The Hunt for the Lost Soul: Unraveling the Evolution of the SoulSearcher Malware

fortinet.com/blog/threat-research/unraveling-the-evolution-of-the-soul-searcher-malware

February 25, 2022

A threat report published by Symantec in October 2021 recently caught our attention. It discusses an unknown threat actor conducting an espionage campaign in Southeast Asia using a new custom malware arsenal. What piqued our curiosity most was the mention of a DLL payload loaded from the registry that had yet to be discovered.

The reason the module was difficult to find became apparent after analyzing its loader. The module is stored as a compressed blob with a custom header in the registry. It is never written to disk, rendering it unlikely to appear in datasets like VirusTotal.

And so, we embarked on a journey to hunt for the lost module. We have now uncovered a sample of the module and a plethora of components and variants dating as far back as 2017. Reverse engineering the samples has allowed us to observe the progression of the development of this malware throughout the years. Over time, custom code was added, components were upgraded, capabilities expanded, the code became neater, and modularity increased.

This blog will examine the different components of this malware and their progression over time, thereby mapping the evolution of the Soul malware framework.

Affected Platforms: Windows
Impacted Users: Windows Users
Impact: Collects sensitive information and executes additional malicious modules
Severity Level: Critical

Theory of Evolution

In the earliest phase, the attackers used a backdoor that incorporated code of the open-source Gh0st RAT and NetBot Attacker tools, albeit with considerable modifications. The backdoor is embedded as a compressed blob in its dropper executable, which writes it to disk and runs it.

Within a year, the backdoor's code was refactored and had custom code added to it, completing its transformation into what we refer to as a Soul module. Its loader, which we dubbed SoulSearcher, changed as well. Instead of dropping the payload to disk, the compressed module is stored in the registry and is loaded in-memory.

Since the beginning of 2020, we have detected increasingly intricate SoulSearcher variants, some of which support loading multiple modules from the registry. They have significantly transformed over time, and their configuration artifacts shed light on possible Soul module capabilities.

Aside from the backdoors, additional tools were used, such as keyloggers and a custom-compiled 7zr tool (reduced standalone 7-zip).

A complete timeline of the various components is depicted below, beginning in 2017 with the first keylogger and backdoor and ending with the recent SoulSearcher variants found in November 2021.

Figure 1: Timeline of the Soul malware framework

Side note: This timeline is based on compilation timestamps, and although these can be tampered with, in the case of this malware framework, we consider them to be authentic. This is partly because the time distribution of the collected samples correlates with our understanding of the components' capabilities and their sightings in the wild. Furthermore, related samples such as loaders and their payloads were compiled within seconds of each other.

Several characteristics are common in all of the components we found:

- DynamiCall import obfuscation from the leaked source code of the notorious Hacking Team's
- RCS backdoor
- Stack strings
- Data structure similarity, such as the configuration structure
- Encryption and compression algorithms
- Names of mutexes, events, and file mappings
- Adjacent compilation timestamps

The Original Soul Backdoor

This is the backdoor used in the earliest phase by the threat actor. It was compiled in October 2017 and used revised code from public repositories and other malware leaked online, such as:

- DynamiCall
- Gh0st RAT
  - File manipulation functions
  - CMD shell code
  - Communication messages and structures
- HTran (an open-source connection bouncer tool)
- 7zip

The backdoor is a DLL dropped to the disk by a simple dropper. The dropper LZMA-decompresses the backdoor and a configuration that they both share. The dropper writes the backdoor to a path specified by the configuration and appends the configuration to it as an overlay. Depending on the command-line argument passed to the dropper, the backdoor is executed with LoadLibrary or rundll32.exe. Finally, the dropper deletes itself from the disk.

Configuration

The backdoor reads its configuration from its file overlay and decrypts it by subtracting and XORing each byte with 0x13.

The configuration begins with a sequence of bytes whose significance is unknown but is identical in all the samples we found. Other fields are the backdoor file name or full path, the C2 address, and the port in little-endian. The configuration also contains a service name and description, both unused. In one sample, the string “NetBot” is set as the file name.

Figure 2: Configuration for the backdoor

Two other fields, an array of DWORDs and a flag (highlighted at offset 0x1f0), control if and when the backdoor should suppress command-related communication.

The array's values determine the days and hours to accept commands. In this sample, all values are 2. Each index represents a particular day of the week and the hour of the day. If the value at a given index is 0, requests for commands are withheld on the corresponding day and hour.

The flag determines if receiving commands should be suppressed while there is activity on the machine, according to the following conditions:

- Other than the current session, there is an active console session in the system
- Other than the current session, there is an active or connected RDP session in the system

The sessions are monitored by using the WTSRegisterSessionNotification and WTSEnumerateSessions APIs.

Depending on the configuration, the backdoor can receive commands in active mode (as a client) or passive mode (as a server). There are two port numbers, one for each mode.

- If the server port is not 0, the backdoor contacts the server to receive commands.
- If the listening port is not 0, the backdoor listens on that port and awaits commands from incoming connections (only one connection may be active at a time).

Communication

Messages to the C2 server, including requests for commands, have a fixed structure. Every request to the server is composed of hardcoded HTTP headers impersonating legitimate network traffic to taboola[.]com.

Figure 3: Constant HTTP headers sent in requests to the server

The structure of the HTTP body sent to the server is depicted below. CompressedBuffer is zlib-deflated data.

```
struct BackdoorRequest {
        DWORD MessageType;
        DWORD DecompressedBufferSize;
        DWORD DecompressedBufferSize;
        DWORD CompressedBufferSize;
        BYTE
CompressedBuffer[CompressedBufferSize];
};
```

Figure 4: Format of message body sent by the backdoor

Receipt of commands begins when the backdoor sends information about the machine to the server with a MessageType of 0x11000000:

- Hostname
- IP address(es)
- CPU architecture
- RAM size

The server response structure is similar to that of the request:

```
struct ServerResponse {
        DWORD CommandType;
        DWORD DecompressedBufferSize;
        DWORD DecompressedBufferSize;
        DWORD CompressedBufferSize;
        BYTE
CompressedBuffer[CompressedBufferSize];
};
```

Figure 5: Format of server response message body for backdoor requests

In a separate thread, the backdoor may signal the server if command receipt suppression is currently in effect with MessageType 0x1100000B.

Commands

When the server sends a command, it is one of the CommandType values in the table below, and the CompressedBuffer field is empty. The backdoor sends out an additional request for the command's parameters, with a MessageType value specified according to the specific command.

Type	Name
0xFFFFFFFF	There are no commands to run.
0x20000000	Close the socket and stop receiving commands.
0x21000000	File manipulation functions that include moving, copying, deleting, downloading, and/or uploading files.
0x23000000	It opens an interactive CMD shell, allowing the attacker to execute CMD commands until terminating the shell by sending the “Exit” command.
0x38000000	transmitdata function of HTran.

Figure 6: Table of commands implemented by the backdoor

The same is true when the backdoor works in passive mode, except it is limited to handling file manipulation, CMD, and close socket commands.

The Soul-Searching Loaders

SoulSearcher is a type of second-stage loader seen in the wild since November 2018. All the samples we found are DLLs with a similar flow of operation. They are responsible for executing the Soul module payload and parsing its configuration.

The major differences between the SoulSearcher variants are the type of configuration passed to the payload and the location where the configuration and payload are stored.

Configuration Format	Date of Earliest Sample	Configuration Location	Payload Location
Binary	Nov 2018	• Overlay + registry • File	• Overlay • File
XML	Apr 2020	• Registry • File mapping • File	Registry
Semicolon-separated	Aug 2021	Embedded in SoulSearcher	Registry

Figure 7: Table of SoulSearcher types

Binary SoulSearcher

These are the earliest SoulSearcher samples in our possession. One of these samples has its payload—a Soul module—embedded in it. Every sample exports two functions: DumpAnalyze and DumpAnalyzeEx.

First, SoulSearcher searches for the module and configuration, either in its overlay data or files on the disk. If they are found, it saves the module to the registry. Regardless, SoulSearcher then fetches the payload from the registry, reflectively loads it, and passes the configuration to it as an argument.

The configuration is located at the end of the overlay and is decrypted using SUB-XOR 0x13. It has the same format as that of the original Soul backdoor, with an additional field that determines the size of the compressed Soul module in the overlay. Another part of the configuration is retrieved from the HKCU\Software\OIfkO2i1 registry value and decrypted with SUB-XOR 0x79. If it doesn't exist, this path is also queried in the other users' registry hives.

If the argument "-h <HANDLE>" was passed to the SoulSearcher's export, the configuration and the payload are extracted from sdc-integrity.dat instead of the overlay. They are extracted in the exact same way as before. The supplied argument is a handle to a DLL used to retrieve the directory path in which the .dat file resides.

In any case, the module is saved to the registry at HKCU\Software\kuhO6Ba0kT.

XML SoulSearcher

Every XML SoulSearcher begins with obtaining the configuration previously dropped by an unknown component. Most samples retrieve it from the registry, and some have the option of retrieving it from a file mapping object or a file on the disk.

For example, one sample retrieves the configuration from one of the following, depending on whether it is running as a service:

- A value name in the format of a GUID under the service parameters at HKLM\SYSTEM\CurrentControlSet\Services\<ServiceName>\Parameters
- A file mapping object named Global\CacheDataMapping

The retrieved binary data has the following structure:

```
struct StoredConfiguration {
        DWORD Magic;
        DWORD Unused;
        BYTE  LzmaProperties[5];
        DWORD ConfigSize;
        DWORD CompressedConfigSize;
        BYTE  ConfigMD5[0x21];
        BYTE  CompressedConfigMD5[0x21];
        BYTE
CompressedConfig[CompressedConfigSize];
};
```

Figure 8: Structure of configuration fetched from the registry

The structure is processed in the following manner to retrieve the XML configuration:

1. Verify that the size of CompressedConfig is equal to CompressedConfigSize
2. Verify that CompressedConfigSize and ConfigSize are not 0
3. Verify that both MD5 checksums are not 0
4. Ensure that Magic holds the byte sequence 86 AE 00 00
5. Perform MD5 checksum validation of the compressed configuration
6. LZMA-decompress the configuration
7. Perform MD5 checksum validation of the decompressed configuration

In one variant, an extra step is taken at the start to decrypt the registry data using AES-256 CBC. The key is retrieved from one of two hardcoded paths.

Older samples deserialize the resulting string with the CreateXmlReader API, while newer samples use the TinyXML open-source library. The XML attribute names shed light on the Soul modules loaded from the registry.

Semicolon SoulSearcher

Beginning in August 2021, SoulSearcher variants began using a hardcoded semicolon-separated configuration instead of an XML one from the registry. The first variant of this type was compiled just over a full month before the release of Symantec’s report.

Figure 9: Example of a semicolon-separated configuration

This configuration lacks the indicative XML attribute names, like those in the XML configurations, resulting in a more obscure tool. Nevertheless, here is what we can say about some of these fields:

- We believe the first field, do5Kc1diLHgq5f6 represents the configuration type. In the XML configurations, the type is represented by the string X6bmLMbAL29AlxB.
- One of the values states whether the SoulSearcher was installed as a service. If so, the configuration includes fields for details about the service, such as its name.
- Some of the values determine which Soul modules should be loaded.
- One field may contain a registry path from which to load a Soul module (while other modules are loaded from hardcoded paths).

When the Soul is Found

Older SoulSearcher variants load a single Soul module, while some more recent XML and Semicolon SoulSearchers may load up to four, depending on their configuration.

```
struct StoredModule {
        DWORD Unused;
        QWORD ModuleSize;
        QWORD CompressedModuleSize;
        BYTE  ModuleMD5[0x21];
        BYTE
CompressedModule[CompressedModuleSize];
};
```

Figure 10: Structure of the payload fetched from the registry

Every module is fetched from the registry in a similar manner as the configuration:

1. Verify that the size of CompressedModule is equal to CompressedModuleSize
2. LZMA-decompress the module
3. Perform MD5 checksum validation of the decompressed module
4. Ensure that the architecture of the module matches the architecture of the SoulSearcher

This procedure is identical in every SoulSearcher sample apart from the Binary SoulSearchers, whose structure slightly differs.

The SoulSearcher reflectively loads the module in-memory and calls its Construct export. Some earlier variants also call additional exports of the module.

Soul Backdoor Reincarnated

We found that one Binary SoulSearcher sample from November 2018 had an embedded payload.

This Soul module closely resembles the original backdoor in terms of functionality, although its code is much neater. Thorough examination revealed that the code of the original backdoor was reorganized as various exports. For instance, the code responsible for sending and receiving HTTP messages was divided into the SendMsg and RecvMsg exports.

Figure 11: Soul backdoor module exported functions

Configuration

The SoulSearcher calls the module's BeginConnect export with the configuration as an argument. The configuration has the same binary format as the original backdoor's configuration but without the service-related fields.

Figure 12: Configurations of the original backdoor (left) and the newer backdoor module (right)

Communication

Unlike the original backdoor, this Soul module only receives commands as a client.

If resolving the server address via gethostbyname API fails, the backdoor also tries querying two hardcoded DNS servers using an undocumented feature of the DnsQuery API:

- 193.0.14.129 (DNS root server)
- 8.8.8.8 (Google Public DNS)

The constant headers of the request have been changed to impersonate traffic to s-microsoft[.]com, and the GetSubInfo export collects the machine information.

Figure 13: New constant HTTP headers sent in requests to the server

Commands

The message structures are the same as those of the original backdoor. As seen in the table below, several new command codes were not present in the original backdoor. When one of the five named commands is received, the backdoor downloads and executes a DLL from the server. The command names are disclosed in the binary and are passed to the command DLLs as part of their arguments. Because the DLLs themselves are unknown to us, we can only speculate on their functionality based on their names and the implementations of the same command types in the original backdoor code.

Type	Name
0xFFFFFFFF	There are no commands to run.
0x20000000	Close the socket and stop receiving commands.
0x21000000	**File**
0x23000000	**Cmd**
0x38000000	**Htran**
0x39000000	Update configuration in the registry. The server response buffer is ADD-XOR 0x79 encrypted before being written to the registry (hardcoded path).

0x3A000001	Free command structures and release command-related mutexes.
0x3B000000	**MemoryLoader**
0x3C000000	**UsbNtf**

Figure 14: Table of commands implemented by the backdoor module

An additional socket connection is created to download a command DLL from the server. First, the backdoor sends a message of type 0x1100000C with a buffer that contains the constant value 0x4096C083. Like all requests, it is sent via SendMsg in the aforementioned BackdoorRequest structure. Next, it sends another message of the same type, but this time the buffer is structured as shown below. The Architecture field contains a value of either 32 or 64 depending on the backdoor's architecture.

```
struct CommandRequest {
        DWORD
CommandType;
        BYTE
Architecture[6];
};
```

Figure 15: Structure of request for a command DLL from the server

The server replies to the backdoor with the following structure:

```
struct CommandResponse {
        DWORD Unused0;
        BYTE  Unused1[6];
        QWORD ModuleSize;
        QWORD CompressedModuleSize;
        BYTE  ModuleMD5[0x21];
        BYTE  CompressedModuleMD5[0x21];
        QWORD Unused2;
        BYTE
CompressedModule[CompressedModuleSize];
};
```

Figure 16: Structure of server response for a command DLL

The backdoor uses the structure to load the command DLL in the following manner:

1. Validate MD5 checksum of compressed module
2. LZMA-decompress the compressed module
3. Validate MD5 checksum of decompressed module
4. If steps 2 or 3 fail, reissue the request to the server
5. Reflectively load the module in memory
6. Call the module’s Construct export with arguments that include, among other things:
    1. The constant value 0x4096C083 (same value sent to the server priorly)
    2. The name of the command (such as “File” or “UsbNtf”)
    3. The backdoor configuration
    4. The CommandResponse structure received from the server

More Souls Than One

As mentioned earlier, each XML SoulSearcher parses an XML-formatted configuration that contains attributes with informative names. Based on such artifacts, we were able to classify potential payloads of various samples in our possession.

Backdoor

These SoulSearcher samples are closely coupled to their payloads to the extent that they are intricate orchestrators rather than plain loaders. In addition to parsing a configuration, they invoke multiple exported functions of the Soul module to create full backdoor logic. The configuration fields and imported function names indicate remote shell capability and the utilization of Dropbox.

Configuration Fields	Exports Names
• Ip • Dns • CntPort • LstPort • Blog • DropboxBlog • SvcName • SvcDisp • SvcDesc • SvcDll • OlPass • OlTime • SelfDestroy	• Construct • ConnectHost1 • ForceCloseSocket • CopyReserveMem • Recv • RecvEx • Send • SendEx • BindShell • Accept • TransmitData_htran • KillChildenProcessTree • ExtractIPToConnect • ExtractIPToConnect1 • GetDeviceInfoString1 • GetPseudoSocketInfo • Decrypt_ByteToByte

Figure 17: Configuration fields and imported function names seen in older SoulSearcher versions

Advanced RAT

One SoulSearcher parses numerous configuration fields different from the backdoor:

- AesPass
- ClipBoardMntEnable
- DestroyDate
- DestroyDay
- DestroyMode
- DestroyWiFiName
- DestroyWiFiSearchMinu
- DirDiskInternal
- DropboxAppToken1
- DropboxAppToken2
- DropboxAppToken3
- EnableDropbox
- EnableFileMnt
- EnableHijack1
- EnableKeyLog
- EnableService
- ExcludeDir
- FileExt
- FileSizeMb
- Hijack1DllPath
- Hijack1RegSubKey_MemMod1
- Hijack1RegValueName_Cfg
- Hijack1RegValueName_MemMod1
- IncludeDir
- RecDataPath
- RegKey_Exist
- RegKey_Rec
- RegSubKey_Exist
- RegSubKey_Rec
- RegValueDataSz
- RegValueName_Exist
- RegValueName_Rec
- SaveInFile
- SaveInReg
- ScreenMngEnable
- ServiceDescription
- ServiceDisplayName
- ServiceDllPath
- ServiceHide
- ServiceImagePath
- ServiceName
- ServiceRegValueName_Cfg
- ServiceRegValueName_MemMod1
- ServiceRegValueName_MemMod2
- ServiceRegValueName_MemMod3
- ServiceSessionIsolationBypass
- TriggerTime
- UsbExt
- UsbExtMode
- z7zPass
- z7zSizeMb
- z7zStoreDir

Figure 18: Configuration fields found in one SoulSearcher sample

If the EnableDropbox attribute is set to true, the SoulSearcher loads a module from the path specified by ServiceRegValueName_MemMod3. If the EnableKeylog is set, a module is loaded from the path specified by ServiceRegValueName_MemMod1.

Proxy

These samples' configuration indicates proxy capabilities over HTTP and HTTPS, as well as the ability to run CMD commands.

- CmdPrefix
- CmdSuffix
- EnableHttps1
- Port
- Port2
- ProxyIP1
- ProxyIP2
- ProxyPort1
- ProxyPort2
- ProxyUserName1
- ProxyUserName2
- ProxyUserPass1
- EnableHttps2
- Interval
- MachineGUID
- ProxyUserPass2
- RegPath
- RegRootKey
- RegValueName_Cfg
- RegValueName_Svr32
- RegValueName_Svr64
- URL2
- Url

Figure 19: Configuration fields related to proxy functionality

Additional Components

First Stage Loader

As mentioned before, SoulSearcher is a second-stage component. We also identified a first-stage loader of the Binary SoulSearcher variant.

This loader is a DLL with a single exported function, named SntpService, and depends on a utility DLL named SntpService.dll, which is expected to already reside on disk. These names are likely used to resemble a legitimate security software product of Sophos of the same name (as seen here).

The loader checks if its process name is either MSDTC.exe or svchost.exe prior to running SntpService in a new thread. In the latter case, a mutex named DBWinMutex_1 is created (also used in the Soul module).

The loader performs two operations. First, it decrypts two .dat files from its directory and saves the output to the registry:

- sdc-integrity.dat is written to HKCR\.rat\PersistentHandler\TypeFace
- scs-integrity.dat is written to HKCR\.rat\PersistentHandler\MagicNumber

The decryption scheme is AES-256 CBC with the SHA256 hash of a hardcoded value used as the key. Both files are then deleted from the disk, implying this procedure occurs only on initial infection or when updates are deployed.

Second, the data from the TypeFace value is used to load SoulSearcher. It consists of a structure that contains a buffer and its size. The loader skips the buffer's last 0x3d0 bytes, as those are its configuration, and passes the rest of the buffer to the Decrypt_ByteToByte function of SntpService.dll. The output is a PE, which the loader reflectively loads and then invokes its DumpAnalyze export. The loader passes a handle of itself to the SoulSearcher as an argument, both as a pointer and in string format: "-h <HANDLE>".

Additional exports of SntpService.dll are also resolved:

Figure 20: Imported functions from SntpService.dll

We found a variant of the utility DLL uploaded to VirusTotal with the name of Kaspersky Antivirus's AvpCon.dll. Similar to the Sophos case cited earlier, this is likely done to appear legitimate. Despite its exports being named "Encrypt" and "Decrypt", all functions actually perform LZMA compression or decompression. This correlates with a Binary SoulSearcher sample that we found compressed, not encrypted.

Figure 21: Exports of AvpCon.dll

Keyloggers

The keyloggers were compiled between mid-2017 to late 2020. They all share very similar code, with few changes between them. In addition to the keyloggers Symantec reported on, we found another sample from September 2020. Although its keylogging function is identical to the other samples, the rest of the code has significant differences.

The keyloggers read their configuration from a file with the same name but with the .dll extension trimmed. Our sample, however, uses a configuration from the registry, and the file acts as a kill-switch: if it exists, the keylogger terminates. This sample also has stack strings and DynamiCall obfuscations not present in previous samples.

The keylogger ensures it is running in Explorer.exe and retrieves its configuration by reading its own last 0x208 bytes and decrypting them. The decryption is done by adding and XORing each byte with constant values. Next, the encrypted configuration is set in the registry at HKCU\Software\F32xhfHX. On future executions, the configuration will be fetched from this key. The configuration contains two paths:

- Keylogger module file – C:\Windows\SndVolSSO.DLL
- Keylogging output file – C:\users\minh\AppData\Local\OneDrive\Cache.dat

Interestingly, the output file path includes a username, hinting that this sample may have been intended for a specific target machine.

The keylogger monitors keystrokes using GetRawInputData and clipboard data and logs them in an output file as plaintext. The output file is timestomped to make its timestamp identical to svchost.exe on the infected machine. Errors returned from GetRawInputData are logged to C:\ProgramData\Users.inf. The keylogger also logs IME virtual-key codes, which support some Asian languages.

Figure 22: Example of keylogger output file “Cache.dat”

Command-Line Executer Service

This is a lightweight service DLL that executes a CMD command from the registry key HKCR\.c\Type\Type00. It runs the command on 20:00, and if no process named powershell.exe is active on the system. It is compiled with DynamiCall obfuscation.

7zr.exe

This custom-compiled 7zr executable is modified to include DynamiCall obfuscation.

Conclusion

The Soul malware framework has been in active use since 2017, and the threat actors have been steadily evolving their tools and capabilities to this day. It should be emphasized that despite the reliance of the earlier tools on open-source code, custom keyloggers were already in use at the time, and significant development of custom code has transpired since. Its modular, multi-stage, reflectively executed payloads demonstrate competent adversarial tradecraft and are signs of a well-resourced group. Although the attackers’ identity is currently unknown, we believe that they are possibly state-sponsored.

The details shared in this report stem from the comprehensive analysis of numerous samples. Nevertheless, we have a feeling that this is just the tip of the iceberg, with more payloads and capabilities in the group’s arsenal to expose in the future.

Fortinet Solutions

FortiEDR detects and blocks these threats out-of-the-box without any prior knowledge or special configuration. It does this using its post-execution prevention engine to identify malicious activities:

Figure 23: FortiEDR blocking the Soul backdoor communication to the C2 server

All network IOCs have been added to the FortiGuard WebFiltering blocklist.

The FortiGuard AntiVirus service engine is included in Fortinet's FortiGate, FortiMail, FortiClient, and FortiEDR solutions. FortiGuard AntiVirus has coverage in place as follows:

W64/SoulSearcher.B7D1!tr
W32/SoulSearcher.B7D1!tr
W64/SoulSearcherKeyLogger.B7D1!tr.spy
W32/SoulSearcher.B7D1!tr
Data/SoulSearcher.B7D1!tr

In addition, as part of our membership in the Cyber Threat Alliance, details of this threat were shared in real-time with other Alliance members to help create better protections for customers.

Appendix A: MITRE ATT&CK Techniques

ID	Description
T1569.002	System Services: Service Execution
T1055	Process Injection
T1112	Modify Registry
T1567	Exfiltration Over Web Service
T1041	Exfiltration Over C2 Channel
T1132	Data Encoding
T1082	System Information Discovery
T1083	File and Directory Discovery
T1140	Deobfuscate/Decode Files or Information
T1071.001	Application Layer Protocol: Web Protocols
T1056.001	Input Capture: Keylogging
T1059.003	Command and Scripting Interpreter: Windows Command Shell

T1115	Clipboard Data
T1592	Gather Victim Host Information
T1090.001	Proxy: Internal Proxy
T1070.006	Indicator Removal on Host: Timestomp

Appendix B: IOCs

IOC	Type	Details
1af5252cadbe8cef16b4d73d4c4886ee9cecddd3625e28a59b59773f5a2a9f7f	SHA-256	SoulSearcher
a6f75af45c331a3fac8d2ce010969f4954e8480cbe9f9ea19ce3c51c44d17e98	SHA-256	SoulSearcher
c4efb58723fd75d51eb92302fbd7541e4462f438282582b5efa3c6c7685e69fd	SHA-256	SoulSearcher
edb14233eccb5b6e2d731831e7b18b8b17ea6a3f8925fb5899ce2ef985a66b68	SHA-256	SoulSearcher
fdf0db7f6b60d7563268c15c634adb47e8eec34adfcbf9b10e973916c7517157	SHA-256	SoulSearcher
c7481d6975646b605aba3fb11686e34ee205f7e280069e9d5bf0c1c2eca79be8	SHA-256	SoulSearcher
0f7af0cad4aade0e7058051a449059b35358ddda075d88b2d289625adc02deef	SHA-256	SoulSearcher
3cb4887bec169c75f58bc4ed1c6fd3703cc46512596e62186cf8329448dbb47b	SHA-256	SoulSearcher
cb954f06c94493c87f25651271657aeb1e3e24f26b6552d3e616bbc2dc660679	SHA-256	SoulSearcher
78feb564c4f6c240ddb17dd0f49ae96df04ee594ed24df81f583136fccf60c1d	SHA-256	SoulSearcher

bc91a4fb16f14fb1c436c2bdc7c80b87a02caa5de17897614d07bc7bda200590	SHA-256	SoulSearcher
7edd7d406159ab0eecb22ddbd6060de7c24a4eb0b61fa527935310b94d3b9db4	SHA-256	SoulSearcher
b02b8b6c3d517c6b8652b898963068ba12cd360b5cdcf0aad5fe6ff64f0e9920	SHA-256	SoulSearcher
ec164902cbe8daaa88ae923719c5dac900715f3e32d4cea6e71ca04c7cecf3e2	SHA-256	SoulSearcher
bac4b50727c69ca7cc3c0a926bb1b75418a8a0eabd369a4f7118bb9bba880e06	SHA-256	First stage loader for SoulSearcher
69a9ab243011f95b0a1611f7d3c333eb32aee45e74613a6cddf7bcb19f51c8ab	SHA-256	Original Soul backdoor
579fa00bc212a3784d523f8ddd0cfc118f51ca926d8f7ea2eb6e27157ec61260	SHA-256	Original Soul backdoor
8ff18b6fb5fe4f221cd1df145a938c57bdd399dc24e1847b0dc84a7b8231458f	SHA-256	Original Soul backdoor
f97161aaa383e51b2b259bb618862a3a5163e1b8257832a289c72a677adec421	SHA-256	Original Soul backdoor dropper
d3647a6670cae4ff413caf9134c7b22b211cb73a172fc1aa6a25b88ff3657597	SHA-256	Original Soul backdoor
f5cd13b2402190ec73c526116abea5ebab7bd94bcdb68cc2af4f3b75a69ba9c5	SHA-256	Keylogger
a15eda7c75cf4aa14182c3d44dc492957e9a9569e2d318881e5705da2b882324	SHA-256	Keylogger
967e8063bd9925c2c8dd80d86a6b01deb5af54e44825547a60c48528fb5f896d	SHA-256	Keylogger
64f036f98aad41185163cb328636788a8c6b4e1082ae336dad42b79617e4813d	SHA-256	Keylogger
7b838fcad7a773bfd8bc26a70f986983553d78b4983d0f2002174f5e56f7f521	SHA-256	Soul backdoor

40fda8137d8464d61240314b6de00ae5c14ed52019e03e4dcadfc00b32c89d23	SHA-256	Command-line executer service
5dee99beb0b6ba1ebdb64515be1d9307262d9b57b0900310d57290dca40bb427	SHA-256	7zr.exe
6b70ad053497f15b0d4b51b5edabeced3077dddb71b28346df7c7ea18c11fcdf	SHA-256	7zr.exe
852c98a6fbd489133411848775c19a2525274eac9a89a09a09d511915c7cbafc	SHA-256	AvpCon.dll
gmy.cimadlicks[.]net	Network	-
app.tomelife[.]com	Network	-
community.weblives[.]net	Network	-
23.91.108[.]12	Network	-
Global\vQVomit4	Mutex	-
Global\mFNXzY0g	Mutex	-
Global\DefaultModuleMutex	Mutex	-
Global\DBWinMutex_1	Mutex	-
Global\DBWinMutex_2	Mutex	-
Global\VirusScanWinMsg	Event	-
Global\3GS7JR4S	Event	-
Global\SecurityEx	Event	-
Global\CacheDataMappingFile	File mapping	-
C:\Windows\System32\wlbsctrl.dll	File name	-

C:\Windows\System32\ikeext2.dll	File name	-
C:\Windows\System32\d6w48ttth.dll	File name	-
C:\Windows\System32\shsvc.dll	File name	-
C:\Windows\System32\netcsvc.dll	File name	-
C:\Windows\System32\fc2qhm7r9.dll	File name	-
C:\Windows\SndVolSSO.DLL	File name	-
SvrLdr_xpsservices.dll	File name	-
timedateapi.dll	File name	-
msfte.dll	File name	-
wsecapi.dll	File name	-
C:\Programdata\Microsoft\svchost.exe	File name	-
NvStreamer.dll	File name	-
Helpsvc32.dll	File name	-
SVCLDR64.dll	File name	-
DataOper64.dll	File name	-

C:\ProgramData\Users.inf	File name	-
%LOCALAPPDATA%\OneDrive\Cache.dat	File name	-
C:\ProgramData\Security_checker\sc.dll	File name	-
C:\ProgramData\Xps viewer\xpsservices.dll	File name	-
C:\Program Files (x86)\Common Files\System\ado\msado28.dll	File name	-
C:\ProgramData\networks.dat	File name	-
C:\ProgramData\Microsoft\Crypto\RSA\Keys.dat	File name	-
SntpService.dll	File name	-
sdc-integrity.dat	File name	-
sds-integrity.dat	File name	-
HKCR\.z\OpenWithProgidsEx	Registry	-
HKCR\.z\OpenWithListEx	Registry	-
HKCR\.sbr\Order	Registry	-
HKCR\.sbr\StartOverride	Registry	-
HKU\<any_key>\Software\kuhO6Ba0kT	Registry	
HKU\<any_key>\Software\OIfkO2i1	Registry	-
HKU\<any_key>\Software\7QAEGXJc	Registry	-

HKCR\.c\Type\Type00	Registry	-
HKR\Software\Microsoft\EventSystem\8C345CCE-5C37-446E-9E36-B57A54FC9C45	Registry	-
HKLM\SYSTEM\CurrentControlSet\Services\<service>\Parameters\8C345CCE-5C37-446E-9E36-B57A54FC9C45	Registry	-
HKR\.kci\PersistentHandler	Registry	-
HKCR\.3gp2\Perceived-Type	Registry	-
HKCR\.3gp2\Content-Type	Registry	-
HKCR\.rat\PersistentHandler\MagicNumber	Registry	-
HKCR\.rat\PersistentHandler\TypeFace	Registry	-
HKCU\Software\Microsoft\FTP\MostRecentApplication	Registry	-
HKCU\Software\Microsoft\FTP\UserInfo	Registry	-
HKCU\Software\F32xhfHX	Registry	-

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.