

Hunting Lazarus Part VII: The Server That Was Not Just FTP

By Threat Research Team

Published: 2026-05-06 · Archived: 2026-05-12 02:01:20 UTC

This is Part VII of the "Hunting Lazarus" series. The [earlier installments](#) documented the Contagious Interview campaign run by Lazarus Group – the DPRK-linked APT – which targets cryptocurrency and Web3 developers with fake job interviews, fabricated company identities, and malicious code repositories. [Part VI](#) described an operation that ran its own pipeline across its own machines and harvested its own people in the process. This installment names one of the machines that pipeline kept using.

The FTP exfiltration sink was port 21. The rest of the host told a larger story.

Executive Summary

Red Asgard's investigation of the Hetzner host at `195.201.104.53` found that the server was not only an FTP exfiltration sink.

Key findings:

- Port 21 ran FileZilla Server 1.12.1 with TLS session resumption enforced.
- At the observation window described in Part V, the FTP tree contained seventy victim folders organized under team-numbered prefixes.
- The same host exposed six Express.js services on non-standard ports: 6101, 6931, 6109, 6939, 6936, and 80.
- Port 6931 was a live OtterCookie Socket.IO C2 broadcasting macOS victim state every thirty seconds.
- Port 6101 was a silent OtterCookie predecessor still accepting Socket.IO connections.
- Port 80 leaked a Windows development path from a Linux deployment:
`D:\server\ss\client\build\index.html` .
- The host demonstrated co-residence: BeaverTail FTP exfiltration and OtterCookie C2 infrastructure running on the same server.

The finding changes the model from "one campaign per host" to "one host as a shared campaign substrate."

The Linux Server Looking for a Windows Folder

A Linux server should not have been looking for a Windows build directory.

But on the Hetzner host at 195.201.104.53, port 80 returned the same error on every request:

```
ENOENT: no such file or directory, stat 'D:\server\ss\client\build\index.html'
```

The server was not broken in an interesting way. It was broken in an evidentiary way.

The path belonged to a Windows development environment. The process was running on Linux. The frontend bundle had not been built before deployment. Someone had shipped a production service with its development assumptions still exposed.

Port 21 on the same host was already known – the FTP exfiltration sink. But port 80 was the clue that the FTP server was not alone.

The host was not just storing stolen data.

It was running campaigns.

Port Map

Port	Service	Observed behavior	Assessment
21	FileZilla FTP	TLS required; session resumption enforced; victim folders present	BeaverTail FTP exfiltration sink
80	Node.js / Express	Repeated ENOENT for D:\server\ss\client\build\index.html	Misdeployed frontend / Windows dev-path leak
6101	Express / Socket.IO	Handshake completed; no victim roster broadcast	Silent OtterCookie predecessor / reserve C2
6109	Express	Accepted connection; Node.js HTTP behavior	Unresolved operational service
6931	Express / Socket.IO	Broadcast victim list every thirty seconds	Live OtterCookie C2
6936	Express	Accepted connection; Node.js HTTP behavior	Unresolved operational service
6939	Express	Accepted connection; Node.js HTTP behavior	Unresolved operational service

Port 21 – The Server We Already Had

The Hetzner node at 195.201.104.53 – one of the three FTP servers that surfaced near the close of [Part V](#) – entered the investigation early. Port 21 was the reason.

The configuration was specific enough to identify it as operator infrastructure rather than a generic FTP host. The service was a FileZilla 1.12.1 server. TLS was required for any data channel. Session resumption was enforced – which meant a standard FTP client library, written to spec but without TLS session reuse implemented, would fail at the handshake. A casual scanner saw a closed door. An operator's tooling, configured for the same handshake, did not.

That configuration is not accidental. It is a discipline choice. It says the server's owner had decided to discriminate between fly-by traffic and tooling that had been written, or modified, with knowledge of the handshake the server expected. The investigation had to match that handshake before it could read what the server held.

What the server held, at the observation window described in Part V, was a directory tree of seventy victim folders, organized under team-numbered prefixes. Earlier waves of the investigation had already mapped those teams – 10team , 2team , and others – to specific operators administering specific subsets of the BeaverTail credential-harvest pipeline. Each victim folder contained a structured dump: browser-stored credentials, session cookies, saved form data, filesystem-adjacent secrets. The folder structure was the operator's working desk.

That description covered port 21. It was complete on its own terms. An entire prior part of this series sat behind the contents of that directory tree.

But a host has more than one port.

The Other Ports

A scan of the same address across non-standard ports produced a different picture entirely.

Six Express.js instances were running. Ports 6101, 6931, 6109, 6939, 6936, and 80. Each presented a Node.js HTTP signature on connection. None of them were the FTP server. None of them were administrative leftovers from a default install.

The Express.js process is not a hosting-provider artifact. It does not appear because Hetzner installed it. It appears because someone wrote a service, picked a port, and started it. Six of them implies six decisions, made deliberately, on the same host that was already running the FTP exfiltration sink for a different campaign.

Two of those six Express services mapped to OtterCookie command-and-control infrastructure: one live node broadcasting active victim state, and one silent predecessor still listening.

That is the finding this article is built around. The Hetzner host was not a single-purpose server. It was a shared substrate – one machine carrying multiple campaigns, multiple malware families, multiple operational lanes, all sharing the same host, the same iron, and the same network egress.

The remainder of this article walks through each of the operational ports in turn, then returns to what the geometry as a whole means for the operation, for the investigation, and for any party considering what to do about either.

Why Co-Residence Matters

Observation	Why it matters
FTP exfiltration and OtterCookie C2 ran on the same host	Separate malware families shared infrastructure
Port 6931 broadcast live macOS victim state	The host was operational during observation
Port 6101 still accepted Socket.IO connections	Old campaign infrastructure was not fully removed
Port 80 leaked a Windows build path	Deployment hygiene exposed development context
Multiple unresolved Express services remained open	The host carried more operational surface than the known FTP sink

Port 6931 – The Live Node

Port 6931 broadcast a victim list over Socket.IO every thirty seconds.

This is not how an administrative panel behaves. Administrative panels respond to requests. Operational C2 nodes broadcast – they push state outward to whatever is listening, on a clock, regardless of whether anyone is asking. The investigation's scanner connected to port 6931 once, held the connection open, and watched the server narrate its own roster every thirty seconds for as long as the connection stayed up.

Five machines were connected during the observation window. All five were running macOS, version 14 or 15. Their campaign numbers – 902 through 906 – matched the OtterCookie naming scheme observed in earlier waves of the investigation, on a different but related node.

OtterCookie does not target Windows. The five machines on port 6931 had been reached through a delivery mechanism distinct from the BeaverTail job-interview chain that produced the FTP victim directories one port over. Same host. Different malware family. Different victim population.

OtterCookie itself is the subject of a later part of this series. For the purposes of this one, it is enough to establish three things about it. It is a JavaScript implant. It targets macOS. It uses Socket.IO as its command-and-control protocol, with capabilities that include screen capture, clipboard contents, and keystroke logging – the tooling profile of an implant designed to collect what a working developer's machine produces in real time, rather than what is already stored at rest.

The placement mattered. The FTP server one port over collects what a victim's machine has already saved. OtterCookie collects what a victim is doing right now. Two different theft windows, served from one host, against two different victim populations, through two different malware families, on two different ports.

The five victims under campaigns 902 through 906 are not named in this article. Notification work is ongoing. The structural finding does not require their names; the campaign numbers are sufficient. What the record establishes is that live victims were checking in to a Socket.IO server on the same machine that was already known as an FTP exfiltration sink for a separate campaign.

That is the primary finding for this part. Everything that follows builds on it.

The OtterCookie C2 layer also extends beyond this single host. The investigation has documented additional OtterCookie nodes on separate infrastructure, and a financial aggregation layer downstream that is fed by OtterCookie data alongside data from other malware families in this operation. Those extensions are the subject of the next article in this series. For Part VII, the relevant fact is local: OtterCookie command-and-control infrastructure was present on this Hetzner host, including one live node broadcasting macOS victim state and one silent predecessor still listening, on the day the FTP server one port over was still receiving uploads from the BeaverTail pipeline.

Port 6101 – The Silence

Port 6101 was another Express.js instance.

The handshake completed cleanly. The Socket.IO server accepted the connection, held it open, and broadcast nothing. No victim list. No keepalive. No campaign identifier. The connection stayed up. Nothing came through it.

A silent C2 is not the same as a closed C2.

A closed C2 is dead. The process is gone. The port is refused. The host has moved on.

A silent C2 is alive. The process is running. The port is open. The server is waiting for instructions, or for a new campaign generation, or for a victim cohort that has not yet been built.

Port 6101 had a history. Earlier in the investigation, on a wave the working notes labelled W62, the same port had been broadcasting. The campaign numbers it carried then were 23 and 77. Those campaigns concluded. The port did not. The process kept running, the listener kept its handle, and the campaign role moved on to a new port – 6931 – with new campaign numbers in the nine-hundreds.

What port 6101 demonstrated was not abandonment. It was reserve capacity.

The pattern is the kind of thing one expects from infrastructure that has been built to last. New campaigns are not deployed on freshly provisioned servers each time. They are routed onto existing hosts that are already provisioned, already whitelisted, already trusted by whatever upstream tooling needs to reach them. The port is recycled. The process stays. The campaign on it changes.

A defender reading this host on the day port 6101 fell silent would have logged a reduction in activity – one C2 quiet, one campaign apparently over – and moved on. The defender would have been wrong about what the silence meant. The silence was not the end of a campaign. It was reserve capacity, with the campaign role already running on a different port.

The remaining three Express ports – 6109, 6939, and 6936 – sat between the live one and the silent one in shape. Each accepted connections. Each completed a Node.js HTTP handshake. None broadcast a victim roster on the cadence that identified port 6931 as an active C2, and none had a documented broadcasting history of the kind that identified port 6101 as a former one. Whether they were under-instrumented control planes, partially deployed services, or campaign reserves, the investigation does not pin down. What the count establishes is simpler: the

host was not running one C2 process and several stubs. It was running an operational fleet of Node services, of which the two on 6931 and 6101 were the two whose role was visible from the outside.

The Numbering Gap

The jump from campaigns 23 and 77 to campaigns 902 through 906 is the kind of detail that asks to be over-read.

The temptation is to claim that campaigns 78 through 901 must have existed in between – that the campaign counter is a monotonic record of every campaign the operation has ever run, that the gap is a population the investigation never saw.

The technical record does not support that claim, and this article does not make it.

What the record supports is narrower. The numbering scheme is not a per-incident label. It is not a one-off ticket assigned to a single phishing run. It is a system with persistence – a counter that accumulates across campaigns, increments through generations, and was high enough by the observation window to imply a much longer operational lifecycle than any single wave's snapshot shows.

How many campaigns ran in the gap, the investigation does not say. That the operation was running a numbering system designed to absorb a long history of campaigns – that, the observed numbers establish.

The distinction matters because it shapes the right reading of the snapshot. The five live macOS machines on port 6931 are not the operation's lifetime victim count for this malware family. They are the population that happened to be checking in during the window the investigation was watching. Whatever the counter represents, the visible machines on port 6931 are only a snapshot, not the operation's lifetime count for this malware family.

The Windows Path

The port-80 error message that opens this article is not unique to this host.

The same pattern – a Windows development environment, a Linux production deployment, a retained development path leaking through error output – has appeared elsewhere in the investigation's corpus. Part VI documented build artifacts from a separate piece of operator tooling that exhibited the same shape.

The temptation, again, is to read across.

The temptation, again, is one this article does not indulge.

A Windows-to-Linux build artifact is not, on its own, an identity. It is a discipline finding. It says that whoever wrote the service developed it on a Windows machine, deployed it to a Linux host, and did not strip the development path from the error response that runs every time the server hits an unhandled route. That is a portrait of build hygiene, not of a person, and not of an attribution chain to any other artifact that happens to share the same hygiene gap.

What two independent appearances of the same pattern in the corpus do support is a structural observation. The operation does not have a unified build pipeline. Different services, different operators, different machines, the same hygiene gap. It is the kind of inconsistency one would expect from a distributed development team rather than a single shop with a packaging standard. The leakage is not a fingerprint. It is a culture.

That inference goes as far as it goes. The two artifacts are not linked beyond that, and this part does not link them further.

Co-Residence

The full picture of 195.201.104.53 fits in one sentence:

A FileZilla FTP exfiltration server on port 21, six Express.js services on the non-standard ports, two of which mapped to OtterCookie command-and-control infrastructure – one live node broadcasting macOS victim rolls over Socket.IO every thirty seconds, one silent predecessor still listening on its own port – and a Node.js HTTP service on port 80 emitting a Windows development path on every request – all on the same host, all operational, all behind the same public address.

Read separately, those services represent at least three operational lanes.

The BeaverTail credential-harvest pipeline used FTP. That was the first lane. The OtterCookie campaign used Socket.IO over Express. That was the second. The development path on port 80 belonged to neither pipeline directly. Its origin remains unattached to a specific malware family, but its provenance was a Windows machine that someone in the operation used. That was the third lane – or the partial trace of one.

Read together, they describe a substrate. The campaigns are not built each on their own machines. They share the iron. They share the egress. They share the exposure surface. A host that takes a single takedown notice does not take down one campaign – it takes down a stack of them, each operating on a separate port, each indifferent to the existence of the others until the moment the host goes dark.

The operation that earlier parts of this series have described as a factory was not running its lines on isolated workshops. It was running them on a shared floor.

The choice has costs and benefits. The benefit is operational efficiency. One host, one exposed surface, one egress point, one address that upstream logs and defenders could pivot around. The cost is precisely the visibility this article describes. A scanner that reads only port 21 sees an FTP server. A scanner that reads the rest of the host sees a campaign substrate. The same property that makes the host efficient to operate makes it efficient to read.

Detection Opportunities

Defenders should treat a host exposing this combination as suspicious:

```
21/tcp  FTP with TLS/session-resumption behavior
80/tcp  Node.js/Express error surface
```

6101/tcp Socket.IO or Express listener
 6109/tcp Express listener
 6931/tcp Socket.IO broadcast behavior
 6936/tcp Express listener
 6939/tcp Express listener

Useful pivots:

Signal	Defensive use
Socket.IO broadcast of victim roster	Identify active C2 behavior, not just open ports
Repeated <code>ENOENT</code> exposing Windows paths on Linux	Hunt misdeployed Node/Express operator tooling
Multiple high-numbered Express services on one VPS	Flag multi-campaign substrate behavior
FTP plus Socket.IO on same host	Correlate stored-data theft with live-session theft

What This Changed

For the investigation, the Hetzner finding rewrote the model.

Before, the corpus had been read campaign by campaign. The BeaverTail pipeline had its own architecture. OtterCookie had its own architecture. The fake-exchange platforms documented in earlier parts had their own architecture. Each sub-investigation produced a writeup that described its target as if it occupied its own dedicated infrastructure, because the single-server view that produced each writeup never escaped the assumption that one campaign belonged to one host.

The Hetzner host violated that assumption. So did other hosts that came under the same lens once the investigation knew what to look for. The same address sometimes hosted a credential exfiltration sink, an active C2 node, a silent predecessor C2 still listening on its own port, and a partially deployed service whose error responses identified a Windows build environment.

The substrate was the unit. The campaigns were the load.

That distinction matters for what an enforcement action against any one campaign actually achieves. A takedown directed at OtterCookie on this host would have removed two of the six Express services – the live broadcasting C2 and the silent predecessor still listening – and left the FTP server untouched, the three unresolved Express services untouched, and the development-path service on port 80 still answering. A takedown directed at the FTP exfiltration sink would have closed port 21 and left the OtterCookie C2 broadcasting on port 6931, the predecessor port 6101 still listening, the three unresolved Express services still up, and the development-path leak still answering on port 80.

Single-campaign takedowns produced single-campaign holes in a multi-campaign machine.

The reverse was also true. A successful seizure of the host would not be a takedown of OtterCookie or of BeaverTail or of the unbuilt service on port 80 – it would be a seizure of all of them at once, regardless of whether the seizure order had named all of them or only one. An action scoped to one campaign would either under-deliver or over-deliver against the substrate it actually reached. There was no way for it to land cleanly on a single lane.

The shape of the operation determined the shape of any meaningful intervention. Both sides of the contest had to read the host before they read the campaign.

What This Article Does Not Claim

- It does not claim campaigns 78 through 901 existed.
- It does not identify the unnamed macOS victims.
- It does not attribute the Windows path leak to the same builder described in Part VI.
- It does not claim every unresolved Express port was a live C2.
- It does not publish access mechanics, credentials, or victim data.

Close

Part VI described an operation that ran its own pipeline across its own machines and harvested its own people in the process. The factory ate its workers because the factory did not distinguish, in any technical sense, between a worker and a target.

Part VII names one of the machines the factory kept using.

195.201.104.53 was not exhausted. Its FTP service was still serving victim directories at the close of the observation window. Port 6931 was still broadcasting macOS check-ins every thirty seconds. Port 6101 was still listening for whatever campaign would inherit it next. Port 80 was still answering with a Windows path the developer had never cleaned up.

The host was not exposed by an enforcement action. It was exposed by the same property that made it useful to the operation – the willingness to run multiple campaigns on the same iron, to open multiple ports for separate services, to share a substrate across operational lanes that did not otherwise know about each other.

A separate investigation, working a different cluster of infrastructure, has documented its own version of this geometry. Future parts will return to that comparison. For now, the finding for this part is local: one host, multiple campaigns, multiple malware families, one shared listening surface.

The investigation continues.

The malware the machine was serving turned out to matter more than the machine itself.

Share this article

Help spread the word about security best practices.

Source: <https://redasgard.com/blog/hunting-lazarus-part7-server-not-just-ftp>