Unplugging PlugX: Sinkholing the PlugX USB worm botnet

By Sekoia TDR, Felix Aimé and Charles M.

Published: 2024-04-25 · Archived: 2026-04-05 23:44:23 UTC

Table of contents

Key Takeaways

- **In September 2023, we successfully sinkholed a command and control server linked to the PlugX worms.** For just $7, we acquired the unique IP address tied to a variant of this worm, which had been previously documented by Sophos.

- **Almost four years after its initial launch, between ~90,000 to ~100,000 unique public IP addresses are still infected, sending distinctive PlugX requests daily to our sinkhole**. We observed in 6 months of sinkholing more than 2,5M unique IPs connecting to it.

- **While studying the cryptography of PlugX's communications, we discovered that it was possible to send disinfection commands to the compromised workstations.** Two approaches can be implemented: one that disinfects only the workstation, and a more intrusive one that disinfects both the workstation and the USB drive.

- **Despite the fact that this worm cannot be completely stopped, we are offering the affected countries the possibility of disinfection**, with a concept of sovereign disinfection process.

In March 2023, Sophos published an article entitled "A border-hopping PlugX USB worm takes its act on the road" putting the light on a PlugX variant with worming capabilities. This variant, created in 2020, aimed to **propagate via compromised flash drives**, bypass air gaps, infect non internet facing networks and steal documents from them. According to the Sophos blogpost, all of these PlugX samples communicate with only one IP address, 45.142.166[.]112 hosted by GreenCloud.

In September 2023, we managed to take ownership of this IP address to sinkhole that botnet. We initially thought that we will have a few thousand victims connected to it, as what we can have on our regular sinkholes. However, by setting up a simple web server we saw a continuous flow of HTTP requests varying through the time of the day.

Facing that, we decided to record the received requests in a database in order to map the infections. In total, between **90 to 100k unique IP addresses** are sending PlugX distinctive requests every day to our sinkhole server since September 2023. If the botnet can be considered as "dead," as the operators don't control it anymore, anyone with interception capabilities or taking ownership of this server can send arbitrary commands to the infected host to **re-purpose it** for malicious activities.

Therefore, we looked at the concept of **sovereign disinfection**, by proposing to Law Enforcement Agencies and national Computer Emergency Response Teams, to remove the implant from the infected host, remotely.

This blog post details our process of sinkholing an IP address, the techniques used to gather telemetry from the infected workstations, and the inner workings of the communications cryptography of PlugX, enabling remote disinfection of workstations.

PlugX, an old cyber weapon in the Chinese arsenal

The first known version of PlugX was first seen during a Chinese campaign targeting government related users and a specific organisation in Japan, which started in 2008 according to Trend Micro.

It was mainly deployed against victims located in Asia until 2012, and then expanded progressively its pool of targets to occidental entities. Most of the time, PlugX is loaded by using a DLL Side-Loading scheme where a legitimate executable loads a malicious – or patched DLL – which then will map and execute in memory the core component of PlugX, which resides in an encrypted binary blob on the file system [T1574.002].

The management interface of PlugX allows the operator to manage several infected hosts with functionalities commonly seen in such backdoors as remote command execution, file upload/download, file system exploration, grab data in the context of the execution etc.

This backdoor, initially developed by Zhao Jibin (aka. WHG), evolved throughout the time in different variants. The PlugX builder was shared between several intrusion sets, most of them attributed to front companies linked to the Chinese Ministry of State Security.

When things go wrong: adding a wormable component to PlugX.

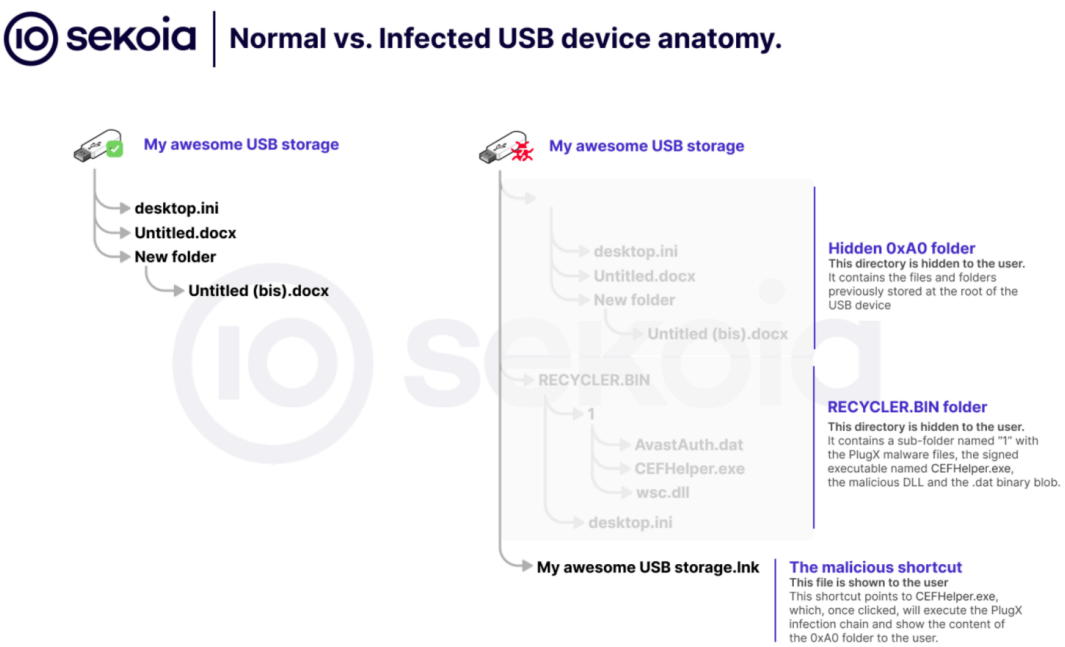
In July 2020, according to several researchers, the operators behind the **Mustang Panda** intrusion set had the (bad) idea to implement a wormable component to PlugX, possibly to target multiple countries in one campaign or expand its capabilities by reaching non-connected networks in order to steal files from the non-connected – but infected – workstations.

This wormable component infects connected USB flash drives by adding to them a Windows shortcut file taking the name of the infected flash drive, and a DLL side loading triad (legitimate executable, malicious DLL and binary blob) inside the drive RECYCLER.BIN hidden folder. The legitimate content of the USB devices is moved to a new directory whose name is the non-breaking space character (hexadecimal ascii code: 0xA0).

As for the Raspberry Robin USB worm, when a user opens the USB device, only a shortcut with the name of the USB device is presented to him, pushing him to click on it. By clicking on the shortcut, the PlugX infection chain

is executed. PlugX starts by closing the current window and reopening a new one in the directory (as previously mentioned named 0xA0) containing the legitimate files.

Then, it copies itself to the host inside %userprofile%/AvastSvcpCP/, and enables its persistence by creating a new key under HKCU[…]\CurrentVersion\Run registry Key. Finally, it re-executes itself from the host before terminating. Once executed from the host, the worm component of this PlugX variant checks every 30 seconds for the connection of a new flash drive to automatically infect it.



What could go wrong? The worm, which aims to target specific networks, **becomes uncontrollable** and replicates itself on many networks through flash drives, pushing the operators to abandon their unique C2? Indeed, the inner workings and the management interface of PlugX was not designed for managing thousands of infected hosts.

The remaining question is whether the Command and Control (C2) server was subjected to a Distributed Denial of Service (DDoS) attack due to an influx of victims, or was the campaign successful to such an extent that the threat actors found it feasible to abandon their C2. We don't know.

One thing is sure, a very short timeframe – a mere month – was observed between the compilation date for the Dynamic Link Library (DLL) wsc.dll, and the unique noted appearance by a trusted source of the IP address 45.142.166[.]112 acting as a C2 associated to PlugX.

Sinkholing some parts of malicious infrastructures by registering expired domain names or non-registered domains (in the case of domain generation algorithms or backup infrastructures) is quite common in the Threat Intelligence landscape. For example, we have done it quite recently one of the infection vectors of the Raspberry Robin botnet. However, taking ownership of a specific IP address is less common and can sometimes be more challenging.

After having checked that this IP was not used anymore via nmap, we simply kindly asked the hosting company to take ownership of the command and control server used in this campaign 45.142.166[.]112. Thanks to its

responsive and comprehensive customer support, we got a shell on a box with this specific IP address for $7 in a few minutes, allowing us to start our initial investigation.

It is worth to know that other variants of this worm exists, and three other C2s are known by the industry. One of them in particular (103.56.53[.]46) shows that it has an InetSim listening on it leading us to think that it was sinkholed silently by other security researchers or… threat actors ?

When your SSH lags, it's a good sign of something bad happening.

Upon establishing a connection to the server, we noted a significant delay, which suggested that multiple attempts were being made to access it. By setting up an ephemeral web server for a few seconds, we were bombarded by thousands of HTTP requests from infected workstations at ~2MB seconds, concluding that even if its C2 was being inactive since 2020, **the worm continued to propagate on a global scale**.

The PlugX worm use three different TCP ports (110, 443, 80) to communicate with in raw TCP or HTTP protocol. Its HTTP requests are quite common for PlugX implants, with their four discriminant headers (*-se, *-st, *-si, *-sn), as shown below. The attentive reader will note the typo in the hardcoded User-Agent value.

```
POST /[a-f0-9]{8} HTTP/1.1
Accept: */*
jsp-se: 0
jsp-st: 0
jsp-si: 61456
jsp-sn: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;Win64;x64)AppleWebKit/537.36
Host: 45.142.166.112:443
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
```

To handle this amount of connection attempts and to map the source of the infected hosts, we created a simple infrastructure. Its architecture consisted of an Nginx server, forwarding the connection attempts to a tiny script, which checks that the received HTTP connections were issued by a PlugX instance by looking at the URI and the headers.

If the request matches a PlugX one, the client IP address is forwarded to a second – and more robust – server, attempting to geolocate the infected workstation, adding it to a database with several metrics such as the associated autonomous system, the first seen and last seen time frame, the number of hits and the associated country.

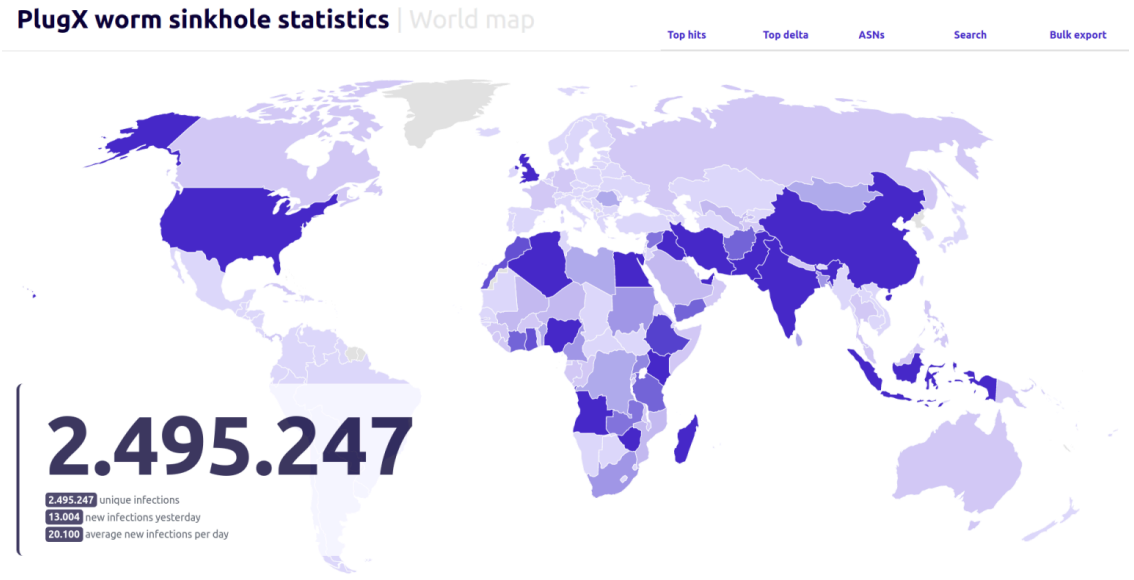
One point to bear in mind when establishing a sinkhole is the inability to reduce the bandwidth used by the beaconing of the compromised workstations. In this specific situation, the consumption fluctuated between 2MB and 0.8MB per second, depending on the time of day. Therefore, we had to optimise our sinkhole at different levels and not respond to all beconing requests to keep our small server.

Facing to that, we tested two approaches. Our first approach was to forward the request to the backend only if the outcome of random.randint(0,30) was 15. Given that PlugX continually sends beconing, we considered this an effective strategy. However, we missed 25% of the compromised workstations because PlugX often beacons for a short period of time before being quarantined by a security solution.

More recently, a transport-level approach was employed. This incorporated the use of iptables' hashlimit module which facilitated treatment of only one request per minute originating from a unique IP source and port, subsequently discarding the rest of the packets – and our answers – by DROPing them.

What's the final goal of this worm? Is the answer in the infected countries?

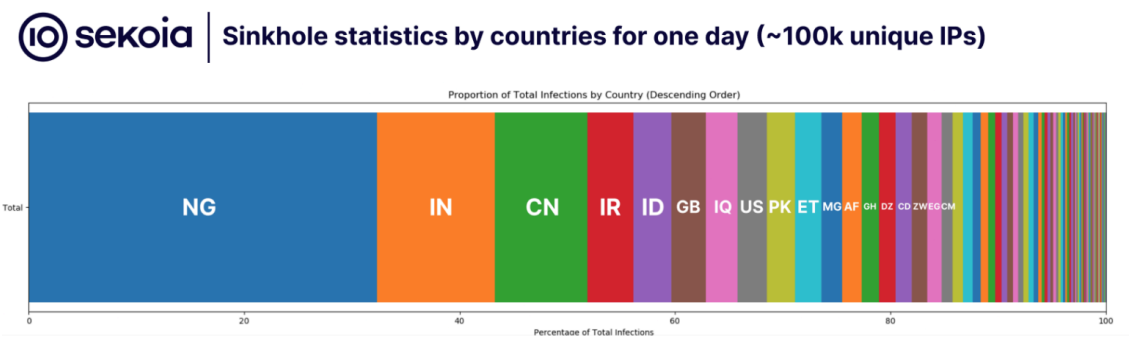
The created database enabled us to track the daily progress of worm infections and map them into an interactive world map as shown below, which can be accessed through a web portal. As of this writing, the worm has been observed in **over 170 countries** globally.



Before diving into the sinkhole data, it's important to acknowledge a few key points. Primarily, it's a drawback that the PlugX C2 communications lack unique victim identifiers, unlike many other implants. Consequently, the analysis of the data we've gathered relies solely on victim IP addresses, and we must remember that this approach has several limitations, such as:

- Dynamic IP addressing is still present: many internet subscribers still have dynamic IP addressing. Therefore, one infected workstation can be related to multiple public IP addresses through time.

Therefore, simply presenting a specific number of infections is **totally irrelevant** in the case of this botnet as it will growth every day. However, it can still be interesting to do a ranking of countries by IPs seen reaching our C2 server – as it can represent the countries where the worm is the most active. Therefore, the following chart shows a snapshot of the most infected countries on 3rd of April, 2024. It reflects a total of **100,952 unique IP addresses**.



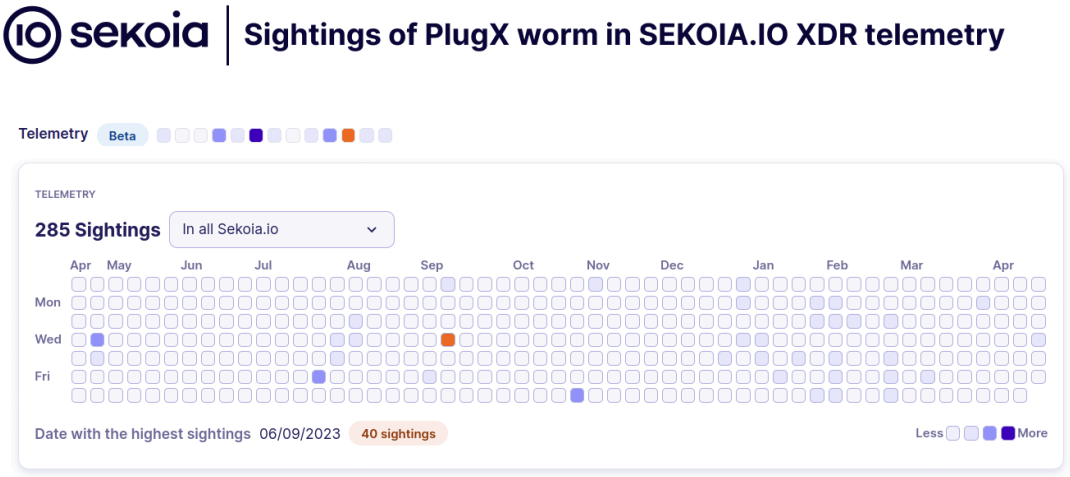
Based on that data, it's notable that around 15 countries account for over 80% of the total infections. It's also intriguing to note that the leading infected countries don't share many similarities, a pattern observed with previous USB worms such as RETADUP which has the highest infection rates in Spanish spelling countries. This suggests the possibility that this worm might have originated from multiple patient zeros in different countries.

The geographical distribution of the most infected countries, as presented on the previous map, may indicate a possible motivation behind this worm. However, this must be taken with a grain of salt, because after four years of activities, it had time to spread everywhere.

Many nations, excluding India, are participants in **China's Belt and Road Initiative** and have, for most of them, coastlines where Chinese infrastructure investments are significant. Analysing the geographical distribution of the infections from a security perspective reveals that numerous affected countries are located in regions of strategic importance for the security of the Belt and Road Initiative – like the **straits** of Malacca, Hormuz, Beb-el-Mandeb or Plak.

Consequently, it is plausible, though not definitively certain as China invests everywhere, that this worm was developed to **collect intelligence** in various countries about the strategic and security concerns associated with the Belt and Road Initiative, mostly on its maritime and economic aspects.

The likelihood that the targeting is connected to countries participating in the Belt and Road Initiative, or poses a security threat to it, is notably intriguing that the PlugX worm is linked to **Mustang Panda**, especially as this intrusion set is well-known for targeting many countries in the BRI.



From sinkholing to disinfection

Achieving a zombie worm: the questions.

Despite the command and control (C2) server being inactive for a few years, the worm has spread globally since its creation. Consequently, any individual who either controls the IP address or gains access at any point in the network pathway between an infected workstation and the C2 server (even when it's down) might attempt to **manipulate the worm behaviour**, to execute a payload, for example.

For instance, they could endeavour to execute their own payload on the infected workstation.
This situation prompted us to consider possible **disinfection methods** by using our access to this server. In order to explore these disinfection possibilities effectively, we must delve into three key questions:

This understanding is vital to **prevent side effects** and estimate the impact of a disinfection campaign. For example, if PlugX is removed from the USB devices, it will have a much bigger impact than if PlugX is only deleted from the workstation itself.

Reverse is the answer

Many articles document PlugX's features, whether they pertain to communication protocols or functionalities. However, not every variant implements all known commands. In this case, the number of commands is relatively low. Consequently, the first question is easy to answer: this variant does indeed have a **self-deletion command** (identified as 0x1005).

This command is quite straightforward as it requires no arguments. When PlugX receives the self-deletion command 0x1005 in an encrypted payload, it performs the following actions:

- Retrieves the current directory path and attempts to delete files and subdirectories.
- Retrieves the service name and deletes the corresponding registry key.
- Creates a batch file, %TEMP%/del_AsvastSvcpCP.bat, to delete the remaining files.
- Runs this script before terminating.

It is important to note that the deletion takes place from the current execution directory. However, **this PlugX variant only contacts the C2 when executed from the host**. This addresses the second question: it is possible to remove PlugX from the hosts but not from the infected USB devices.

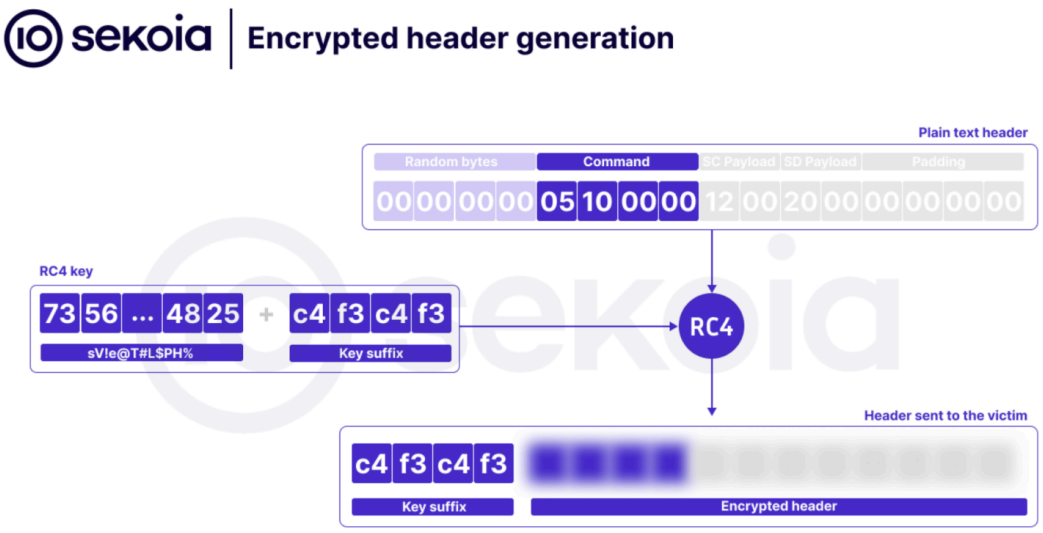
The last question is more complex to address. As previously mentioned PlugX can communicate via different protocols: TCP and HTTP. TCP communication is the simplest to understand. Each message is divided into two parts:

- An encrypted 16-byte header
- A payload, compressed and then encrypted. This payload contains the arguments of the command (none in the case of the deletion command).

The header contains:

- The command identifier;
- The size of the encrypted payload;
- the size of the decrypted and decompressed payload.

The **RC4 algorithm** is used to encrypt both the header and payload with the same key. This key is the concatenation of a hard-coded string in the code (sV!e@T#L$PH%) and the first four bytes of the header sent by the C2. It is then straightforward to send a deletion command, especially since there is nothing preventing the **reuse of the same four bytes** (to have the same key for all of the victims).



From a practical standpoint, we prefer using the HTTP protocol over raw TCP. HTTP communications are based on the same principles as TCP communications even though they require to understand how the state machine of the server is implemented via the HTTP headers.

However, it is quite straightforward to send the deletion command as it can be sent as a response to the first POST request (corresponding to the session opening) and takes just a few bytes. As we can use the same key suffix for all of the victims, we can send the same encrypted deletion payload to all of the victims in order to remove the implant from them.

Removing or not removing PlugX form the flash drives…

… that's the question. Indeed, PlugX hides the files stored on the flash drive, and it is only during the execution of PlugX that these files are revealed to the user. If a security solution removes one of the files associated with PlugX on the flash drive, the end user will consequently lose easy access to their own files.

The only disadvantage of this is that the flash drive remains infected. That's why we thought of implementing our own payload to send to the workstation using PlugX functionalities. Here, this payload needs to delete PlugX (as the auto delete command) but also checks if an infected flash drive is plugged in. In this case, it deletes PlugX from it and restores the original flash drive directory structure.

This is possible because PlugX allows the upload and execution of a payload using the following commands:

- 0x1002: used to create a thread listening to a sequence of more complex command.
- 0x300e: used to expand a environment variable. It is used to expand the %TEMP% variable.
- 0x3007, 0x10003008, 0x10003009: upload our payload on the %TEMP% directory (CreateFile, WriteFile, CloseFile).
- 0x300c: Create a process from our uploaded payload.

This strategy requires more commands to send than the previous one but it works. Anyway, there are some **limitations**. We don't want to add a persistence mechanism to our payload so the infected flash drive needs to be plugged in when our payload is executed. Furthermore, this process is **very intrusive** as it modify the directory tree of the USB key. This is why it's implementation requires a lot of caution and a **code review by peers** is necessary before any deployment.

Legal implications of wild disinfection.

Given the potential **legal challenges** that could arise from conducting a widespread disinfection campaign, which involves sending an arbitrary command to workstations we do not own, we have resolved to defer the decision on whether to disinfect workstations in their respective countries to the discretion of national Computer Emergency Response Teams (CERTs), Law Enforcement Agencies (LEAs), and cybersecurity authorities.

The principle behind this concept of **sovereign disinfection** is pretty simple and straightforward. National CERTs and/or relevant LEAs ask to receive data from our sinkhole related to their specific countries. This allows them to assess whether it's necessary to initiate a disinfection process. Since some workstations located in one country can connect to the internet through another country (such as via VPNs or satellite internet providers), these authorities provide in return a list of autonomous systems that are OK to be disinfected.

Once in possession of the disinfection list, we can provide them an access to start the disinfection for a period of three months. During this time, any PlugX request from an Autonomous System marked for disinfection will be responded to with a removal command or a removal payload.

Limits of wild disinfection as a conclusion

As stated before, there are limitations to the two discussed methods of remote disinfection. Firstly, the worm has the capability to exist on air-gapped networks, which makes these infections beyond our reach. Secondly, and perhaps more noteworthy, the PlugX worm can reside on infected USB devices for an extended period without being connected to a workstation.

An additional important aspect to consider is the legal limitations. Worms have to be treated at a global scale but can't if we want to follow the law. Because simply desinfecting a country implies that it could be prone to reinfection. Evidently, it is a matter of time before these infections reoccur.

Therefore, **it is impossible to complete remove this worm**, by issuing a unique command to all the infected workstations. Consequently, we also strongly recommend that security editors create effective detection rules against this threat on the workstation side to prevent the reuse of this botnet in the future.

USB devices remain a major infection vector

Moving beyond the stereotypical scenario of a red-team using a dropped USB device in a parking lot to compromise a network, USB devices are still actively employed to infect both isolated and connected networks by both cybercrime and state-sponsored threat actors.

Therefore, we encourage you to deploy strong policies against that such as preventing any file execution from a removable device or completely disable removable storage in your organisation by applying the right Windows Group policies.

Thank you for reading this blog post. Please don't hesitate to provide your feedback on our publications by clicking here. You can also contact us at tdr[at]sekoia.io for further discussions.

Indicators of compromise

Files Hashes

```
432a07eb49473fa8c71d50ccaf2bc980b692d458ec4aaedd52d739cb377f3428
e8f55d0f327fd1d5f26428b890ef7fe878e135d494acda24ef01c695a2e9136d
3a53bd36b24bc40bdce289d26f1b6965c0a5e71f26b05d19c7aa73d9e3cfa6ff
2304891f176a92c62f43d9fd30cae943f1521394dce792c6de0e097d10103d45
8b8adc6c14ed3bbeacd9f39c4d1380835eaf090090f6f826341a018d6b2ad450
6bb959c33fdfc0086ac48586a73273a0a1331f1c4f0053ef021eebe7f377a292
b9f3cf9d63d2e3ce1821f2e3eb5acd6e374ea801f9c212eebfa734bd649bec7a
```

Infrastructure

```
45.251.240[.]55
45.142.166[.]112 (Sinkholed by Sekoia)
103.56.53[.]46
43.254.217[.]165
```

Yara rules

```
rule apt_MustangPanda_PlugXWorm_lnk {
    meta:
        id = "bea0b6e6-0999-431d-8ea2-324aa7497657"
        version = "1.0"
        malware = "PlugXWorm"
        intrusion_set = "MustangPanda"
        description = "Detects PlugXWorm Malicious LNK"
        source = "Sekoia.io"
        classification = "TLP:WHITE"
    strings:
        $ = "RECYCLER.BIN\\1\\CEFHelper.exe" wide
    condition:
```

```
        uint32be(0) == 0x4c000000
        and filesize < 2KB
        and all of them
}

import "pe"
rule apt_MustangPanda_MaliciousDLL_random_exports {
    meta:
        id = "d14ae417-bc6f-40b1-a027-084522fce516"
        version = "1.0"
        intrusion_set = "MustangPanda"
        description = "Detects malicious DLL used by MustangPanda"
        source = "Sekoia.io"
        classification = "TLP:WHITE"
    strings:
        $trait = { 66 89 55 FC }
    condition:
        pe.is_dll() and filesize < 100KB and
        for any e in pe.export_details: (
                $trait in (e.offset..e.offset+50)
            and e.name matches /^[a-z]{10,}$/
        )
        and not pe.is_signed
}
```

Feel free to read other Sekoia TDR (Threat Detection & Research) analysis here :

Share

mustang panda plugx sinkhole worm

Share this post:

Source: https://blog.sekoia.io/unplugging-plugx-sinkholing-the-plugx-usb-worm-botnet/