Golang Beacons and VS Code Tunnels: Tracking a Cobalt Strike Server Leveraging Trusted Infrastructure

Published: 2025-01-07 · Archived: 2026-04-05 18:51:02 UTC

Command and control (C2) infrastructure is vital for communicating with compromised hosts, enabling threat actors to exfiltrate data, move laterally, and maintain access. As defenders strengthen traditional detection methods, adversaries have turned to creative techniques to control implants, often leveraging trusted platforms to evade scrutiny.

In late November, our research team identified a Cobalt Strike server displaying a well-known watermark and a unique TLS certificate, a pivot point shared by 50 other IPs. Shortly thereafter, a **Golang**-compiled beacon tied to the server was uploaded to multiple malware sandbox platforms. Further analysis revealed the beacon's communication using Visual Studio Code dev tunnels-an uncommon tactic increasingly observed among threat actors leveraging trusted infrastructure to evade detection.

Key Findings:

- A **Cobalt Strike server** using a unique TLS certificate and an oft-seen watermark.
- A **Golang-compiled beacon** communicating with the initial server and leveraging Visual Studio Code Tunnels.
- Additional Azure-hosted infrastructure, though its connection to the initial server, remains uncertain due to shared hosting.

In this post, we explore these findings in-depth, offering defenders key indicators to detect and mitigate similar activity in their environments.

Our research began with the identification of a server hosted at `189.1.231[.]190`, located in Hong Kong and operating on the Huawei Cloud network. First observed on November 20th, Hunt scans detected two Cobalt Strike servers on ports 443 and 1001.

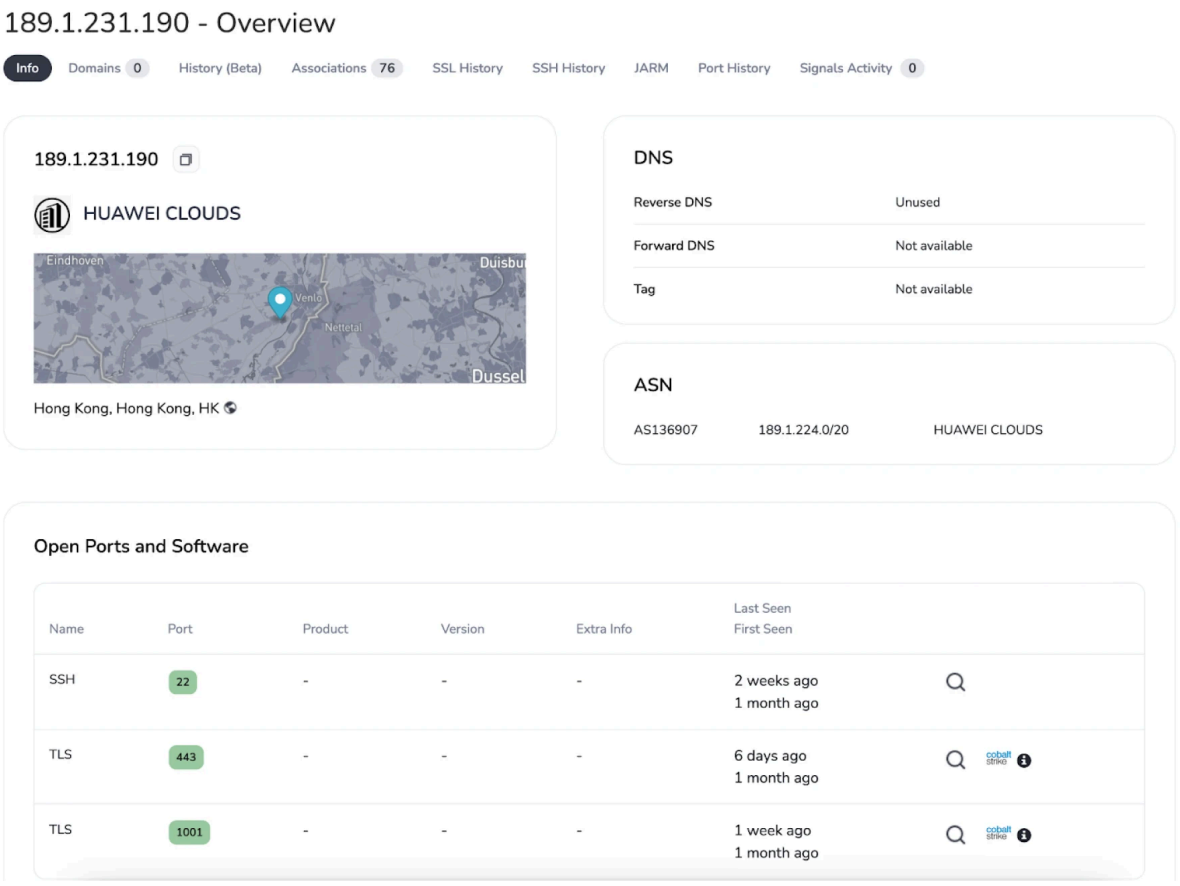

Figure 1: IP overview of the team server that caught our attention (Hunt).

Looking at the beacon configuration for port 443 (readily available by clicking on the "i"), showed that under the endpoints field was a **devtunnels.ms** domain-a key element we will revisit later in this post.

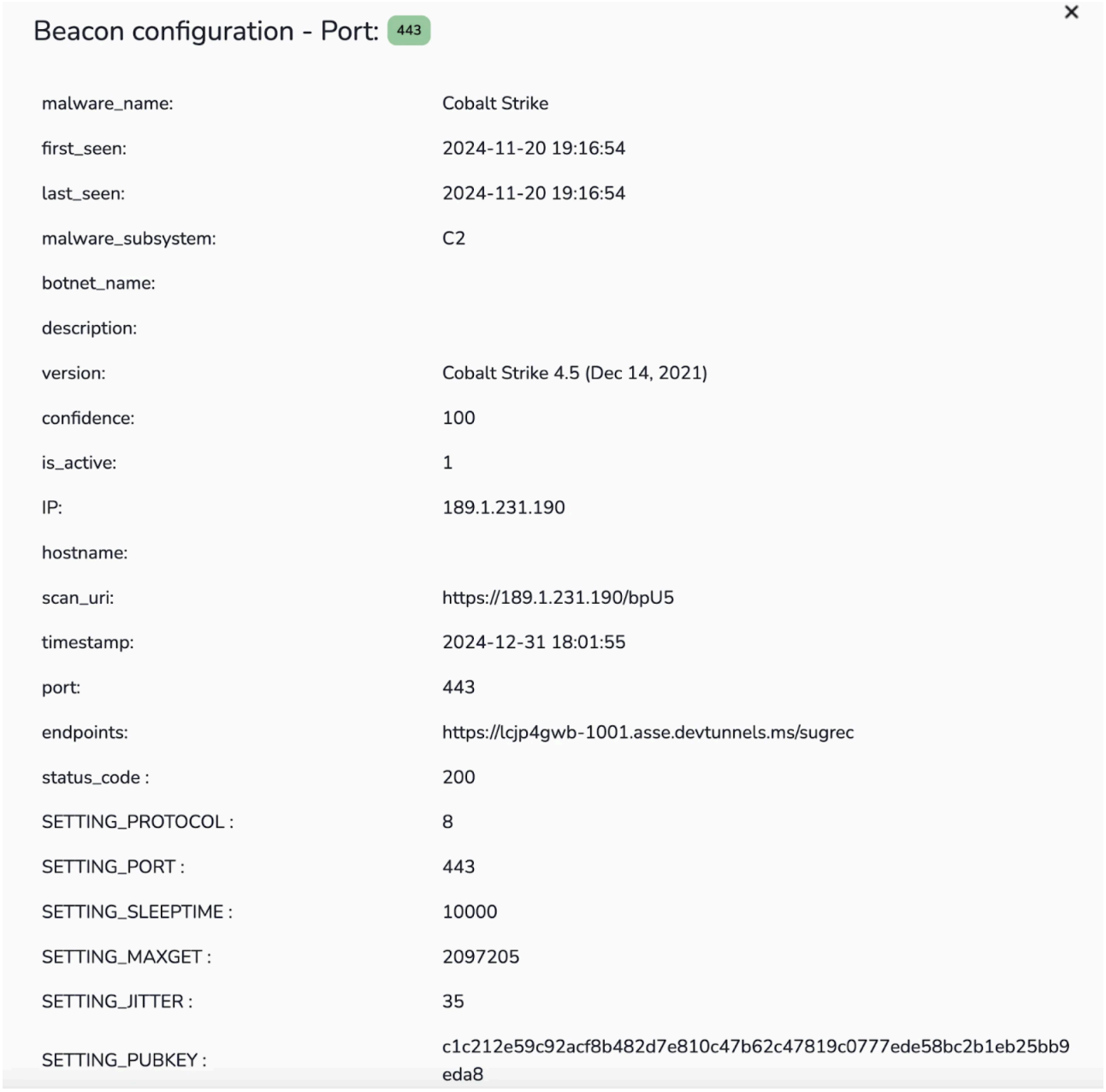

Figure 2: Snippet of beacon configuration.

Port 1001, in turn, displayed a similar configuration, with its **/sugrec endpoint** also being used for C2 communication. Both team servers use the 100000 watermark-a widely recognized identifier within Cobalt Strike deployments. While rare watermarks have previously helped uncover unique threat activity clusters, as discussed in our earlier research, this case stands apart.

As seen in Figure 1, the Associations tab lists **76 additional team servers using the same watermark**. This widespread adoption reduces the likelihood of a targeted or exclusive operation. However, it's possible that threat actors deliberately chose this identifier to blend into the noise, exploiting the assumption that widely used configurations may not attract the same level of scrutiny as rarer indicators.

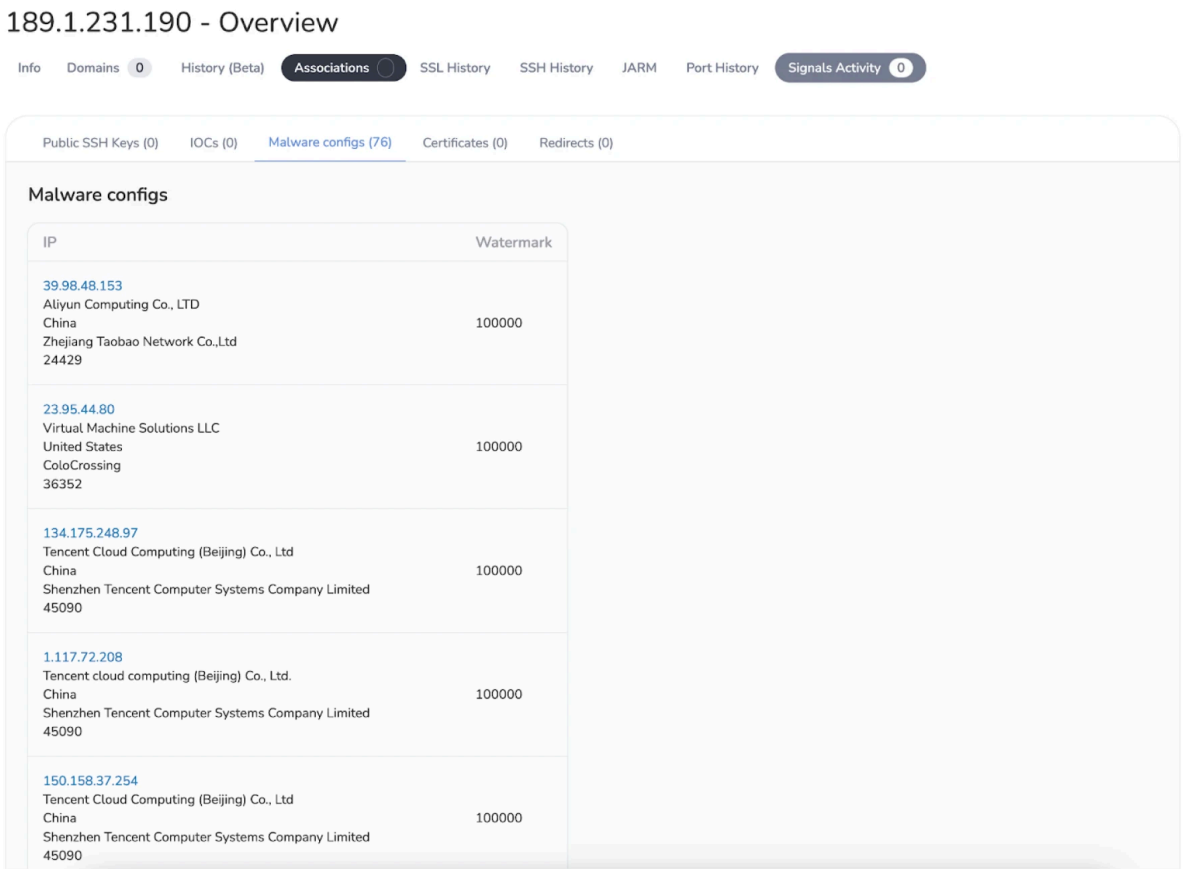

Figure 3: Snippet of associated team servers sharing the 100000 watermark (Hunt).

Additionally, both servers employ a self-signed TLS certificate with the following SHA-256 hash: **EB5AC849E783E3C6EDDCD5619CA230B6D8E218E3C7326E0148C21EEF3847FF69**

The full certificate details are shown below:

- Common Name: US
- Country: CN
- Organization: Software
- Organizational Unit: qq[.]com
- Location: Somewhere
- State: Cyberspace

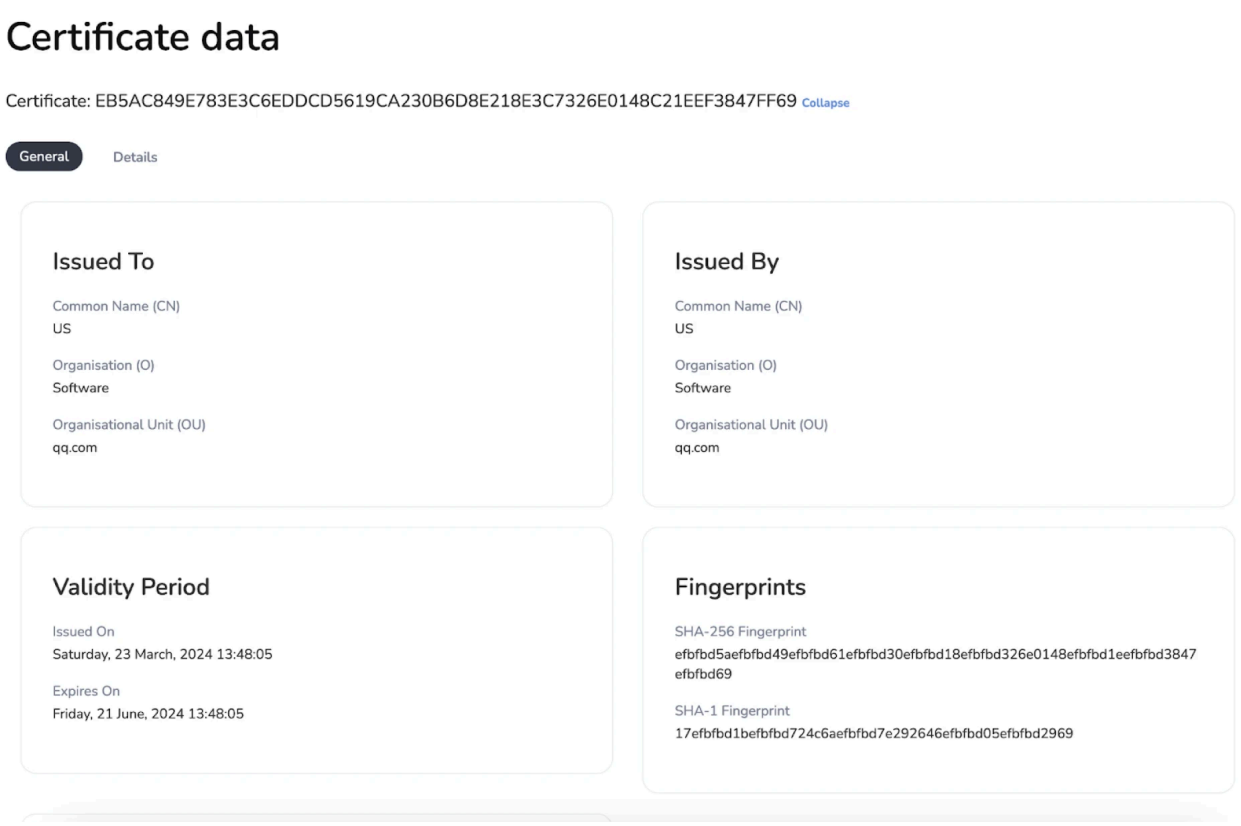

Figure 4: Certificate data for the qq[.] com-themed certificate (Hunt).

Hunt scan data indicates this certificate is currently used by 59 additional IP addresses across the internet.

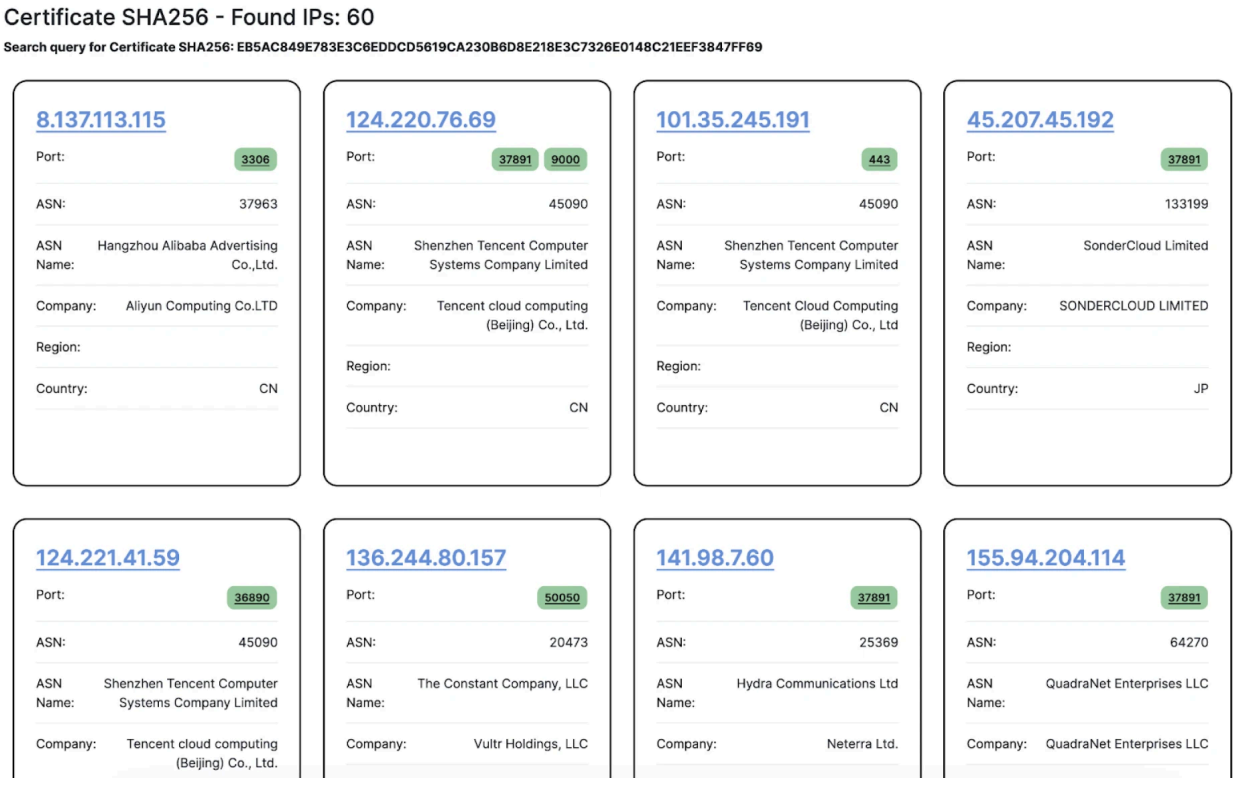

Figure 5: Snippet of the associated certificates (Hunt).

The Golang Beacon: Putting the Pieces Together

The malicious file, `yqWiQTrBWj.exe` (SHA-256: c717d8b26de612e15015cd55940215be336963b6062196f9d847912b98582627), was uploaded to multiple malware sandbox platforms and flagged by several vendors as a Cobalt Strike beacon. For this analysis, we focus on the results obtained from VirusTotal and Hatching Triage.

The Role of Golang in Offensive Operations

Golang has emerged as a favored language in offensive operations due to its **cross-platform compatibility** and ease of use. Tools like **Geacon**, an open-source project that implements Cobalt Strike functionality in Go, have been observed in numerous network intrusions. However, we could not establish a direct link between this sample and Geacon, leaving its exact origins uncertain.

Behavioral Analysis

Despite sparse results from VirusTotal and Triage, we were able to identify some behavioral patterns:

1. **Environment Checks**: Upon execution, the malware calls the GetCommandLine API to determine whether it is running in a virtualized environment.
2. **Host Profiling**: The file collects various system details, including the **operating system version**, **hosts file**, **computer name**, and **machine timezone**. This data is likely exfiltrated to the attacker, who can then decide whether the host warrants further exploitation.

Among the programming-related artifacts in the sample, a PDB path was uncovered:
D:/CS4.5/cs4.5_dabaige_client/script/bypassHR/exe/result/Heapalloc.go

The **CS4.5** reference likely corresponds to **Cobalt Strike version 4.5**, which aligns with the versions observed on the two identified team servers. This version was officially released on **December 14, 2021**.

Network Connections

As part of its network communication, the executable makes an HTTP request to:

`https://189.1.231[.]190:1001/sugrec`

Figure 6: Network communication results in VirusTotal showing a request to the initial team server.

Navigating to this URL in a controlled analysis environment produced a JSON-formatted message:

```
{"err_no":0, "errmsg":"","queryid":"0x417943670752ec"}
```

Copy

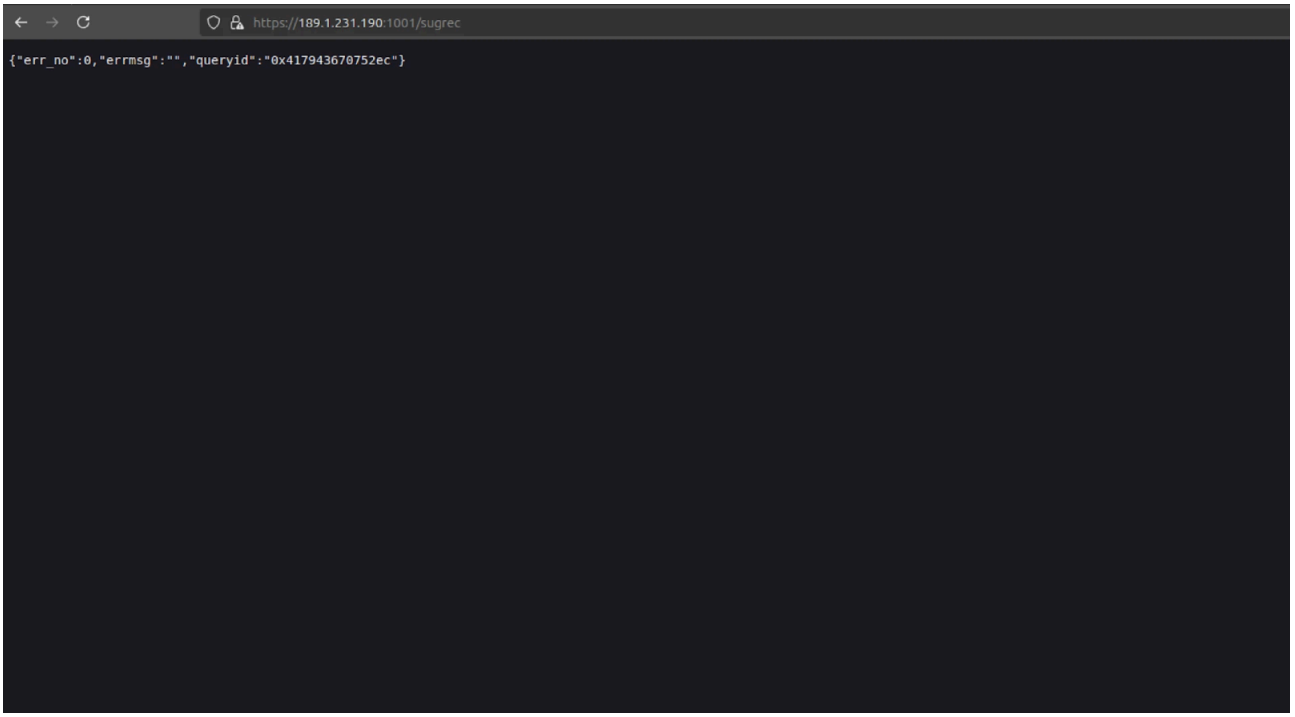

Figure 7: JSON-formatted server response to the /sugrec endpoint.

Another GET request is made to `https://lcjp4gwb-1001.asse[.]devtunnels.ms/_/passApi/js/wrapper.js`.

While the specifics of the above will be covered in the next section, the domain resolves to IP address 20.197.80[.]108, which is hosted on Microsoft's Azure infrastructure in Southeast Asia.

The server responds with a **200 OK** status, a snippet of which can be found below:

Figure 8: Snippet of the HTTP response to the wrapper.js endpoint of the tunnel (Triage).

When accessed directly, the URL redirects to an overlay warning page, resembling typical phishing protection measures. However, embedded within the page's content is a timestamp indicating when the tunnel was created. In our case, the tunnel was established less than **24 hours prior** to the date of writing.

*There are ways to circumvent the interstitial page; however, those methods are best kept for a different post.

This timestamp offers valuable insight into the operational timeline, revealing when the tunnel was established and potentially indicating the campaign's stage within its lifecycle. Such information can help defenders identify active campaigns early, assess their potential scale, and prioritize mitigation efforts.

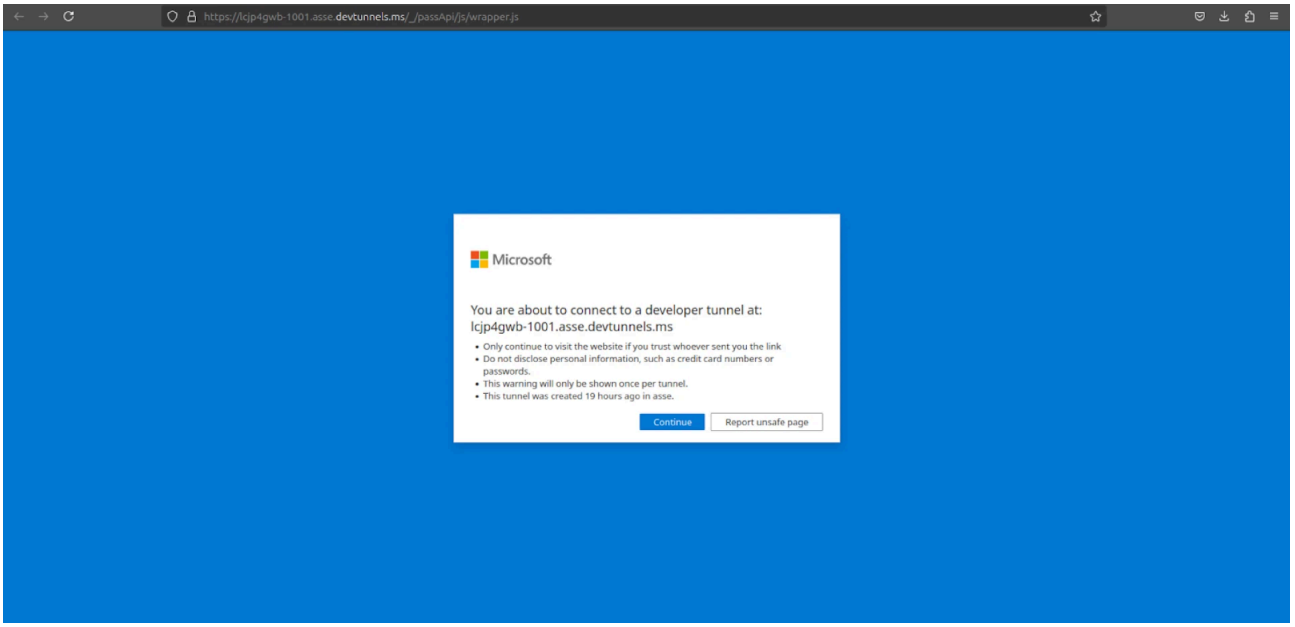

Figure 9: Microsoft overlay page when navigating to the /wrapper.js endpoint.

Tunneling With Visual Studio Code

Initially designed to facilitate secure remote access for developers, Visual Studio (VS) Code tunnels have been repurposed by threat actors as command and control (C2) channels, exploiting their integration with Microsoft Azure infrastructure to blend malicious activity into legitimate traffic.

Recent reporting indicates this tactic is not going anywhere anytime soon. In Operation Digital Eye, SentinelOne attributed the use of VS Code tunnels to purported Chinese APT actors targeting critical infrastructure. Similarly, Unit 42's analysis of the Stately Taurus campaign revealed the weaponization of these tunnels to execute arbitrary commands and deliver additional payloads. Lastly, researchers from Itochu Cyber & Intelligence Inc. presented at JSAC 2024 on a Tropic Trooper campaign that leveraged a RAT in conjunction with VS Code tunnels to infiltrate networks.

Querying the IP address hosting the Visual Studio dev tunnel in Hunt reveals a handful of domains suggesting additional tunnels may be in use. Among these, two domains stand out as different from the others. However, it's important to note that this server is shared among multiple users, and the domains are not necessarily connected to the activity described in this post.

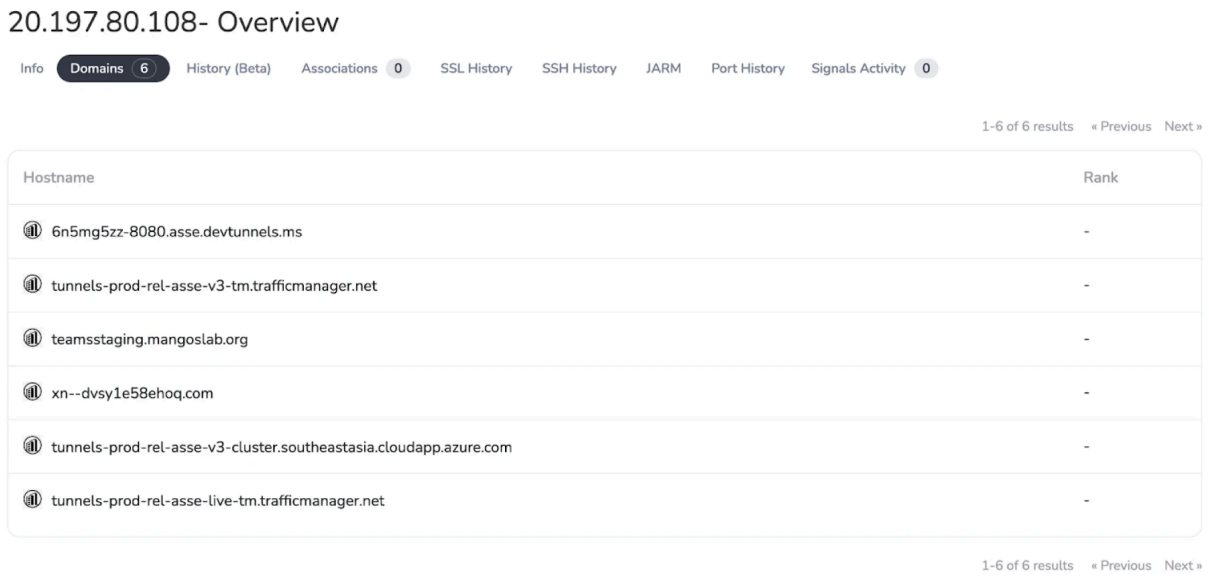

Figure 10: Domains resolving to 20.197.80[.]108 in Hunt. Note the two domains (teamsstaging, *and xn-*)

The SSL history provides further insights, showing a pattern of certificates that can be used for hunting similar infrastructure using the Common Name "**Kubernetes Ingress Controller Fake Certificate**," a detail previously reported by Chris Duggan (@TLP_R3D) on X.

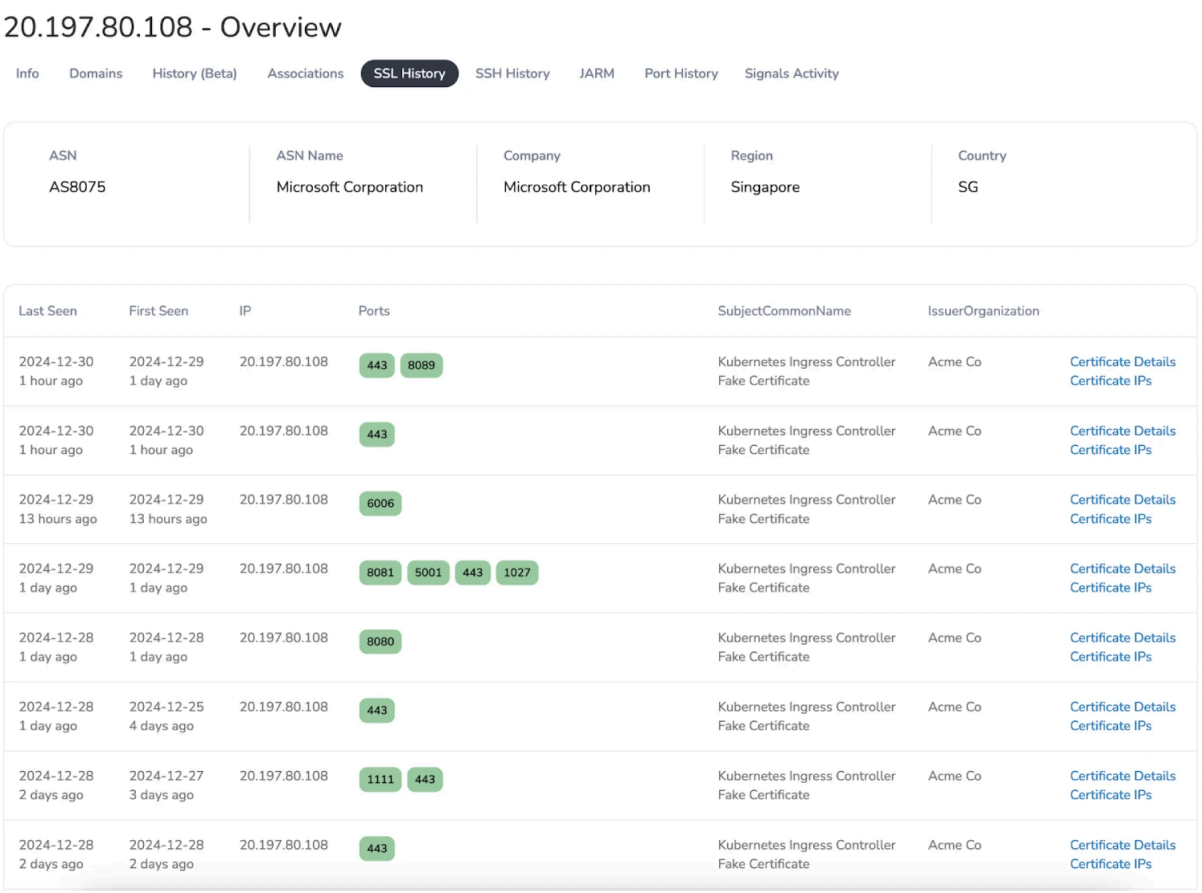

Figure 11: SSL History for the Visual Studio dev tunnel. (Hunt).

Dev Tunnels include an inspect feature, allowing users to analyze tunnel traffic-a capability often used for debugging and connection management. Attempting to access the page for our suspect tunnel at `https://lcjp4gwb-1001-inspect[.]asse.devtunnels.ms` redirects to a Microsoft login page.

This strongly suggests that the actor(s) used a Microsoft account (of which we do not have information on) to create and manage the tunnel. Below is a partially redacted example of the redirect response.

```
https://login.microsoftonline[.]com/common/oauth2/v2.0/authorize?client_id=[REDACTED]&redirect_uri=https%3A%2F%2Fglobal.rel.
```

Copy

While we did not uncover evidence of a Visual Studio Code executable embedded directly within the beacon or elsewhere on the attacker's server, the presence of these artifacts strongly reinforces the use of the software's tunnels as part of the attack infrastructure.

Currently, there is no indication of a particular target associated with the malicious beacon. However, we will continue to monitor for any changes or developments that may provide further clarity on the operation's intent.

Conclusion

The activity identified in this post highlights how threat actors are leveraging trusted platforms like Visual Studio Code tunnels to obscure malicious activity. From a Cobalt Strike server with a widely seen watermark and shared TLS certificates to a Golang-compiled beacon, our findings illustrate the lengths adversaries will go to in order to "blend in" and accomplish their goals.

As we continue to investigate the abuse of these remote tunnels, security teams can take an initial step by monitoring traffic to known dev tunnel domains. Additionally, scrutinizing or restricting their use in environments where Visual Studio Code is not operational can help reduce the risk of undetected activity. Proactive measures like these offer an opportunity to improve visibility into adversarial tactics leveraging trusted infrastructure.
