Operation Sharpshooter

Campaign Targets Global Defense, Critical Infrastructure

McAfee Advanced Threat Research
Operation Sharpshooter

The McAfee® Advanced Threat Research team and McAfee Labs Malware Operations Group, employing McAfee® Global Threat Intelligence, have discovered a new global campaign targeting nuclear, defense, energy, and financial companies. This campaign, Operation Sharpshooter, leverages an in-memory implant to download and retrieve a second-stage implant—which we call Rising Sun—for further exploitation. According to our analysis, the Rising Sun implant uses source code from the Lazarus Group’s 2015 backdoor Trojan Duuzer in a new framework to infiltrate these key industries.

Operation Sharpshooter’s numerous technical links to the Lazarus Group seem too obvious to immediately draw the conclusion that they are responsible for the attacks, and instead indicate a potential for false flags. Our research focuses on how this actor operates, the global impact, and how to detect the attack. We shall leave attribution to the broader security community.

Have We Seen This Before?
This campaign, while masquerading as legitimate industry job recruitment activity, gathers information to monitor for potential exploitation. Our analysis also indicates similar techniques associated with other job recruitment campaigns.

This research has uncovered a new implant framework using code from the 2015 backdoor Duuzer, which was last seen targeting South Korea and Japan in 2015. Apart from Rising Sun, we have seen no other variants since that time.

Global Impact
In October and November 2018, the Rising Sun implant has appeared in 87 organizations across the globe, predominantly in the United States, based on McAfee telemetry and our analysis. Based on other campaigns with similar behavior, most of the targeted organizations are English speaking or have an English-speaking regional office. This actor has used recruiting as a lure to collect information about targeted individuals of interest or organizations that manage data related to the industries of interest. The McAfee Advanced Threat Research team has observed that the majority of targets were defense and government-related organizations.
Campaign Analysis

This operation began October 25. A series of malicious documents carried the author’s name Richard. These documents contained Korean-language metadata, indicating they were created with a Korean version of Microsoft Word. All the malicious documents had English-language job description titles for positions at unknown companies, distributed by an IP address in the United States and through the Dropbox service. The documents contained a malicious macro that leveraged embedded shellcode to inject the Sharpshooter downloader into the memory of Word. Once the Word process was infected, the downloader retrieved the second-stage implant Rising Sun.

The shellcode of the downloader is 3.1KB in size and retrieved another implant hosted at hxxps://www.kingkoil.com.sg/query.php.

Figure 1. Targeted organizations by sector in October 2018. Colors indicate the most prominently affected sector in each country. Source: McAfee® Global Threat Intelligence.
Figure 2. Infection flow of the Rising Sun implant, which eventually sends data to the attacker's control servers.
Shellcode behavior
The shellcode executed by the Visual Basic for Applications macro in winword.exe acts as a simple downloader for the second-stage implant. The shellcode takes four steps to infect the endpoint with the second-stage payload:

1. It builds Library and API names by populating string arrays using hardcoded bytes. (String construction is done 1 byte at a time.) This technique is used for constructing all strings in the shellcode, including the control server information.

   ```
   mov  byte ptr [esp+1C0h], 75h ; 'u'
   mov  byte ptr [esp+1C9h], 72h ; 'r'
   mov  byte ptr [esp+1C8h], 60h ; 'l'
   mov  byte ptr [esp+1C1h], 56h ; 'm'
   mov  byte ptr [esp+1CCh], 68h ; 'o'
   mov  byte ptr [esp+1CDh], 66h ; 'n'
   mov  byte ptr [esp+1C Eh], 2EH ; '-'
   mov  byte ptr [esp+1Ch], 64h ; 'd'
   mov  byte ptr [esp+1D0h], 6Ch ; '1'
   mov  byte ptr [esp+1D1h], 6Ch ; '1'
   mov  byte ptr [esp+1D2h], 0
   ```

2. It resolves the Libraries and APIs using LoadLibraryA(), GetProcAddress():
   - urlmon.dll
   - shfolder.dll
   - ntdll.dll
   - kernel32.dll
   - shell32
   - LoadLibraryA
   - GetProcAddress
   - URLDownloadToFileA
   - SHGetFolderPathA
   - strcpy
   - strcat
   - CreateProcessA
   - memset
   - ShellExecuteA
3. The implant downloads two files from its control server:

- **Second-stage payload**: The second-stage binary is downloaded from
  https://www[dot]kingkoil.com.sg/query.php to the startup folder on the endpoint:
  `%Startup%\mssync.exe`
  This step ensures persistence on the system for the second-stage implant as part of the download process, thereby removing the need for the second-stage implant to set up persistence for itself.

- **Second OLE (Word) document**: Another OLE document is downloaded from
  https://www[dot]kingkoil.com.sg/Strategic Planning Manager.doc to:
  `%LOCALAPPDATA%\Strategic Planning Manager.doc`
  This document is probably benign, used as a decoy to hide the malicious content.

```
lea   eax, [esp+1EH] ; CSIDL_STARTUP\mssync.exe
dec   eax
xor   ecx, ecx
call  dword ptr [esp+68H] ; URLDownloadToFileA
```

Figure 3. The second-stage implant downloaded from the control server.

```c
lea   eax, [esp+10H] ; CSIDL_LOCAL_APPDATA\Strategic Planning Manager.doc
dec   eax
xor   ecx, ecx
call  dword ptr [esp+68H] ; URLDownloadToFileA
```

Figure 4. The decoy document downloaded from the control server.

4. Once both the second-stage implant and decoy document have been downloaded, the two payloads are executed:

- The second-stage implant is executed using the CreateProcessA() API.

- The decoy document is opened using the ShellExecuteA() with the “open” verb.
The Advanced Threat Research team discovered another PDF document (10mins.PDF) by the same author. It appears to be a smart phone–related questionnaire. This document was hosted on the same server as the two job-related malicious documents. The questionnaire appears to come from a big data analytics company that specializes in antifraud protection and financial compliance.

Figure 5. Control server strings constructed in the shellcode.

Figure 6. 10Mins.PDF
Rising Sun behavior

The Rising Sun implant is a fully functional modular backdoor that performs reconnaissance on the victim's network.

Imports

This implant starts by building its imports via dynamic API resolution: LoadLibrary()/GetProcAddress(). The library and API names are hardcoded as DWORD/WORD values in the implant and comprise a blob of bytes 0x147 bytes in size. This blob of data is decrypted using a simple single-byte XOR scheme with the key 0xC8.

This scheme used for building the Library and API names is a variant of the byte-chunk string-construction technique often used by Lazarus implants. The scheme typically involves:

- Hardcoded library and API names in the form of DWORD/WORD/byte chunks in the implant.
- Assigning variables with these hardcoded values during the execution of the implant.
- Constructing character arrays that consist of the library and API names to be resolved.
- Optionally these arrays may have to be decoded using something as simple as a single-byte XOR decoding scheme.
- Using LoadLibrary()/GetProcAddress() to now resolve the libraries and APIs using the constructed name arrays.

Configuration data

The configuration data used by the implant is encrypted using an RC4 stream algorithm. The implant decrypts the configuration data at runtime and for communicating with the control server. The addresses decrypted from the implant:

- http://137[dot]74.41.56/board.php

Figure 7. XOR-encoded library and API names in the implant.
Initial reconnaissance
The implant fetches the following data from the endpoint and exfiltrates it to the control server:

- Network adapter info
- Computer name
- User name
- IP address information
- Native system information
- OS product name from registry:
  SOFTWARE\MICROSOFT\Windows NT\CurrentVersion | ProductName

Additional configuration
The implant decrypts additional information during the reconnaissance process:

VboxHook.dll tmp SOFTWARE\Microsoft\Windows NT\CurrentVersion ProductName RUNAS; RUN; DLL; winsta0\default Kernel32.dll lnk SOFTWARE\Microsoft\Windows\CurrentVersion\Run C:\Program Files\Internet Explorer\iexplore.exe ntuser LOG8

This configuration data is not completely used by the implant, but there is a high possibility of other variants of the implant using the complete configuration data. The configuration data may have been copied from another implant family without scrubbing unused strings from the data.

Data encryption and exfiltration
The implant carries out data encryption and exfiltration using the following steps:

- Once the data has been gathered from the endpoint, the implant encrypts it using the RC4 stream encryption algorithm.
- After the data has been encrypted, the implant performs another layer of obfuscation of the data by Base64-encoding the RC4 encrypted data.
The implant performs an HTTP POST request to the control server:


As part of the request, the implant sends data in one of the following formats:

- boardID=<random_number>&page=<request_type>&wr_id=<encoded_time_stamp>&session_id=<RC4+base64 encoded data>
- bo_table=<random_number>&page=<request_type>&wr_id=<encoded_time_stamp>&session_id=<RC4+base64 encoded data>
- no=<random_number>&page=<request_type>&wr_id=<encoded_time_stamp>&session_id=<RC4+base64 encoded data>

The first variable in the HTTP data can be any of the following (randomly selected) values:

```javascript
var1_enum =
{
  "code="
  "no="
  "bo_table="
  "boardID="
  "pageKey="
  "structureid="
}
```

The `<request_type>` can be one of the following values:

```javascript
request_type =
{
  "free" //indicates initial reconnaissance data
  "query" //indicates a request to fetch the command ID from the control server
  "suggestion" //indicates request to fetch additional data from the control server
  "result" //indicates data obtained from a command’s execution
}
```
Implant capabilities
The implant carries 14 backdoor capabilities. It receives a command code (along with supporting data for the command) from the control server to execute a specific function. Unless otherwise specified, the implant sends the output of an executed command to the control server as an HTTP POST request with optional data in the form:

<var1_enum>=<random_number>&page=result&wr_id=<encoded_time_stamp>&session_id=<RC4 + Base64-encoded output of command>

**Capability #1: Execute commands**
Command code = 0x6D0017005500F7.

*Description*
The implant executes a command specified by the control server. The command is executed using cmd.exe:

```
cmd.exe /c "<command> > %temp%\AM<random>.tmp" 2>&1
```

The contents of the temporary file consist of the output of the command executed. The temp file is read, and the contents are subsequently sent to the control server. The temp file is then deleted from the endpoint. This capability also supports changing the current working directory for the implant and natively supports specific cd commands, without having to execute them through the shell.

Supported cd commands:
- `cd <directory_path>`
- `cd .`
- `cd \`

Figure 9. Command execution using the CreateProcess() function for cmd.exe.
**REPORT**

**Capability #2: Get drive information**

Command code = 0x0AD005F00A300C7.

**Description**

For every drive on the system, the implant gets the following information:

- Drive type
- Total number of bytes on disk
- Total number of free bytes on disk
- Name of a specified volume

**Figure 10.** Implant collecting drive information from the endpoint.

**Capability #3: Launch process from Windows binary**

Command code = 0x8300DA00C50092.

**Description**

- Launch a process from a binary specified by the filepath provided by the control server.
- Send a buffer (size=0x400) containing repeating 0x55 to the control server if successful or 0xAA if failed.

**Capability #4: Get processes information**

Command code = 0x62009A001C002B.

**Description**

Enumerate all processes currently running and record:

- Process name
- Process creation time
- Process exit time
- Process kernel mode time
- Process user mode time

**Figure 11.** Process related time stamps collected by the implant.
**REPORT**

**Capability #5: Terminate process**
Command Code = 0x57001D00E20060.

**Description**
- Terminate a process specified by the control server.
- The process can be specified using either:
  - Process name
  - Process ID
- Send a buffer (size=0x400) containing repeating 0x55 to the control server if successful or 0xAA if failed.

**Capability #6: Get file times**
Command code = 0x0A3001A006E00F8.

**Description**
- Find files based on a filename search string (for example, *.* or *.txt)
- For each file found, get the following times:
  - File creation time
  - Last access time (including read, write, or execute operations)

**Capability #7: Read file**
Command code = 0x98009C0034002D.

**Description**
- Read the contents of a file specified by the control server and exfiltrate the contents of the file.

---

Figure 12. Reading a file's contents.
**Capability #8: Clear process memory**
Command codes = 0x1800D500D5008F, 0x22001A00CA005E, 0x4D00D700AC0091, and 0x0C2009200D30028.

**Description**
- Clear a memory blob in the process by overwriting it with junk bytes.

**Capability #9: Write file to disk**
Command codes = 0x8D001F00FB0061 and 0x0B700550029003C.

**Description**
- Get a file path from the control server and create a file corresponding to the file path.
- Get content to be written to the file from the control server by sending an HTTP POST request with HTTP data in the format:
  `<var1_enum>=<random_number>&page=suggestion&wr_id=<encoded_time_stamp>&name=jquery2017<encoded_time_stamp>09.css`
- Send a buffer (size=0x400) containing repeating 0x55 to the control server if successful or 0xAA if failed.
**Capability #10: Delete file**
Command code = 0x78005D008B00C6.

*Description*
- Delete a file specified by the control server if it is not a directory.
- Send a buffer (size=0x400) containing repeating 0x55 to the control server if successful or 0xAA if failed.

**Capability #11: Get additional file information for files in a directory**
Command code = 0xD0057005B00C4.

*Description*
- If the file path specified is a directory, then enumerate all files in the directory and send to the control server, including:
  - File size
  - File attributes
  - File creation time
- If the file path is not a directory (regular file), then the implant fetches a DWORD pointed to by offset 0x3C in the file.
  - This parses MZ (executable) files, in particular where the location of IMAGE_NT_HEADERS is specified at offset 0x3C.
  - The implant reads the compile date of the MZ files by reading the time stamp (DWORD) at IMAGE_NT_SIGNATURE + 0x08.

```assembly
... (assembly code for delete file capability) ...
... (assembly code for get file information) ...
```

Figure 14. Implant reading the compilation timestamp of a specified MZ (Windows executable) file.
**Capability #12: Connect to an IP address**  
Command code = 0x0B700150099005C.

**Description**
- Tests a connection to a specified network IP address over a specified port number.
- The implant only attempts to connect to the network address.
- Based on the connection attempt, sends a buffer (size=0x400) containing repeating 0x55 to the control server if successful or 0xAA if failed.

**Capability #13: Change file attributes**  
Command code = 0x0EC001700B2005D.

**Description**
- Modifies the following file information based on the content specified by the control server:
  - File attributes (hidden, system, etc.)
  - If the file is an MZ, then the compile time stamp of the file is also modified in the PE header.
  - If the file is not an MZ, then the implant can move the file to a different location after modifying its attributes.

**Capability #14: Variant of change file attributes (capability #13)**  
Command code = 0x0E200D2007C008E.

**Description**
- Changes file attributes (hidden, system, etc.) and moves the file to a different location.
Attribution

Attributing an attack to any threat group is often riddled with challenges, including potential “false flag” operations by other threat actors. Technical evidence alone is not sufficient to attribute this activity with high confidence. However, based on our analysis, this operation shares multiple striking similarities with other the Lazarus Group attacks; thus we present them for further analysis. Although these similarities point to Lazarus, we also must consider the possibility of false flags.

- The malicious Word documents were created in a Korean-language environment. (The code page is in Korean.)
- The implant uses a variant of the dynamic API resolution technique we have observed with multiple Lazarus implants.
- The operation is very similar to a Lazarus operation from 2017 that targeted the US defense and energy sectors. The techniques, tactics, and procedures match those in this previous operation.
- Rising Sun is an evolution of the Lazarus backdoor Duuzer, which was circulated in 2015 and targeted South Korea.

Comparing Rising Sun to Duuzer

The Advanced Threat Research team found that Rising Sun shares code with the Duuzer implant family, which was identified by the security community as belonging to Lazarus. We compared the following samples and detail their similarities and differences.

Samples used for comparison:
- Rising Sun: f3bd9e1c01f2145eb475a98c87f94a25
- Duuzer: 73471f41319468ab207b8d5b33b0b4be

Configuration data

Although the decryption schemes used by Rising Sun and Duuzer are different, both implants use similar configuration data used to drive their reconnaissance capabilities:

<table>
<thead>
<tr>
<th>Configuration data decoded by Duuzer</th>
<th>Configuration data decoded by Rising Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>VboxHook.dll</td>
<td>VboxHook.dll</td>
</tr>
<tr>
<td>tmp SOFTWARE\</td>
<td>tmp SOFTWARE\</td>
</tr>
<tr>
<td>Microsoft\Windows NT\CurrentVersion</td>
<td>Microsoft\Windows NT\CurrentVersion</td>
</tr>
<tr>
<td>ProductName RUNAS; RUN; DLL; winsta0\default Kernel32. dll lnk SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td>ProductName RUNAS; RUN; DLL; winsta0\default Kernel32. dll lnk SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
</tr>
<tr>
<td>perfd000 dat</td>
<td>perfd000 dat</td>
</tr>
</tbody>
</table>
**Library/API resolution**

Both implants use the same technique of constructing and decoding library and API names for dynamic API resolution. We explained this technique (a variant of byte-chunk library/API name construction) in a preceding section. Although the encoded data blob consisting of the library/API strings in Duuzer is 0x181 bytes in size and is decoded using 0x30 as the XOR key, the encoded data blob in Rising Sun is 0x147 bytes in size and is decoded using 0xC8 as the XOR key.

**Library names**

Another similarity between the two implant families is that some of the decoded library names consist of randomized characters. For example, Duuzer capitalizes random characters of the following library name:

- uSEr32.dL

Rising Sun does something similar in these library names:

- vErSion.dll
- advap32.dLL
Similarities between Rising Sun and Duuzer

The implant families are a direct match in several capabilities as well as in the code structure and API use to implement these capabilities. The following capabilities are a direct match:

**Initial reconnaissance (gather preliminary system info)**

Both implants capture the same information from the endpoint during their initial reconnaissance. The order of information and the API/code signatures are an exact match.

Information captured by both implants:
- Network adapter info
- Computer name
- User name
- IP address information
- Native system information
- OS product name from registry:
  SOFTWARE\MICROSOFT\Windows\NT\CurrentVersion | ProductName

Figure 18. Similarities in Duuzer (at left) and Rising Sun in their preliminary reconnaissance code.
Capability #1: Execute commands
Both implants can execute commands using cmd.exe with the output redirected to a temp file on the endpoint:
- `cmd.exe /c "<command> > <%temp%><Temp_File_Prefix><random>.tmp" 2>&1`

Both implants support changing directories natively, without having to execute cd commands through the shell. Supported cd commands:
- `cd <directory_path>
- `cd`
- `cd\`
**Capability #2: Get drive information**
Both implants gather the same data using similar code signatures:
- Drive type
- Total number of bytes on disk
- Total number of free bytes on disk
- Name of a specified volume

**Figure 21.** Similar code signature and drive information gathered by Duuzer (at left) and Rising Sun.

**Capability #3: Launch a process from Windows binary**
Both implants use the same API and flags to launch new processes on the endpoint.

**Capability #4: Get processes information**
Both implants exfiltrate the exact same process information:
- Process name
- Process creation time
- Process exit time
- Process kernel mode time
- Process user mode time

**Figure 22.** Duuzer’s (at left) and Rising Sun’s process time information gathering code signatures.

**Capability #5: Terminate process**
Both implants support the capability to terminate a process running on the system based on either the:
- Process Name
- Process ID
**Capability #6: Get file times**
Both implants implement the same capabilities:
- Find files based on a filename search string (for example, *.* or *.txt)
- For each file found, get the following times:
  - File creation time
  - Last access time (including read, write, or execute operations)

**Capability #7: Read a file**
Both implants can read the contents of a file specified by the control server and exfiltrate the contents of the file.

**Capability #8: Clear process memory**
There are no significant similarities between the two implants.

**Capability #9: Write a file to disk**
Both implants can write content served by the control server to a file on disk (with file path also specified by the control server) using the same sequence of actions:
- Get a file path from the control server and create a file corresponding to the file path.
- Fetch content to be written to the file from the control server using the implant-specific communication mechanism.
- Once the content has been written to the file path, send either a success or a failure response to the control server.

**Capability #10: Delete file**
Both implants can delete a file specified by the control server if it is not a directory.
**Capability #11: Get additional file information for files in a directory**

Both implants have the same capability to get file information for files in a specified directory, including the following data:

- File attributes
- File size
- File creation time
- Last access time
- File write time
- MZ compile time

Figure 25. Similar code between Duuzer (at left) and Rising Sun for reading the MZ's compile time stamp.

**Capability #12: Connect to an IP address**

Both implants test connections to a specified IP address using the same actions, APIs, and code signatures:

- Test a connection to a specified network IP address over a specified port number.
- Only attempt to connect to the network address.
- Based on the connection attempt, send either a success or a failure response to the control server.

**Capability #13: Change file attributes**

Both implants can modify the same file attributes:

- File attributes (hidden, system, etc.)
- If the file is an MZ, then the compile time stamp of the file is also modified in the PE header.

Figure 26. Similar code used by both Duuzer (at left) and Rising Sun to modify file attributes and times.

**Capability #14: Variant of change file attributes**

Both implants can change file attributes and move the file to a different location.
Differences between Rising Sun and Duuzer

There are some notable differences in implementation between the two families.

**Communication mechanism:** Duuzer uses a simple socket-based communication mechanism to send and receive data from its control server. Rising Sun uses an HTTP-based mechanism. This difference may be an enhancement by the attackers because masking the control server communication is more effective against detection by the human eye and network intrusion prevention systems. High-level differences in the communication mechanisms:

- Communication schemes (native socket vs. HTTP).
- Command codes used to indicate a specific capability
- Return codes/data indicating success or failure of a command’s execution

**Encoding schemes:** Apart from the library and API name construction and decoding, the encryption schemes used in the implant are quite different. While Duuzer uses a custom XOR scheme to decode its configuration data, Rising Sun uses the RC4 stream algorithm.

Conclusion

Our discovery of a new, high-function implant is another example of how targeted attacks attempt to gain intelligence. The malware moves in several steps. The initial attack vector is a document that contains a weaponized macro to download the next stage, which runs in memory and gathers intelligence. The victim’s data is sent to a control server for monitoring by the actors, who then determine the next steps.

We have not previously observed this implant. Based on our telemetry, we discovered that multiple victims from different industry sectors around the world have reported these indicators. Operation Sharpshooter’s similarities to Lazarus Group malware are striking, but that does not ensure attribution. Was this attack just a first-stage reconnaissance operation, or will there be more? We will continue to monitor this campaign and will report further when we or others in the security industry receive more information. The McAfee Advanced Threat Research team encourages our peers to share their insights and attribution of who is responsible for Operation Sharpshooter.
Indicators of Compromise

MITRE ATT&CK™ techniques
- Account discovery
- File and directory discovery
- Process discovery
- System network configuration discovery
- System information discovery
- System network connections discovery
- System time discovery
- Automated exfiltration
- Data encrypted
- Exfiltration over command and control channel
- Commonly used port
- Process injection

Hashes
- 8106a30bd35526bded384627d8eebce15da35d17
- 66776c50bcc79bbcecdbe99960e6ee39c8a31181
- 668b0df94c6d12ae86711ce24ce79dbe0ee2d463
- 9b0f22e129c73ce4c21be4122182f6dcbc351c95
- 31e79093d452426247a56ca0eff860b0ecc86009

Control servers
- 34.214.99.20/view_style.php
- 137.74.41.56/board.php
- kingkoil.com.sg/board.php

Document URLs
- hxxp://208.117.44.112/document/Strategic Planning Manager.doc
- hxxp://208.117.44.112/document/Business Intelligence Administrator.doc
- hxxp://www.dropbox.com/s/2shp23ogs113hnd/Customer Service Representative.doc?dl=1

McAfee detection
- RDN/Generic Downloader.x
- Rising-Sun
- Rising-Sun-DOC
About McAfee

McAfee is the device-to-cloud cybersecurity company. Inspired by the power of working together, McAfee creates business and consumer solutions that make our world a safer place. By building solutions that work with other companies’ products, McAfee helps businesses orchestrate cyber environments that are truly integrated, where protection, detection, and correction of threats happen simultaneously and collaboratively. By protecting consumers across all their devices, McAfee secures their digital lifestyle at home and away. By working with other security players, McAfee is leading the effort to unite against cybercriminals for the benefit of all.

www.mcafee.com

About McAfee Labs and Advanced Threat Research

McAfee Labs, led by McAfee Advanced Threat Research, is one of the world’s leading sources for threat research, threat intelligence, and cybersecurity thought leadership. With data from millions of sensors across key threats vectors—file, web, message, and network—McAfee Labs and McAfee Advanced Threat Research deliver real-time threat intelligence, critical analysis, and expert thinking to improve protection and reduce risks.