

Threat Intelligence Report: ZionSiphon OT Malware First Attempts? Psyops? Both?

 dti.domaintools.com/research/threat-intelligence-report-zionsiphon

DomainTools

May 21, 2026



Executive Summary

ZionSiphon is a malware sample ([“SCADA_SecurityPatch_v8.4.exe”](#)) that has been circulating in public sandboxes since 2025. It is best understood as a Windows-based implant with explicit industrial control system (ICS) targeting intent but with a critical limitation in its verification of geographic data that fundamentally constrains its operational viability. While earlier analysis established the malware as functionally capable at the host level, subsequent findings confirm the presence of a critical XOR bug in its geographic validation logic, preventing the payload from activating in its intended environment. *Additionally, there is no evidence of vendor-specific protocol handling, no confirmed register mapping, and no interaction with PLC firmware or engineering toolchains. The malware appears to rely on file-based or high-level configuration manipulation, which may not translate into actual process changes in most industrial environments.*

The malware’s architecture remains coherent and deliberate. It combines geographic scoping, environment-aware execution, and embedded process manipulation logic, demonstrating a structured conceptual model of water treatment and desalination systems. Its internal string corpus provides high-confidence evidence of targeting, including references to Mekorot and major desalination facilities such as Sorek, Hadera, Ashdod, Palmachim, Shafdan, and Eilat water plants in Israel, alongside a dense vocabulary covering reverse osmosis, chlorine dosing, and salinity control. Filesystem-based validation and vendor-associated paths further reinforce that the malware is engineered to identify and operate within specific industrial environments.



However, the XOR validation flaw introduces a decisive constraint. The malware is designed to restrict execution to Israeli ("IL") network ranges and to self-destruct if those conditions are not met. Due to the encoding error, this validation check never evaluates as true, meaning the malware fails to recognize its target environment even when present. As a result, the payload does not progress to its process manipulation stage and instead frequently triggers its own cleanup routines. This explains previously observed inconsistent or absent execution

behavior: rather than reflecting conditional activation alone, it represents a systemic failure in the activation pathway.

The intended sabotage model remains conceptually clear despite this failure. ZionSiphon includes embedded parameters designed to manipulate critical control points, such as increasing chlorine dosing and altering reverse osmosis pressure, and contains references to industrial protocols including Modbus, DNP3, and S7comm. These elements demonstrate an understanding of how disruption could be achieved within water treatment systems. However, these attempts are just that. The malware does not contain actual ICS code that would attempt to effect those changes. What it does do, is attempt to do so through the manipulation of the OT layer (e.g. the Windows machines that the malware is detonated on to change those levels in the front end) However, because the activation condition cannot be satisfied, this logic remains dormant and unexecuted, reinforcing that the malware is not currently capable of delivering physical-world impact.

This reframes the malware's maturity. ZionSiphon operates entirely at the Windows host layer, using registry persistence, PowerShell-based execution, and USB-oriented propagation logic. It is a real, functioning implant in terms of execution mechanics, but the XOR bug prevents it from transitioning into an active sabotage phase, rendering it effectively non-operational as an ICS attack tool.

Compounding this limitation is the absence of any meaningful communication stack or command-and-control (C2) channel. The malware assessed (SCADA_SecurityPatch_v8.4.exe)

does not maintain operator connectivity, does not receive tasking, and cannot adapt its behavior dynamically once deployed. This removes the possibility of controlled, iterative interaction with target systems, an essential component of any serious ICS attack capability.

As a result, even in a hypothetical scenario where the activation flaw did not exist, the operational model would still be extremely constrained. The malware behaves more like a single-shot, pre-scripted payload than a coordinated intrusion tool. In practical terms, this resembles a "drive-by" action with no ability to correct aim, adjust targeting, or respond to environmental feedback and in this case, executed with insufficient precision to achieve its intended effect.

In its present form, ZionSiphon is therefore more accurately categorized as a *prototype, misconfigured payload, or intentionally constrained artifact, rather than a deployable cyber-physical weapon*. Its structure demonstrates intent and conceptual targeting, but lacks the control, reliability, and feedback mechanisms required for sustained or meaningful impact on water infrastructure systems.

An additional dimension now strengthens this interpretation: the nature of the code itself. The malware exhibits a pattern of semantically rich but technically shallow ICS logic, where process terminology and targeting concepts are convincing, but underlying implementation depth is limited. This includes incomplete protocol handling, an absence of PLC-resident execution, and a lack of deterministic control paths. Combined with the presence of a critical logic error in a core validation function, this suggests a development process that may have incorporated partial automation or assisted code generation, rather than rigorous engineering validation. While not determinative, the structure is consistent with a scenario in which elements of the code, particularly naming conventions, scaffolding, or ICS-related logic may have been *augmented* through [LLM-style assistance](#), while overall assembly and operational framing remain human-directed.

The presence of explicit ideological messaging embedded within the binary further complicates interpretation. Decoded content referencing attacks on Israeli population centers introduces a clear narrative and psychological layer that is independent of technical execution. When combined with a payload that cannot activate and a second stage that cannot execute past the Windows host, this raises the possibility that the malware functions, at least in part, as a PSYOP-adjacent artifact, where the objective is to project capability, signal intent, and shape perception rather than achieve immediate operational effect.

From a capability perspective, ZionSiphon still reflects an early-stage attempt to approximate a Stuxnet-like attack model, leveraging a Windows foothold to influence industrial processes. However, the combination of incomplete ICS integration, execution fragility, and the XOR validation failure indicates that it falls well short of a reliable implementation. Whether this reflects immature development, operator error, or deliberate constraint remains unresolved, but it clearly places the malware within a mid-tier or experimental development context.

The most accurate interpretation is therefore layered:

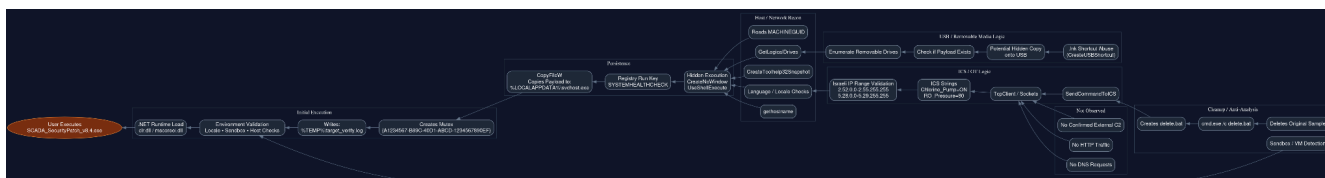
- Technical Layer: A real Windows-based malware implant with coherent targeting logic
- Capability Layer: Non-functional as an ICS weapon due to a critical validation flaw
- Development Layer: Likely a prototype or partially validated build, with possible assisted code generation elements
- Psychological Layer: Potential signaling artifact, where perceived capability exceeds actual execution

ZionSiphon should therefore be understood as a conceptually mature but functionally broken ICS-targeting malware, whose significance lies less in its current operational capability and more in what it reveals about the evolving accessibility, modularity, and perception-driven use of cyber-physical attack tooling.

Malware Analysis: SCADA_SecurityPatch_v8.4.exe

ZionSiphon ([SCADA_SecurityPatch_v8.4.exe](#)), as understood by the malware sample from 2025, is best understood as a Windows-hosted operational malware implant built around explicit OT and water-sector targeting intent rather than a purely conceptual or symbolic artifact. The cumulative evidence gathered through static reverse engineering, sandbox telemetry, recovered string analysis, and vendor reporting materially reduces earlier uncertainty about the malware's purpose and operational model. The sample is a PE32 Mono/.NET executable that relies on the Common Language Runtime (mscorlib.dll) for execution, placing its logic entirely within the Windows host layer rather than within PLC firmware or native controller environments. Architecturally, this positions ZionSiphon as host-based OT intrusion tooling designed to compromise operator or engineering workstations as the pathway toward potential process disruption.

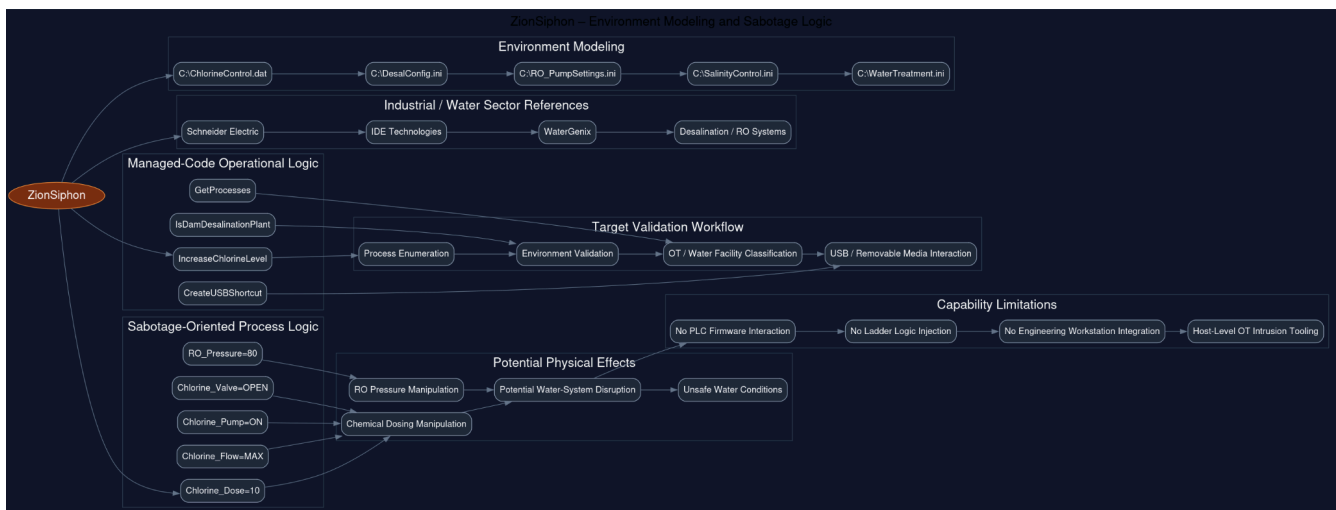
The malware's execution workflow is internally coherent and operationally plausible. Embedded identifiers and execution strings including RunAsAdmin, SystemHealthCheck, Start-Process -Verb RunAs, target_verify.log, and delete.bat map to a structured lifecycle involving privilege escalation, persistence, environment validation, and cleanup. Hybrid Analysis telemetry confirmed that the sample stages itself into %LOCALAPPDATA%\svchost.exe, establishes persistence through the current-user Run registry key under SYSTEMHEALTHCHECK, and invokes cleanup routines through batch execution if execution conditions are not met. The masquerading of the payload as svchost.exe reflects recognizable adversary tradecraft intended to blend into legitimate Windows process naming conventions rather than functioning as a placeholder or incomplete concept.



The strongest evidence of deliberate targeting lies within the malware's environment modeling and industrial process references. The binary contains extensive water-sector-specific configuration names and file paths including C:\ChlorineControl.dat, C:\DesalConfig.ini, C:\RO_PumpSettings.ini, C:\SalinityControl.ini, and C:\WaterTreatment.ini, alongside references to industrial and desalination-associated entities such as Schneider Electric, IDE Technologies, and WaterGenix. These are not generic enterprise strings or superficial theming elements. Instead, they reflect a logically structured representation of desalination and water-treatment operational environments. Additional managed-code identifiers including IncreaseChlorineLevel, IsDamDesalinationPlant, GetProcesses, and CreateUSBShortcut

further demonstrate that the malware was designed to enumerate processes, validate target environments, and interact with removable media in ways consistent with industrial intrusion workflows.

The sabotage-oriented process logic embedded in the sample reinforces this assessment. Strings such as Chlorine_Dose=10, Chlorine_Flow=MAX, Chlorine_Pump=ON, Chlorine_Valve=OPEN, and RO_Pressure=80 define a *conceptual model* for manipulating chemical dosing and reverse-osmosis pressure systems. These parameters correspond to operationally sensitive functions within water-treatment infrastructure and imply an intent to push process variables into potentially disruptive or unsafe states. *While there is no evidence that the malware contains mature PLC-specific payload delivery or ladder-logic manipulation capability, the embedded logic clearly moves beyond espionage-oriented collection tooling and into the domain of intended process interference.*



At the same time, the malware remains technically constrained and immature when compared to fully operational ICS-native malware families such as Stuxnet, TRITON, Industroyer, or IOCONTROL. Although the sample references industrial protocols including Modbus and DNP3, there is still no evidence of complete protocol implementation, vendor engineering software integration, PLC firmware interaction, or safety-system manipulation. Hybrid Analysis telemetry also confirmed the absence of meaningful DNS resolution, outbound HTTP traffic, or operational command-and-control communications during execution. Despite containing networking functionality through .NET TcpClient, socket APIs, and connection-handling routines, the malware did not demonstrate active beaconing or remote tasking behavior. This strongly suggests an autonomous or pre-scripted execution model in which actions are triggered locally through environment validation rather than controlled dynamically through external infrastructure.

The detonated filename itself further reinforces this operational model. The use of naming the file `SCADA_SecurityPatch_v8.4.exe` strongly suggests deliberate masquerading as a

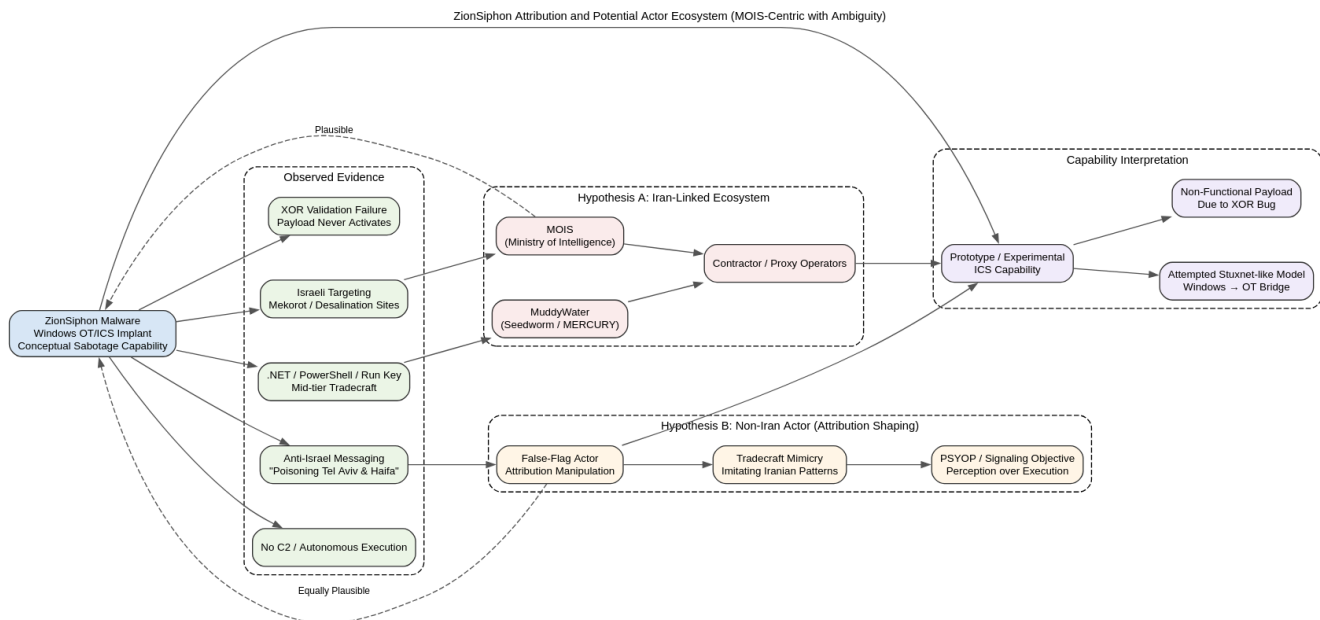
legitimate industrial software update or maintenance utility. Combined with the extensive OT-themed naming conventions and water-sector references embedded throughout the sample, this creates a highly plausible watering-hole or trusted-update delivery scenario. Under such a model, attackers could establish a malicious website or compromised vendor portal advertising a supposed SCADA or water-treatment software patch and direct operators or engineers toward it through spear-phishing or industry-themed communications. Once executed, the malware could establish persistence, validate the target environment, and opportunistically seed removable media for downstream propagation into more sensitive operational enclaves.

Execution gating and cleanup behavior further support the conclusion that the malware was designed for selective deployment rather than indiscriminate execution. The presence of `target_verify.log`, environment-validation logic, Israeli IP-range geofencing, and cleanup mechanisms such as `delete.bat` indicate that the malware evaluates its environment before activating fully and removes artifacts if conditions are not satisfied. This reflects a controlled operational philosophy intended to minimize exposure and reduce forensic visibility outside intended targets.

Overall, ZionSiphon occupies an unusual position within the spectrum of OT malware. It is substantially more operationally coherent than simple propaganda or proof-of-concept malware and demonstrates a complete host-level intrusion lifecycle including privilege escalation, persistence, environment validation, removable-media interaction, and cleanup. At the same time, it lacks the mature ICS-native functionality associated with the most sophisticated cyber-physical malware families. The result is a malware framework that appears operationally credible at the Windows host layer and explicitly aligned toward water-sector disruption, while still remaining developmental, partially constrained, and dependent on contextual execution and human-assisted propagation to achieve meaningful operational impact.

Actor Assessment and Strategic Framing

Attribution remains unconfirmed. The available evidence supports a plausible Iranian nexus, but it does not prove that ZionSiphon was created or deployed by an Iranian state actor, an Iranian proxy, or any specific Iranian APT cluster. The malware's target selection, embedded references to Israeli water infrastructure, and decoded anti-Israel messaging align with operational themes long associated with Iranian cyber activity directed at Israeli civilian infrastructure. Public reporting also places the malware in the context of Israeli water-treatment and desalination targeting.



Within that frame, the best technical fit is still a mid-tier, [MOIS](#)-aligned ecosystem such as [MuddyWater](#) or a related contractor/proxy environment, rather than a top-tier bespoke ICS weapons program. The sample's architecture is a managed *PE32 Mono/.NET* executable that runs through *mscoree.dll*, stages itself as *%LOCALAPPDATA%\svchost.exe*, persists via the *SystemHealthCheck* Run key, elevates through PowerShell using *Start-Process -FilePath ... -Verb RunAs*, and cleans up through *delete.bat* and *target_verify.log*. That pattern is closer to commodity or lightly customized Iranian tradecraft than to a highly specialized controller-native platform. The Falcon Sandbox report also confirms a malicious score, registry persistence, self-deletion behavior, guarded-memory anti-analysis features, Base64 decoding capability, and the absence of relevant DNS, HTTP, or contacted-host infrastructure.

At the same time, the malware *does not exhibit the hallmarks of a mature ICS weapon*. It contains water-sector process logic, industrial vocabulary, and protocol references, but no demonstrated PLC-resident code, no ladder-logic manipulation, no vendor-specific engineering-stack abuse, no validated register maps, and no deterministic command path into real control systems. Public reporting similarly describes it as a targeted OT/ICS malware strain aimed at Israeli water systems, but not as a proven [Stuxnet](#)- or [TRITON](#)-class capability. The most defensible technical reading is that this is a Windows-hosted OT sabotage implant whose ICS layer is still incomplete, experimental, or intentionally simplified. ([Darktrace](#))

The strongest evidence for an Iran-aligned framing comes from the decoded strings. The sample contains extensive water/OT targeting strings tied to Israeli infrastructure and desalination operations, including facility and environment markers such as *Mekorot*, [Sorek](#), [Hadera](#), [Ashdod](#), [Palmachim](#), [Shafdan](#), and [Eilat](#) Desal, alongside control-oriented terms such as *DesalPLC*, *OsmosisPLC*, *WaterPLC*, *ChlorineCtrl*, *ChlorineDose*, *RO_Pump*, and

BrineControl. More importantly, the decoded ideological content includes the explicit line “Poisoning the population of Tel Aviv and Haifa”, and the malware also contains an execution-guardrail message reading “Target not matched. Operation restricted to IL ranges. Self-destruct initiated.” These strings materially strengthen the assessment that the malware is framed as anti-Israel and specifically oriented toward Israeli water infrastructure.

```
Base64 Coded String in malware:
SW4gc3VwcG9ydCBvZiBvdXIgYnJvdGhlcnMgaW4gSXJhbiwgUGFsZXN0aW5LLCBhbmQgWWVtZW4gYWdhaw5zdCBaaW9uaXN0IGFnZ3Jlc3Npb24uIEkgYW0gIjB4SUNTi14=

Decode:
In support of our brothers in Iran, Palestine, and Yemen against Zionist aggression. I am "0xICS".
```

However, those same strings are also the clearest reason not to overstate attribution. Ideological text that points toward Iran and against Israel can support an Iran-aligned hypothesis, but it can also function as attribution theater. An actor seeking to implicate Iran, exaggerate Iranian capability, or exploit existing expectations about Iranian cyber behavior could deliberately embed precisely these kinds of messages. The sample’s combination of overt anti-Israel language, Israeli geofencing, Mekorot branding, and incomplete ICS execution depth is consistent not only with a genuine Iranian capability in development, but also with a scenario in which another actor is muddying the water by constructing a malware artifact that looks Iranian on first inspection. In that sense, the ideological layer is evidentiary, but not dispositive.

That ambiguity is especially important because ZionSiphon also reads as an early attempt at a Stuxnet-like attack path against Israel, but without Stuxnet-like engineering maturity. The malware clearly models cyber-physical effects: it hunts for desalination and treatment artifacts, references Modbus and DNP3, and embeds static sabotage values such as *Chlorine_Dose=10*, *Chlorine_Flow=MAX*, *Chlorine_Pump=ON*, *Chlorine_Valve=OPEN*, and *RO_Pressure=80*. It is trying to move from a Windows foothold on operator or engineering systems into process disruption. That is strategically significant. But the implementation still falls well short of a real, deterministic industrial attack platform, which makes it plausible both as a prototype capability and as a signaling artifact meant to invoke the idea of an Iranian Stuxnet-for-Israel scenario.

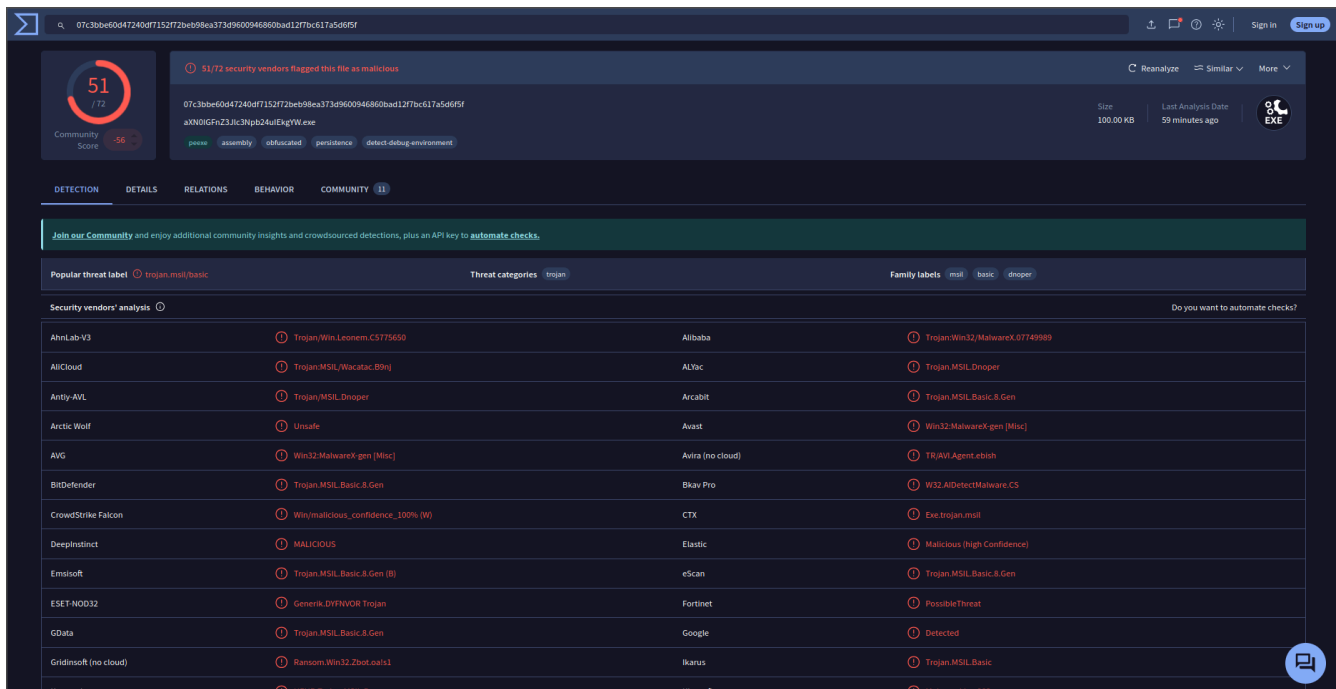
The Falcon Sandbox findings reinforce the dual-use interpretation. The sample is operationally real at the host level: it persists, stages, executes, validates the environment, and self-cleans. But it shows no relevant DNS requests, no relevant HTTP traffic, and no relevant contacted hosts, which means there is no public evidence of a live C2-backed campaign around this sample. That absence supports the view that ZionSiphon is either a self-contained, pre-scripted sabotage implant or a demonstration artifact whose strategic value derives partly from being discovered and analyzed.

The most accurate conclusion is therefore deliberately layered. ZionSiphon may well be consistent with MOIS-linked Iranian operational patterns, and MuddyWater remains the closest tradecraft fit among known Iranian clusters. But there is still no real proof that an Iranian actor built it, and the available data also supports the possibility that another actor intentionally embedded Iran-supporting and anti-Israel text to create exactly that impression. In practical terms, ZionSiphon should be understood as a hybrid artifact: a real host-based malware implant with clear OT sabotage intent, a likely experimental or early-stage attempt to approximate a Stuxnet-like attack path against Israeli infrastructure, and a possible PSYOP or attribution-shaping tool whose ambiguity may itself be part of its operational effect.

Detection Profile and Operational Maturity Assessment

Integration of multi-source analysis including static reverse engineering of the uploaded samples, sandbox telemetry from [ANY.RUN](#) and Hybrid Analysis, and detection data from [VirusTotal](#) provides a consolidated view of ZionSiphon's true position within the threat landscape. The resulting picture is not ambiguous: the malware is operationally real at the host-implant layer, but its ICS disruption capability remains unproven and likely immature in its current form.

From a detection standpoint, VirusTotal confirms that the sample is [broadly recognized as malicious across multiple engines](#). However, the classification is inconsistent and generic, with most vendors labeling the file as a .NET or MSIL-based trojan, loader, or agent. There is no consensus naming, and critically, no engine identifies the sample as ICS malware or associates it with industrial protocol abuse. This absence is not incidental. It indicates that the malware's OT-specific logic is not driving its detection profile. Instead, detection is triggered by conventional behaviors, such as PowerShell-based execution, registry persistence, process masquerading, and general suspicious activity, placing *ZionSiphon* firmly within the *detection envelope of commodity Windows malware*.



This observation aligns directly with the static and dynamic analysis of the binary. Reverse engineering confirms that the sample is a Mono/.NET executable with a minimal import table and all functional logic embedded internally. The malware establishes persistence through a Run key (SystemHealthCheck) pointing to a disguised payload (svchost.exe), relaunched itself with elevated privileges via PowerShell, and implements cleanup routines using delete.bat and target_verify.log. These behaviors are not theoretical; they are consistent across sandbox environments and embedded directly in the binary. The implant layer is therefore fully functional and operationally credible, with no indication of being a placeholder or decoy.

Where the assessment becomes more complex is at the ICS interaction layer. The binary contains extensive water-sector targeting artifacts, including configuration paths, process identifiers, and explicit manipulation strings such as *Chlorine_Dose=10*, *Chlorine_Flow=MAX*, and *RO_Pressure=80*. These elements demonstrate clear intent to interfere with water treatment processes, particularly chemical dosing and pressure regulation. *However, the implementation lacks the depth required for reliable real-world execution.*

Dynamic analysis reinforces this limitation. Execution behavior varies significantly between sandbox runs, with some environments exhibiting full persistence and artifact creation, while others produce minimal activity or even a “no threat detected” verdict. This inconsistency suggests that the malware is highly dependent on environmental conditions, potentially due to validation gating, incomplete code paths, or anti-analysis mechanisms. While this behavior could be interpreted as evasive design, it also introduces uncertainty regarding execution reliability, particularly in non-laboratory conditions.

The broader implication is that ZionSiphon occupies a hybrid position between commodity malware and specialized OT tooling. Its underlying framework is indistinguishable from generic .NET malware, as confirmed by both imphash clustering and VirusTotal classification. Its distinguishing features, the ICS targeting logic and sabotage intent, are layered on top of this framework but are not yet expressed in a technically mature or reliably executable form. This architectural choice provides flexibility and ease of development but limits the malware's ability to achieve consistent physical impact.

From an operational perspective, the malware is highly likely to succeed in compromising Windows-based systems, particularly those associated with engineering or supervisory functions in water-sector environments. It can persist, execute, and perform environment validation with high confidence. It may also disrupt local applications or introduce configuration inconsistencies that affect operator workflows. *However, the probability that it can directly and reliably manipulate physical processes such as chlorine dosing or system pressure—remains low without further development or environment-specific customization.*

This duality is central to understanding ZionSiphon. It is not a non-functional artifact, nor is it a mature ICS weapon. It is a functional host-based implant with embedded, but not yet fully realized, OT disruption logic. Its current form suggests either an early-stage capability under development or a modular framework intended for future enhancement. In either case, the gap between intent and execution is evident.

The most defensible conclusion is that ZionSiphon represents a transitional class of malware, bridging traditional IT compromise and potential OT impact. Its significance lies less in its immediate effectiveness and more in what it signals: that targeted, domain-aware cyber-physical tooling can be constructed using relatively accessible components. While it does not yet demonstrate the precision or reliability of established ICS malware families, it provides a clear indication of direction toward more modular, adaptable, and potentially proliferating OT-focused threats.

Strategic Assessment and Forward Outlook

ZionSiphon is best understood as a targeted ICS sabotage capability in development, combining deliberate, domain-aware targeting logic with a modular and reusable technical foundation. The accumulated evidence of static analysis, sandbox telemetry, and decoded string corpus confirms that the malware encodes a coherent conceptual model of water treatment operations, particularly chlorine dosing and reverse osmosis control. At the same time, it relies on a commodity Windows/.NET implant layer for execution, persistence, privilege escalation, and delivery. This hybrid construction places the malware in a transitional

category: operationally real and credible at the host level, but not yet reliably effective at the control-system level.

The broader strategic context reinforces this interpretation, while also introducing a critical layer of ambiguity. Since 2025, Israeli water infrastructure, especially systems associated with Mekorot, has remained a recurring target in cyber operations and reporting. ZionSiphon fits squarely within that targeting pattern, including its geographic scoping to Israeli networks and its explicit references to desalination facilities and water-treatment processes. However, there is still no publicly confirmed instance of successful cyber-induced physical disruption to these systems in the current reporting cycle. This persistent gap between targeting intensity and observable impact is analytically significant. It highlights both the priority placed on this sector and the continued limitations of adversary capabilities.

ZionSiphon embodies that gap directly. Its embedded parameter manipulation strings, process-specific vocabulary, and environment validation logic clearly demonstrate intent to influence physical processes. Yet its reliance on static configuration assumptions, incomplete industrial protocol handling, absence of validated PLC interaction, and environment-dependent execution behavior indicate that it is not yet a mature or deterministic ICS weapon. In its current form, the malware is more likely to produce host-level compromise, configuration disruption, and operational friction than sustained or precise control over industrial processes.

At the same time, the sample introduces an additional dimension that materially affects its strategic interpretation: the presence of explicit ideological messaging and narrative cues embedded within the binary. These elements are not required for execution and instead serve a signaling function, shaping how the malware is interpreted once discovered. Combined with its Israeli targeting, they create an artifact that is not purely technical. This opens the possibility that ZionSiphon is functioning in part as a PSYOP-adjacent tool, where perception of capability and intent is itself an operational objective. Importantly, while the messaging aligns with Iran-aligned narratives, there is no definitive proof that the malware originates from an Iranian actor. The same elements that support an Iranian attribution hypothesis could also be deliberately constructed by another actor to mimic, exaggerate, or redirect attribution, effectively muddying the waters.

From a capability perspective, ZionSiphon also reads as an early-stage attempt to approximate a Stuxnet-like attack model, but within a far less mature development ecosystem. It follows the same broad conceptual pathway leveraging a Windows foothold to reach and influence physical processes but lacks the deep engineering integration, protocol precision, and reliability that defined earlier state-developed ICS weapons. It is therefore best characterized as a process-aware prototype, reflecting ambition and direction rather than fully realized capability.

Despite its current limitations, the malware's architecture carries significant forward-looking implications. By decoupling ICS-specific logic from the underlying implant, ZionSiphon reflects a modular design philosophy that enables rapid iteration and reuse. The Windows-based execution layer provides a stable foundation onto which increasingly sophisticated OT-specific components can be layered. Future variants could therefore evolve quickly, incorporating:

- More complete protocol implementations (e.g., Modbus, DNP3, or vendor-specific interfaces)
- Improved environment detection and targeting precision
- Greater execution reliability and error handling
- Limited feedback mechanisms or controlled tasking capabilities

Such evolution would move the capability from conceptual disruption toward repeatable and controllable operational effects, narrowing the current gap between intent and execution.

The most significant implication is structural rather than purely technical. ZionSiphon signals that cyber-physical attack development is becoming more accessible. Unlike earlier ICS malware such as Stuxnet, which required extensive resources, specialized engineering knowledge, and tightly integrated development pipelines, this model leverages widely available tooling and incremental domain understanding. The barrier to entry is therefore lower, enabling a broader range of actors including contractors, proxy groups, or semi-professional operators to experiment with OT-oriented malware development.

This shift also increases the likelihood of proliferation and adaptation. The same architectural model could be repurposed across sectors by substituting environment-specific logic, extending beyond water infrastructure into energy, manufacturing, or transportation systems. Even if ZionSiphon itself remains limited, it represents a template for iterative development, where successive improvements progressively close the gap between conceptual capability and operational effectiveness.

Finally, the dual-use nature of the malware remains strategically important. In an environment where no confirmed physical attacks have occurred despite persistent targeting, artifacts like ZionSiphon may serve not only as technical tools but also as instruments of signaling and perception management. Their discovery, analysis, and public reporting contribute to an evolving perception of cyber-physical threat capability, influencing defensive postures, policy responses, and strategic calculations.

ZionSiphon should therefore be understood not as a fully realized ICS weapon, but as a directional indicator: a hybrid artifact that combines real host-level capability, emerging cyber-physical intent, and potential psychological or attribution-shaping effects.

Appendix A – Indicators of Compromise: ZionSiphon / SCADA_SecurityPatch_v8.4.exe

File Hashes

Primary sample

Type	Value
Filename	SCADA_SecurityPatch_v8.4.exe
SHA256	07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f
MD5	9f6265271f0b04e98ed28e414a8eee91
File size	100 KiB / 102400 bytes
Type	PE32 Mono/.NET executable
Compiler	Microsoft visual C# v7.0 / Basic .NET
imphash	f34d5f2d4577ed6d9ceec516c1f5a744

Dropped payload

Type	Value
Filename	svchost.exe
Path	%LOCALAPPDATA%\svchost.exe
SHA256	07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f
MD5	eb89f018e6c3a8e9de0a0452acb16e76
SHA1	7cdcb8f372ceb7f5d3c178d9649080106a932f9e

Dropped cleanup script

Type	Value
Filename	delete.bat

Dropped cleanup script

Type	Value
Path	%TEMP%\delete.bat
Size	121 bytes
SHA256	2537628e68a35124cad1c935634e70c90a62801403cbb427b262ff9ced03384a
MD5	b18baa0a3bfcd52bef58ded9402889b

Dropped validation log

Type	Value
Filename	target_verify.log
Path	%TEMP%\target_verify.log
Size	79 bytes
SHA256	d3b4737095600c1d87f0354d8246a1d4a22c737b92a9ec62571f8330cfd03c05
MD5	6a27e9439fa40cb89ee341133b03e4db

Host-Based Indicators

Files and artifacts

C:\SCADA_SecurityPatch_v8.4.exe

%LOCALAPPDATA%\svchost.exe

%TEMP%\delete.bat

%TEMP%\target_verify.log

C:\SCADA_SecurityPatch_v8.4.exe.config

C:\SCADA_SECURITYPATCH_V8.4.EXE.CONFIG

Files and artifacts

C:\SCADA_SecurityPatch_v8.4.INI

Persistence

Registry key	HKCU\Software\Microsoft\Windows\CurrentVersion\Run
--------------	--

Value	SYSTEMHEALTHCHECK
-------	-------------------

Data	%LOCALAPPDATA%\svchost.exe
------	----------------------------

Notes	Hybrid observed the persistence copy from C:\SCADA_SecurityPatch_v8.4.exe to %LOCALAPPDATA%\svchost.exe and creation of the SYSTEMHEALTHCHECK Run key.
-------	--

A.3 Execution and Runtime Indicators

RunAsAdmin	cmd.exe /c "%TEMP%\delete.bat"
------------	--------------------------------

SystemHealthCheck	CreateNoWindow
-------------------	----------------

Start-Process -FilePath	UseShellExecute
-------------------------	-----------------

-Verb RunAs	ProcessStartInfo
-------------	------------------

target_verify.log	GetExecutingAssembly
-------------------	----------------------

delete.bat

Mutex

GUID	{A1234567-B89C-40D1-ABCD-1234567890EF}
------	--

Path	\Sessions\1\BaseNamedObjects\{A1234567-B89C-40D1-ABCD-1234567890EF}
------	---

A.4 Process and Environment Indicators

Process / host discovery

gethostname	GlobalMemoryStatusEx
GetHostName	QueryPerformanceCounter
GetCurrentProcess	RtlGetVersion
GetProcesses	GetVersionExA
get_ProcessName	IsWow64Process
CreateToolhelp32Snapshot	WindowsIdentity.GetCurrent
NtQuerySystemInformation	CheckTokenMembership
GetSystemMetrics	OpenProcessToken
GetSystemInfo	

Environment and anti-analysis

IsDebuggerPresent	OobeInProgress
NtQuerySystemInformation	SystemSetupInProgress
vboxguestadditions	MachineGuid
VBoxGuest	PreferredUILanguages

Hybrid mapped the sample to execution guardrails, sandbox/VM checks, host discovery, process enumeration, and language/locale discovery.

A.5 Water / OT Targeting Indicators

Process name checks

DesalPLC	OsmosisPLC
----------	------------

Process name checks

ROController	DesalMonitor
SchneiderRO	RO_Filter
DamRO	ChlorineDose
ReverseOsmosis	RO_Membrane
WaterGenix	DesalFlow
RO_Pump	WaterTreat
ChlorineCtrl	SalinityCtrl
WaterPLC	
SeaWaterRO	
BrineControl	

Directory checks

C:\Program Files\Desalination	C:\Program Files\RO Systems
C:\Program Files\Schneider Electric\Desal	C:\Program Files\DesalTech
C:\Program Files\IDE Technologies	C:\Program Files\Aqua Solutions
C:\Program Files\Water Treatment	C:\Program Files\Hydro Systems

Configuration file checks

C:\DesalConfig.ini	C:\WaterTreatment.ini
C:\ROConfig.ini	C:\ChlorineControl.dat
C:\DesalSettings.conf	C:\RO_PumpSettings.ini
C:\Program Files\Desalination\system.cfg	C:\SalinityControl.ini

Facility / sector strings

Mekorot	Shafdan
Sorek	Schneider Electric
Hadera	IDE Technologies
Ashdod	WaterGenix
Palmachim	

Hybrid file metadata also lists ProductName and FileDescription as Mekorot, reinforcing the Israeli water-sector masquerade.

A.6 Configuration Manipulation Indicators

Chlorine_Dose	10
Chlorine_Pump	ON
Chlorine_Flow	MAX
Chlorine_Valve	OPEN
RO_Pressure	80

These remain high-confidence impact-oriented strings tied to chlorine handling and reverse-osmosis pressure manipulation.

A.7 Network and Protocol Indicators

Network APIs and socket capability

Hybrid confirms socket capability but also reports:

Network APIs

System.Net	NetworkStream.Read
System.Net.Sockets	NetworkStream.Write

Network APIs

TcpClient	TcpClient.Close
TcpClient.Connect	WSAStartup
TcpClient.BeginConnect	setsockopt
TcpClient.GetStream	gethostbyname
	GetAddrInfoW

Network activity

No relevant DNS requests

No relevant contacted hosts

No relevant HTTP requests

This supports the assessment that no operational C2 was observed during detonation.

Industrial protocol references

Protocol / pattern	Bytes
Modbus	
DNP3	
S7comm	
Modbus request pattern	01 03 00 00 00 0A
DNP3 partial pattern	05 64 0A 0C 01 02
S7comm partial pattern	03 00 00 13 0E 00
S7comm partial pattern	05 00 1C 22 1E

A.8 Targeting and Geofencing Indicators

IPv4 ranges observed in binary/memory

2.52.0.0 – 2.55.255.255

5.28.0.0 – 5.29.255.255

79.176.0.0 – 79.191.255.255

212.150.0.0 – 212.150.255.255

Hybrid directly reported the 2.52.0.0-2.55.255.255 and 5.28.0.0-5.29.255.255 ranges as potential IP ranges in binary/memory.

A.9 USB / Removable-Media Propagation Indicators

Recovered strings and behavioral primitives

CreateUSBShortcut	SetAttributes
-------------------	---------------

DriveInfo	.lnk
-----------	------

GetLogicalDrives	shell32.dll
------------------	-------------

GetFiles	svchost.exe
----------	-------------

CopyFileW

CopyFileExW

Assessed removable-media artifacts

\svchost.exe

*.lnk files in root of removable media

Shortcut TargetPath pointing to hidden executable

Icon spoofing via shell32.dll,4

Hybrid confirms GetLogicalDrives, file-copy behavior, and file-attribute capability, but did not capture a complete successful USB infection event during detonation. Therefore, USB propagation should be treated as **strongly supported by strings and vendor reverse engineering**, not as a fully observed sandbox behavior.

A.10 Self-Deletion and Cleanup Indicators

Cleanup artifacts

%TEMP%\delete.bat

cmd.exe /c "%TEMP%\delete.bat"

Observed deletion targets

C:\SCADA_SecurityPatch_v8.4.exe

%TEMP%\delete.bat

Hybrid observed cmd.exe executing %TEMP%\delete.bat and marking both the original sample and cleanup script for deletion.

A.11 MITRE-Relevant Behavioral Indicators

Technique ID	Name
T1055	Process Injection
T1055.002	Portable Executable Injection
T1059.003	Windows Command Shell
T1070.004	File Deletion
T1070.009	Clear Persistence
T1082	System Information Discovery
T1083	File and Directory Discovery
T1105	Ingress Tool Transfer

Technique ID	Name
T1106	Native API
T1112	Modify Registry
T1134	Access Token Manipulation
T1222	File and Directory Permissions Modification
T1480	Execution Guardrails
T1497.001	Virtualization/Sandbox Evasion
T1547.001	Registry Run Keys / Startup Folder
T1564.003	Hidden Window
T1622	Debugger Evasion

A.12 Negative Indicators / Absence-Based Intelligence

/

No confirmed C2 domains

No confirmed callback IPs

No observed DNS requests

No observed HTTP requests

No observed contacted hosts

No confirmed phishing infrastructure

No confirmed delivery infrastructure

No confirmed fully executed USB infection event in sandbox

These negative indicators suggest a constrained or pre-operational deployment model, likely centered on local execution, environmental validation, and workflow-assisted movement rather than remotely tasked command-and-control.

APPENDIX B Static Reverse Engineering of the Uploaded Sample

The uploaded file, identified by SHA-256

[07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f](#), is a PE32 Mono/.NET executable, indicating that all operational logic is implemented in managed code rather than native binaries. The import table is minimal and limited to the CLR bootstrap (mscorlib.dll via `_CorExeMain`), which is characteristic of .NET malware that delegates functionality to internal assemblies. This structural choice confirms that the sample is a Windows-hosted implant, not a controller-resident ICS payload, and that its operational model depends on execution within userland environments such as engineering workstations or operator systems.

The binary exhibits a coherent and internally consistent execution model. Embedded strings and method identifiers indicate that the malware initiates execution by attempting to relaunch itself with elevated privileges through PowerShell, using a command pattern consistent with `Start-Process -FilePath ... -Verb RunAs`. This is followed by persistence establishment via the Windows registry. Specifically, the malware is designed to write a Run key under `Software\Microsoft\Windows\CurrentVersion\Run` using the value name `SystemHealthCheck`, pointing to a staged payload masquerading as `svchost.exe` within the user's local application directory. This persistence mechanism is operationally credible and aligns with common masquerade techniques intended to blend malicious binaries with legitimate system processes.

The sample's most distinctive feature is its environment validation logic, which is explicitly tailored to water treatment and desalination systems. The binary contains numerous hardcoded file paths and configuration filenames associated with industrial processes, including chlorine control, reverse osmosis, salinity regulation, and water treatment operations. These include paths such as `C:\ChlorineControl.dat`, `C:\DesalConfig.ini`, `C:\RO_PumpSettings.ini`, and `C:\WaterTreatment.ini`, as well as vendor- or application-associated directories such as `C:\Program Files\Schneider Electric\Desal\config.ini` and `C:\Program Files\WaterGenix\system.conf`. The presence of these strings indicates that the malware performs host-based reconnaissance to determine whether it is executing within a relevant operational environment before proceeding.

This validation stage is further reinforced by recovered method names such as `IsDamDesalinationPlant`, `GetProcesses`, and `DriveInfo`, which suggest a structured approach to system classification. The malware likely enumerates running processes, inspects filesystem artifacts, and evaluates system characteristics to determine whether the host is associated with desalination or water treatment infrastructure. Only upon successful validation

does the malware proceed to its impact phase, indicating a gated execution model designed to minimize noise and avoid unintended activation.

The impact logic itself is revealed through a set of explicit configuration strings embedded in the binary. These include Chlorine_Dose=10, Chlorine_Flow=MAX, Chlorine_Pump=ON, Chlorine_Valve=OPEN, and RO_Pressure=80. These values correspond directly to operational parameters within water treatment systems and suggest that the malware is designed to modify or inject configuration data affecting chemical dosing and pressure control. While static analysis cannot confirm whether these values map precisely to real-world control systems, their specificity and coherence indicate a clear intent to disrupt physical processes, particularly those related to water quality and system stability.

In addition to its primary payload, the malware includes functionality for removable media interaction, as evidenced by strings such as CreateUSBShortcut, .lnk, and shell32.dll, 4. This suggests the ability to create deceptive shortcut files on USB drives, potentially enabling lateral movement or execution in segmented environments where direct network propagation is not feasible. This feature is consistent with operational environments in ICS networks, where air gaps or limited connectivity often necessitate physical transfer mechanisms.

The sample also implements a cleanup and self-deletion routine, using artifacts such as target_verify.log and delete.bat. These components indicate that the malware logs the outcome of its environment validation and, if conditions are not met or execution fails, initiates a self-removal process via a batch script. This behavior aligns with a low-footprint operational model, where the malware seeks to avoid detection by minimizing residual artifacts on non-target systems.

From a tradecraft perspective, the binary demonstrates a functional and intentional design. It includes persistence, privilege escalation, environment validation, removable media handling, and process-specific manipulation logic. These elements collectively support the assessment that the sample is a real and operational Windows-based implant, not merely a decoy or string-based artifact. However, the malware does not exhibit the characteristics of a mature ICS platform. There is no evidence of PLC firmware interaction, vendor-specific engineering tool manipulation, or deep integration with industrial control protocols at the controller level.

The most accurate classification is that this sample represents a host-side OT sabotage implant, designed to operate within Windows environments that interface with water treatment systems. Its strength lies in its targeting logic and process awareness, rather than in advanced ICS exploitation techniques. This places it in a transitional category of malware that bridges traditional IT compromise and OT disruption, relying on access to engineering or supervisory systems to influence physical processes.

Appendix D – Source Citations and References

Primary Malware Analysis Sources

1. BleepingComputer

“ZionSiphon malware designed to sabotage water treatment systems”

<https://www.bleepingcomputer.com/news/security/zionsiphon-malware-designed-to-sabotage-water-treatment-systems/>

2. Darktrace

“Inside ZionSiphon: Darktrace’s analysis of OT malware targeting Israeli water systems”

<https://www.darktrace.com/blog/inside-zionsiphon-darktraces-analysis-of-ot-malware-targeting-israeli-water-systems>

Malware Sample and Sandbox Analysis

3. VirusTotal

Sample Analysis – SHA256:

07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f

<https://www.virustotal.com/gui/file/07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f/details>

4. Hybrid Analysis

Sample Report (primary)

<https://hybrid-analysis.com/sample/07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f>

5. Hybrid Analysis

Imphash clustering results

<https://hybrid-analysis.com/search?query=imphash%3A%22f34d5f2d4577ed6d9ceec516c1f5a744%22>

6. ANY.RUN

Dynamic Analysis Report (Malicious Activity)

<https://any.run/report/07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f/eab8c32e-48a0-4333-a962-6e4dced301b8>

7. ANY.RUN

Dynamic Analysis Report (No Threat Detected Variant Execution)

<https://any.run/report/07c3bbe60d47240df7152f72beb98ea373d9600946860bad12f7bc617a5d6f5f/438f6689-1006-4a88-a870-70a392ceeaf8>

ICS / OT Malware Comparative References

8. FrostyGoop – Dragos
“FrostyGoop ICS Malware Targeting Operational Technology”
<https://www.dragos.com/blog/protect-against-frostygoop-ics-malware-targeting-operational-technology>
9. Industroyer2 – Google Cloud (Mandiant)
“Industroyer2: Old Malware, New Tricks”
<https://cloud.google.com/blog/topics/threat-intelligence/industroyer-v2-old-malware-new-tricks>
10. INCONTROLLER – Google Cloud (Mandiant)
“INCONTROLLER: State-Sponsored ICS Tool”
<https://cloud.google.com/blog/topics/threat-intelligence/incontroller-state-sponsored-ics-tool>
11. TRITON – National Cyber Security Centre
“TRITON Malware Targeting Safety Controllers”
<https://www.ncsc.gov.uk/information/triton-malware-targeting-safety-controllers>
12. Stuxnet – National Security Archive
“Stuxnet Dossier”
<https://nsarchive2.gwu.edu/NSAEBB/NSAEBB424/docs/Cyber-044.pdf>

ICS / OT Protocol and Architecture References

13. InfoSec Institute
ICS Protocol Overview (Modbus, DNP3, S7)
<https://www.infosecinstitute.com/resources/scada-ics-security/ics-protocols/>
14. Team Cymru
Industrial Cybersecurity and OT Exposure
<https://www.team-cymru.com/post/industrial-cybersecurity-for-ics-and-ot-devices>

Infrastructure and IP Attribution Context

15. RIPE NCC
IPv4 Allocation and Registration Data (Israel)
<https://www.ripe.net>
16. Bezeq
<https://www.bezeq.co.il>
17. Partner Communications Company
<https://www.partner.co.il>

18. Cellcom

<https://www.cellcom.co.il>

Water Infrastructure Context

19. Mekorot

<https://www.mekorot.co.il>

20. Israel Water Authority

https://www.gov.il/en/departments/water_authority