a file named `C:\ProgramData\MicrosoftOneDrive.tlb`, validating a 4-byte marker inside it. If the file is absent or the marker is invalid, the backdoor derives a new pseudo-random 4-byte identifier using system-specific values (computer name, tick count, and PRNG), then creates the file and writes the marker. This becomes the unique ID for the infected host.

The samples we have analyzed contact two command-and-control servers:

- **avocadomechanism[.]com**
- **potherbreference[.]com**

ToneShell communicates with its C2 over raw TCP on port 443 while disguising traffic using **fake TLS headers**. This version imitates the first bytes of a TLS 1.3 record (`0x17 0x03 0x04`) instead of the TLS 1.2 pattern used previously. After this three-byte marker, each packet contains a size field and an encrypted payload.

Packet layout:

- **Header (3 bytes):** Fake TLS marker
- **Size (2 bytes):** Payload length
- **Payload:** Encrypted with a rolling XOR key

The backdoor supports a set of remote operations, including file upload/download, remote shell functionality, and session control. The command set includes:

Command ID	Description
0x1	Create temporary file for incoming data
0x2 / 0x3	Download file
0x4	Cancel download
0x7	Establish remote shell via pipe
0x8	Receive operator command
0x9	Terminate shell
0xA / 0xB	Upload file
0xC	Cancel upload
0xD	Close connection

Conclusion

We assess with high confidence that the activity described in this report is linked to the **HoneyMyte** threat actor. This conclusion is supported by the use of the **ToneShell** backdoor as the final-stage payload, as well as the presence of additional tools long associated with HoneyMyte – such as **PlugX**, and the **ToneDisk** USB worm – on the impacted systems.

HoneyMyte's 2025 operations show a noticeable evolution toward using **kernel-mode injectors** to deploy ToneShell, improving both stealth and resilience. In this campaign, we observed a new ToneShell variant delivered through a kernel-mode driver that carries and injects the backdoor directly from its embedded payload. To further conceal its activity, the driver first deploys a small user-mode component that handles the final injection step. It also uses multiple obfuscation techniques, callback routines, and notification mechanisms to hide its API usage and track process and registry activity, ultimately strengthening the backdoor's defenses.

Because the shellcode executes entirely in memory, **memory forensics** becomes essential for uncovering and analyzing this intrusion. Detecting the injected shellcode is a key indicator of ToneShell's presence on compromised hosts.

Recommendations

To protect themselves against this threat, organizations should:

- Implement robust network security measures, such as firewalls and intrusion detection systems.
- Use advanced threat detection tools, such as [endpoint detection and response \(EDR\) solutions](#).
- Provide regular security awareness training to employees.
- Conduct regular security audits and vulnerability assessments to identify and remediate potential vulnerabilities.
- Consider implementing [a security information and event management \(SIEM\) system](#) to monitor and analyze security-related data.

By following these recommendations, organizations can reduce their risk of being compromised by the HoneyMyte APT group and other similar threats.

Indicators of Compromise

More indicators of compromise, as well as any updates to these, are available to the customers of our [APT intelligence reporting service](#). If you are interested, please contact intelreports@kaspersky.com.

36f121046192b7cac3e4bec491e8f1b5	AppvVStram_.sys
fe091e41ba6450bcf6a61a2023fe6c83	AppvVStram_.sys
abe44ad128f765c14d895ee1c8bad777	ProjectConfiguration.sys
avocadomechanism[.]com	ToneShell C2
potherreference[.]com	ToneShell C2

Source: <https://securelist.com/honeymyte-kernel-mode-rootkit/118590/>