An In-Depth Analysis of Samsam Ransomware and BOSS SPIDER

crowdstrike.com/blog/an-in-depth-analysis-of-samsam-ransomware-and-boss-spider/

May 21, 2018

May 21, 2018

Karan Sood From The Front Lines

Introduction

This analysis provides an in-depth view of the Samsam ransomware, which is developed and operated by the actor tracked by CrowdStrike® Falcon Intelligence™ as BOSS SPIDER. The infection chain and the execution flow vary according to the variant of the malware, both of which are detailed in this writeup. The malware uses external tools such as batch scripts, Mimikatz, and Sysinternals utilities, including PsExec and Sdelete, to aid in the propagation and cleanup of the ransomware. In some cases, the ransomware is delivered in an encrypted manner alongside a runner file, which is used to load and execute the malware in memory. In addition, the malware has anti-forensics capabilities that make it challenging to recover the ransomware payload from an infected system. Despite the steps taken by the malware to evade detection, Falcon Prevent can detect and prevent the family before it ever encrypts any files on the system.

Infection Process

Numerous variations of delivery methods for this ransomware family have been seen in the wild. Earlier variations of the ransomware involved gathering credentials from Active Directory via a credential dumper such as Mimikatz, generating public RSA keys for each user on the list and deploying the payload along with the following files:

- PSExec: A legitimate utility that is part of Sysinternals suite used to execute applications on remote systems
- Backup-Delete Helper File: This file is responsible for enumerating all the drives connected to the victim machine. A detailed analysis is provided in the Backup–Delete Helper File subsection
- A list of compromised accounts
- A folder containing unique RSA public keys for each account
- Four batch scripts that are responsible for copying the files to each victim computer and loading the payload
    - First file: copies the samsam payload and the corresponding public key to %WINDIR%\system32; issues the command "**vssadmin delete shadows /all /quiet**"
    - Second file: copies the backup-delete helper file to C:\Windows
    - Third file: utilizes psexec to remotely execute the delete helper file
    - Fourth file: utilizes psexec to remotely execute the payload; provides the corresponding public key as an argument to the payload

As seen above, the public key and the backup deletion commands are external to the payload itself. In other variants of Samsam, both the key and the backup deletions command have been embedded within the payload. In such cases, the delivery mechanism differs significantly from the one described above. Rather than deploying the payload with numerous files, the payload is instead encrypted and deployed with a runner file and a batch script as described below:

- Runner file: responsible for decrypting and loading the payload; a detailed analysis is provided in the Runner File subsection.
- Batch script: loads the runner file and provides the following arguments to it *(It should be noted that the number of arguments can also differ depending on the variant of the ransomware. Up to four arguments have been observed so far)*.
    - <password>: used to decrypt the encrypted payload
    - <path>: used by the decrypted payload to drop additional files
    - <totalprice>: used by the decrypted payload to dynamically generate the splash screen
    - <priceperhost>: used by the decrypted payload to dynamically generate the splash screen

In other cases, standalone Samsam payloads that do not require any arguments or helper files have also been seen.

As mentioned, there are a variety of ways in which the payload can be delivered and loaded onto a system. Subsequent sections will describe the payload and some of the previously mentioned helper files in more detail.

Technical Analysis

Backup-Delete Helper File

This file is responsible for traversing the file system on a victim computer and targeting specific file extensions associated with data backup. The extensions are provided in **Appendix A: Targeted Backup Extensions**. This program's goal is to ensure that once the files on the system have been encrypted, no backups can be used for file restoration. *Please note that if any shared drives are connected, the helper file will also delete the corresponding files in those folders.*

Once loaded by the batch script, this file recursively enumerates each directory in each drive connected to the victim machine. If a directory name contains the string "**backup,**" the program will enumerate all the files within that directory and perform the following actions:

- Set the file attributes to FILE_ATTRIBUTE_NORMAL. This ensures that the file can be deleted without throwing an exception.
- Delete each file once the file attribute is set.
- Once all the files have been deleted, delete the directory itself.

For all the other directories, once a target file is found, the program ensures that it can delete the file by performing the following actions:

- Checks that the file is not locked. In other words, makes sure that there isn't a handle to the file being accessed by a process and that the file can be opened in READ mode. If the check passes, delete the file.
- If the above check fails, the program kills the process associated with the file.
    - It invokes tasklist.exe to generate a list of running processes on the system by issuing the following command: **tasklist /v /fo csv**.
    - It enumerates the list to find a process with the same name as the file in question. If found, it invokes taskkill.exe to kill the process by issuing the following command: **taskkill /f /pid <process ID of the target process>.**
- Once the process has been killed, it deletes the corresponding file.

The following is a snippet of the output of the tasklist command:

Figure 1: Output of the tasklist command

Runner File

The runner file is responsible for decrypting and loading the payload in memory. This file requires a password as an argument, which is used in the decryption process. Once loaded by the batch script, the runner looks for a file with the extension "**.stubbin**" within the current working directory. This file is the actual encrypted payload and once it is found, the runner reads the contents in memory and deletes the encrypted .stubbin file from the disk immediately. The fact that the payload is decrypted and loaded entirely in memory makes it hard to recover the decrypted ransomware payload forensically.

Once the content is read, the runner decrypts it using the Rijndael algorithm. The password and a salt are provided to the algorithm to generate the decryption key and an IV. The salt value is "Ivan Medvedev", which has been seen in numerous binaries (benign and malicious) employing this algorithm. With these two inputs, the runner file generates a 32 byte key, and a 16 byte IV, which are then subsequently used to decrypt the payload. It is important to note that any additional arguments passed down from the batch file are then passed to the decrypted payload. The runner file is then responsible for invoking the entry point of the payload; thereby executing it in memory.

Payload

Numerous variants of the payload were analyzed, and one of the main differences among them was whether the RSA public key was embedded in the payload or not. As mentioned earlier, the batch script responsible for loading the payload passes the public key as an argument to it, which is then read to a variable and used later in the execution process. This section will explain in detail the steps taken by the ransomware to encrypt the files on the victim machine.

Resource Extraction

Upon being loaded, Samsam first parses its own resource section and extracts the resource names. For each name, it ensures that the name has the extension ".exe", and checks to see if a file with the same name as the resource name exists in the current directory. If so, it

deletes the file from disk, reads the contents of the resource section in chunks of 4096 bytes and writes it to disk in the current directory. So far, up to two different files have been seen as a result of this extraction: selfdel.exe and del.exe. *Please note that this particular activity of dropping two files in the current directory with static names can be used for detection purposes early on in the kill chain.*

Once the files have been written to disk, the payload initiates a new thread to execute selfdel.exe, which is explained in further detail in the Cleanup section. Next, the payload will recursively enumerate all the directories in each drive connected to the victim machine.

File System Enumeration

The payload ensures that the following directories are skipped when enumerating files:

- C:\Windows
- Reference Assemblies\Microsoft
- Recycle.bin

This is to ensure a smooth execution flow. Since system files reside in C:\Windows, and files necessary for the .NET framework are in Reference Assemblies\Microsoft, it is imperative that those files are left intact. One possible reason for skipping files within the Recycle.bin directory might be that this directory is usually among the first ones to be enumerated by ransomware families. A few security solutions depend on that behavior to detect and prevent the ransomware process early on in the execution flow. By skipping that directory, Samsam can sidestep that mitigation and continue to encrypt the files.

The list of file extensions targeted by Samsam are listed in **Appendix A: Targeted Extensions**. Once a file matching one of the extensions is found, the payload calls the encryption subroutine immediately, if the file size is less than or equal to 100MB (104857600 bytes). However, if the file size is greater, the following actions take place:

- If the file size is greater than 100MB and less or equal to 250MB, it appends the file's full path to the list **mylist250.**
- If the file size is greater than 250MB and less or equal to 500MB, it appends the file's full path to the list **mylist500.**
- If the file size is greater than 500MB and less or equal to 1GB, it appends the file's full path to to the list **mylist1000.**
- If the file size is greater than 1GB, it appends the file's full path to the list **mylistbig.**

The payload encrypts the files in those lists once all the targeted files less than or equal to 100MB in size are encrypted first. This is likely done to ensure that as many files as possible are encrypted in case the ransomware process ends prematurely. It should be noted that the encryption subroutine is called on those lists in order of the file size; files in **mylist250** are encrypted first, followed by **mylist500** and so on. Once the files in each list are encrypted, the payload clears the list in memory.

In other variants of Samsam, this behavior of categorizing files per size is absent. Those variants utilize just one list to compile a list of all files within all the drives connected to the victim machine. Next, the file encryption subroutine is invoked to encrypt the target files.

File Encryption Wrapper

For each file, the payload performs the following checks:

- The length of the current file is less than the available free space in the drive. If the check fails, the payload moves on to the next file. This is to ensure that there is enough free space to write the encrypted file to disk.
- The length of the current file is greater than 0 bytes.
- The public key variable is not NULL. Without the public key, no encryption would take place.

If any of the above checks fail, the payload moves on to the next file.

Next, there are two observed variations in the manner in which encryption takes place; these depend on the variant of Samsam running on the system.

Variation A

For each file, the subroutine first checks if a file with the name <target filename>.<encrypted extension> already exists in the current directory. The encrypted extensions value has consistently been "**.encryptedRSA**". If so, the following steps are taken:

- Checks the length of the .encryptedRSA file. If this file size is greater than the target file, the subroutine deletes the target file and moves on to the next file. The program assumes that the file has already been encrypted with Samsam and therefore deletes the original target file.
- If the .encryptedRSA file size is less than or equal to the target file, the subroutine deletes the .encryptedRSA file. Since the actual encryption will cause the resultant file size to be greater than the original file, the .encryptedRSA file size must be greater than the target file. The program assumes that the .encryptedRSA file is not the actual encrypted file and deletes it from disk. Once the .encryptedRSA file is deleted, the target file is encrypted.

Variation B

If the subroutine finds a file with the name <target filename>.encryptedRSA, it immediately skips the target file. There are no checks on the length of this file, nor is there any attempt to delete the target file. It is important to note that the subroutine moves onto the next file, leaving the current target file intact. This is a significant oversight on the malware author's part since it implies that if there are files of any length with the extension .encryptedRSA for each file on the system, no files would get encrypted. In fact the ransomware would simply skip over every file, assuming that the files are already encrypted. This was tested and confirmed during analysis.

File Encryption

Samsam utilizes the AES standard to encrypt the files. For each file, it generates a random 64 byte signaturekey, a 16 byte key, and a 16 byte IV. It also creates an empty file <target filename>.encryptedRSA to which it writes 3072 NULL bytes. This acts as a placeholder for the encrypted file header that is generated later in the execution flow. The payload then determines if there are current processes or services with an open handle to the target file by utilizing Restart Manager APIs. The following explains the steps in detail:

- Start a new session manager by invoking **RmStartSession**.This provides a session handle to the new session.
- The handle is used by **RmRegisterResources** to register the target file as a resource to the new session.
- The handle is then used by **RmGetList** to get a list of processes that are currently utilizing the resource in question (target file).
- The session manager is ended with **RmEndSession**.
- The process ID of each process locking the file is appended to a file, which is then passed to a subroutine responsible for killing those processes.

Once the target file handle is freed from any processes, the encryption subroutine reads its contents in a memory buffer in chunks of 10KB (10240 bytes) at a time. Next, the subroutine uses AES in CBC mode to encrypt the contents in the buffer, which are then written to the .encryptedRSA file starting from the 3073rd byte position (after the header placeholder). Once the entire file content has been encrypted and written to the new file, the file header is generated.

File Header

The randomly generated signaturekey is used to generate a Hash-based Message Authentication Code (HMAC) hash of the encrypted content. This value, inArray, is then encoded with base64, and written to the file header. The following is the structure of the file header:

- <MtAeSKeYForFile>
  <Key>: contains the randomly generated 16 byte key encrypted with the RSA public key, and then encoded using base64. </Key>
- <IV>: contains the randomly generated 16 byte IV encrypted and encoded in the same manner. </IV>
- <Value>: contains the aforementioned inArray </Value>
- <EncryptedKey>: contains the randomly generated 64 byte signature key encrypted and encoded in the same manner. </Encrypted>
- <OriginalFileLength>:contains the file size of the original target file. </OriginalFileLength>
- </MtAeSKeYForFile>

The corresponding RSA private key would help decrypt each value in the file header that would, in turn, decrypt the file contents. The following is an example of an actual file header of a file encrypted by Samsam:

```
<MtAeSKeYForFile>

<Key>
YuepZM/SPIpW/5ONPfFSDo2TkObsKawVR+TqpCi5WDhktZkuYyWr4kAkEttZaZ7q+aTidm5Xfp6fQiOAxcwUdHE
O3mWXeVUuBIlbnM/B4jhSDNAL1eUCyRBO2kaHsJWBZzTA3YyX+N+ETSoMd09fL5Qr8Ez2IxGmAcD5jprG+uLTHp
+3TZPS3I/A4xLl7Uh3vBk0DGEc76ek2zBr00VUaP9QzVcn0/qegRN1EZ+/sd+mB2aDsiObylYlU+oUIShJ02BZW
HKr27A+PWYU+4D9r7rjzSVfs8XtD0nx19LJNJm1Lo7pOAwyW2XDwbum8563GrEPH67eoCS7JzHIWsuGIQ==
</Key>

<IV>
EVk6kpwoJltzEGi1IDwb8/kAZhPrtB/1Vrt+dvRinlWz6YppW9RaKkRyj/hT/Mq9gcfZMdqlGPR8OpQIiZCh+iu
oR20+5vYx+78Vw9BS50D4RsfAQU16VBhQOHtcImiVWtv4Md2wv4Ng3ywg5NAhkRg+c1RutQqhBkMBQpeCPvUcA6
iG17WQfA2JqsAs0Rm5pNRsCnxvMqrpcFaxBmZDuQEc35nkW/p3SmZh50Bn7prvyrmeRfPEpdkstdGbtVwbnNZ5K
K5pW5Ck5hXHw20Rv38pwl3MVCO3MzqB1oTmhJaptI/lwjA1QRVLa+01VK996xtQyirELCEtwDJRXnZRFQ==
</IV>

<Value>+XhRYojeg+Qfcu9Ac7xl66nL6pC9go9+4aU/6+PBpgU=</Value>

<EncryptedKey>
P9rriIuPoI+BxAtjiQaHx3n3bIZIlFL4Taa0ycONfOBa0S629zIgIXizlWDgBo+qccCR8AB8nbQfGpS0x2bY7aX
udcUY6pVwikVGy9wKI8wucvUwhAuYqRARjdDqnD64CM0Pxhv5SU934BV+UdTyrEivm/Yq0xlnabf1mscOOCDbN8
6EYA04SQteULP5I5AcCW8E3b85jsxGVtoEbEI2fjhXTkz8yAtgWng2+IWHhBrsiHiAxY65LtSLdsbQBu5dS8Dhw
ttO4Y4hsZqV6ZnNpJk7NsZe5fp4geOutY/pBhj5L5gZPnb6DhfcTcBkhTaHmohd8joor2x7YCfhXD02RA==
</EncryptedKey>

<OriginalFileLength>8192</OriginalFileLength>

</MtAeSKeYForFile>"
```

Figure 2: File header of a file encrypted with Samsam

Once the file is encrypted, the original target file is deleted from disk.

Post Encryption

After each file has been encrypted, the payload will drop a text file named **HELP_DECRYPT_YOUR_FILES** in the current directory. Figure 3 shows contents of the file.

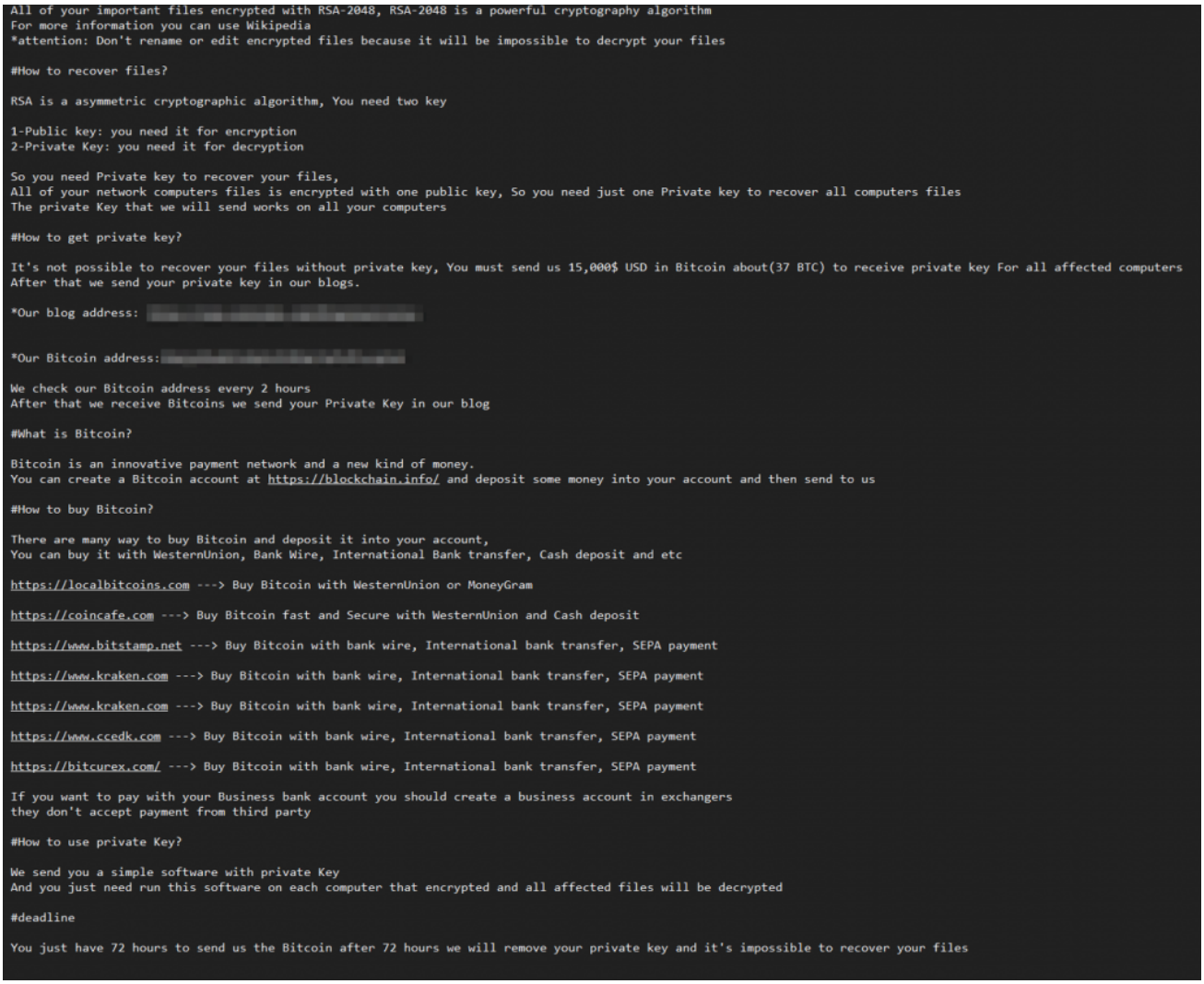


Figure 3: Ransomware note

In some instances, HELP_DECRYPT_YOUR_FILES.html with the same contents is dropped in the current directory. An odd behavior of every variant of Samsam analyzed is that the ransomware note is dropped for every file encrypted. Therefore, in directories where multiple files have been encrypted, multiple copies of the same ransomware note are dropped.

Cleanup

As mentioned earlier, the payload parses its resource section to extract two executables which are then written to the current directory. The following are the properties of both files:

File: selfdel.exe
Size: 5632
MD5: 710A45E007502B8F42A27EE05DCD2FBA
SHA256:

32445C921079AA3E26A376D70EF6550BAFEB1F6B0B7037EF152553BB5DAD116F
Compiled: Wed, Dec 2 2015, 22:24:42 – 32 Bit .NET AnyCPU EXE
Version: 1.0.0.0

File: del.exe
Size: 155736
MD5: E189B5CE11618BB7880E9B09D53A588F
SHA256:
97D27E1225B472A63C88AC9CFB813019B72598B9DD2D70FE93F324F7D034FB95
Compiled: Sat, Jan 14 2012, 23:06:53 – 32 Bit EXE
Version: 1.61
Signature: Valid
Subject: Microsoft Corporation
Issuer: Microsoft Code Signing PCA
InternalName: sdelete
ProductName: Sysinternal Sdelete

The file del.exe is a legitimate Sysinternals utility that is used to delete files from disk. The file selfdel.exe is invoked in a new thread by the payload. It first ensures that a process named samsa.exe is currently executing. If not, it immediately exits. Next, the file waits for 3 seconds, and then issues the following command: **del.exe -p 16 samsam.exe**. This invokes the legitimate utility del.exe, which is used to delete samsam.exe from disk. However, deleting a file simply marks it as deleted in the MFT table, which frees up the clusters that were allocated to the file. The file contents would still be present in those clusters. To overcome this, the utility actually overwrites the clusters; the **-p 16** argument ensures that the clusters are overwritten 16 times, making forensic recovery of the payload impossible. This technique only applies to standalone payloads, and not to the variants that involve a runner file to decrypt and execute the payload in memory.

Once the payload is deleted, selfdel.exe sleeps for 30 seconds, and then deletes **del.exe** from disk; however, it does not delete itself.

How CrowdStrike® Falcon Prevent™ Stops Samsam

Samsam is unique in terms of how it is delivered onto a system. The gathering of user credentials to generate unique RSA public keys ensures that an organization would have to pay for each infected user to decrypt the files. In addition, the use of cleanup files for standalone executables, and the fact that other variants run entirely in memory, makes it extremely difficult to forensically collect the payload from disk or memory. Despite these challenges, Falcon Prevent™ next-gen AV is able to detect and prevent the malware before it can perform any file encryption as seen below:

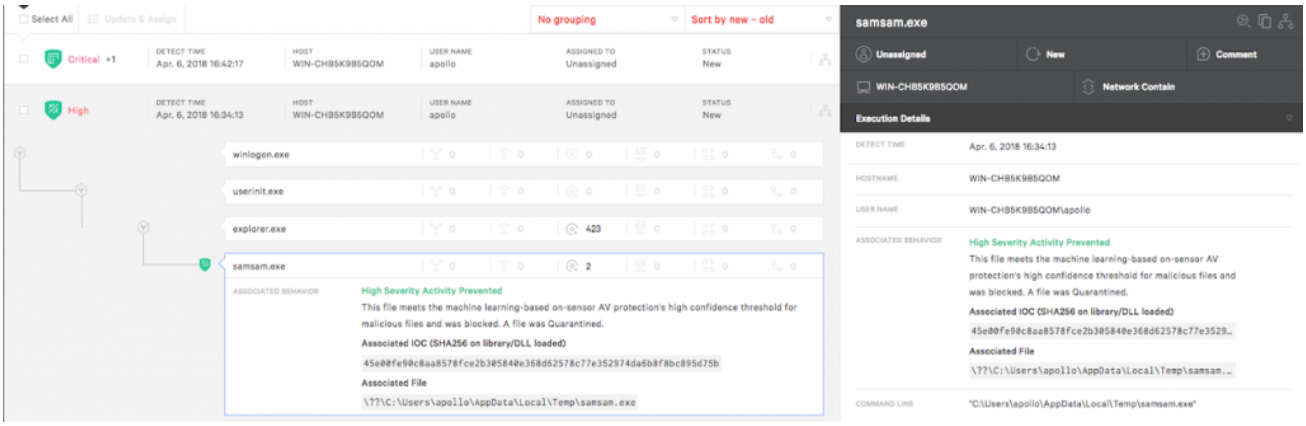


Figure 4: Samsam prevented by Falcon Prevent

Falcon Prevent is also able to detect and prevent the dropped file selfdel.exe from executing once loaded by the payload as shown below:

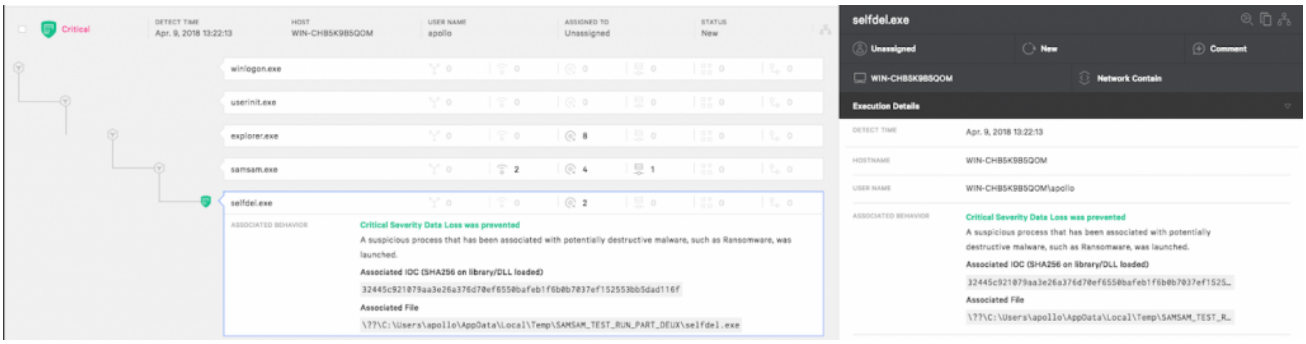


Figure 5: Selfdel.exe prevented by Falcon Prevent

These preventions are based on the behavioral patterns displayed by Samsam in conjunction with CrowdStrike's machine learning algorithm.

Appendix A

Targeted Backup Extensions

.abk, .ac, .back, .backup, .backupdb, .bak, .bb, .bk, .bkc, .bke, .bkf, .bkn, .bkp, .bpp, .bup, .cvt, .dbk, .dtb, .fb, .fbw, .fkc, .jou, .mbk, .old, .rpb, .sav, .sbk, .sik, .spf, .spi, .swp, .tbk, .tib, .tjl, .umb, .vbk, .vib, .vmdk, .vrb, .wbk

Targeted Extensions

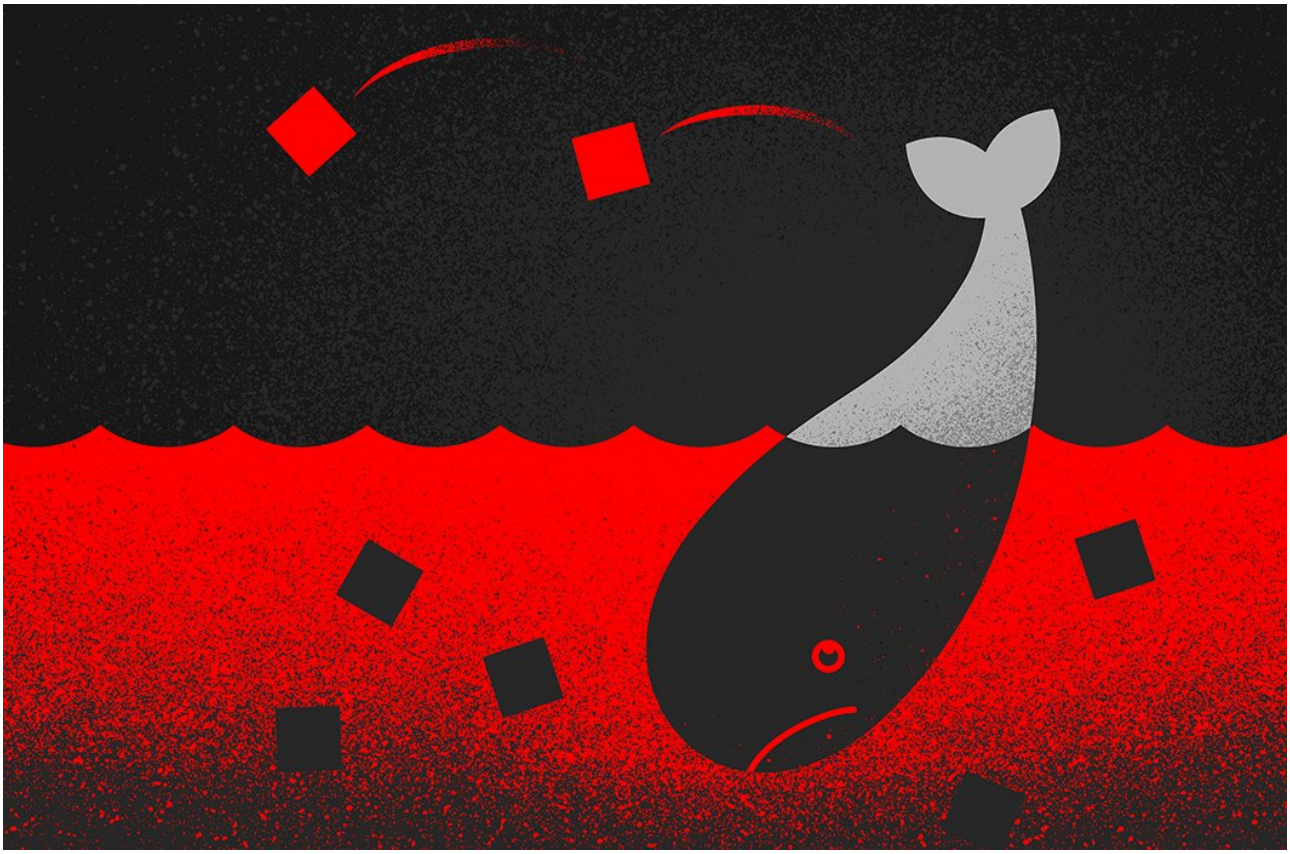
.jin, .xls, .xlsx, .pdf, .doc, .docx, .ppt, .pptx, .log, .txt, .gif, .png, .conf, .data, .dat, .dwg, .asp, .aspx, .html, .tif, .htm, .php, .jpg, .jsp, .js, .cnf, .cs, .vb, .vbs, .mdb, .mdf, .bak, .bkf, .java, .jar, .war, .pem, .pfx, .rtf, .pst, .dbx, .mp3, .mp4, .mpg, .bin, .nvram, .vmdk, .vmsd, .vmx, .vmxf, .vmsn, .vmem, .gz, .3dm, .3ds, .zip, .rar, .3fr, .3g2, .3gp, .3pr, .7z, .ab4, .accdb, .accde, .accdr, .accdt, .ach, .acr, .act, .adb, .ads, .agdl, .ai, .ait, .al, .apj, .arw, .asf, .asm, .asx, .avi, .awg, .back, .backup, .backupdb, .pbl, .bank, .bay, .bdb, .bgt, .bik, .bkp, .blend, .bpw, .c,

.cdf, .cab, .chm, .cdr, .cdr3, .cdr4, .cdr5, .cdr6, .cdrw, .cdx, .ce1, .ce2, .cer, .cfp, .cgm, .cib, .class, .cls, .cmt, .cpi, .cpp, .cr2, .craw, .crt, .crw, .csh, .csl, .csv, .dac, .db, .db3, .dbf, .db-journal, .dc2, .dcr, .dcs, .ddd, .ddoc, .ddrw, .dds, .der, .des, .design, .dgc, .djvu, .dng, .dot, .docm, .dotm, .dotx, .drf, .drw, .dtd, .dxb, .dxf, .jse, .dxg, .eml, .eps, .erbsql, .erf, .exf, .fdb, .ffd, .fff, .fh, .fmb, .fhd, .fla, .flac, .flv, .fpx, .fxg, .gray, .grey, .gry, .h, .hbk, .hpp, .ibank, .ibd, .ibz, .idx, .iif, .iiq, .incpas, .indd, .jpe, .jpeg, .kc2, .kdbx, .kdc, .key, .kpdx, .lua, .m, .m4v, .max, .mdc, .mef, .mfw, .mmw, .moneywell, .mos, .mov, .mrw, .msg, .myd, .nd, .ndd, .nef, .nk2, .nop, .nrw, .ns2, .ns3, .ns4, .nsd, .nsf, .nsg, .nsh, .nwb, .nx2, .nxl, .nyf, .oab, .obj, .odb, .odc, .odf, .odg, .odm, .odp, .ods, .odt, .oil, .orf, .ost, .otg, .oth, .otp, .ots, .ott, .p12, .p7b, .p7c, .pab, .pages, .pas, .pat, .pcd, .pct, .pdb, .pdd, .pef, .pl, .plc, .pot, .potm, .potx, .ppam, .pps, .ppsm, .ppsx, .pptm, .prf, .ps, .psafe3, .psd, .pspimage, .ptx, .py, .qba, .qbb, .qbm, .qbr, .qbw, .qbx, .qby, .r3d, .raf, .rat, .raw, .rdb, .rm, .rw2, .rwl, .rwz, .s3db, .sas7bdat, .say, .sd0, .sda, .sdf, .sldm, .sldx, .sql, .sqlite, .sqlite3, .sqlitedb, .sr2, .srf, .srt, .srw, .st4, .st5, .st6, .st7, .st8, .std, .sti, .stw, .stx, .svg, .swf, .sxc, .sxd, .sxg, .sxi, .sxi, .sxm, .sxw, .tex, .tga, .thm, .tlg, .vob, .wallet, .wav, .wb2, .wmv, .wpd, .wps, .x11, .x3f, .xis, .xla, .xlam, .xlk, .xlm, .xlr, .xlsb, .xlsm, .xlt, .xltm, .xltx, .xlw, .ycbcra, .yuv

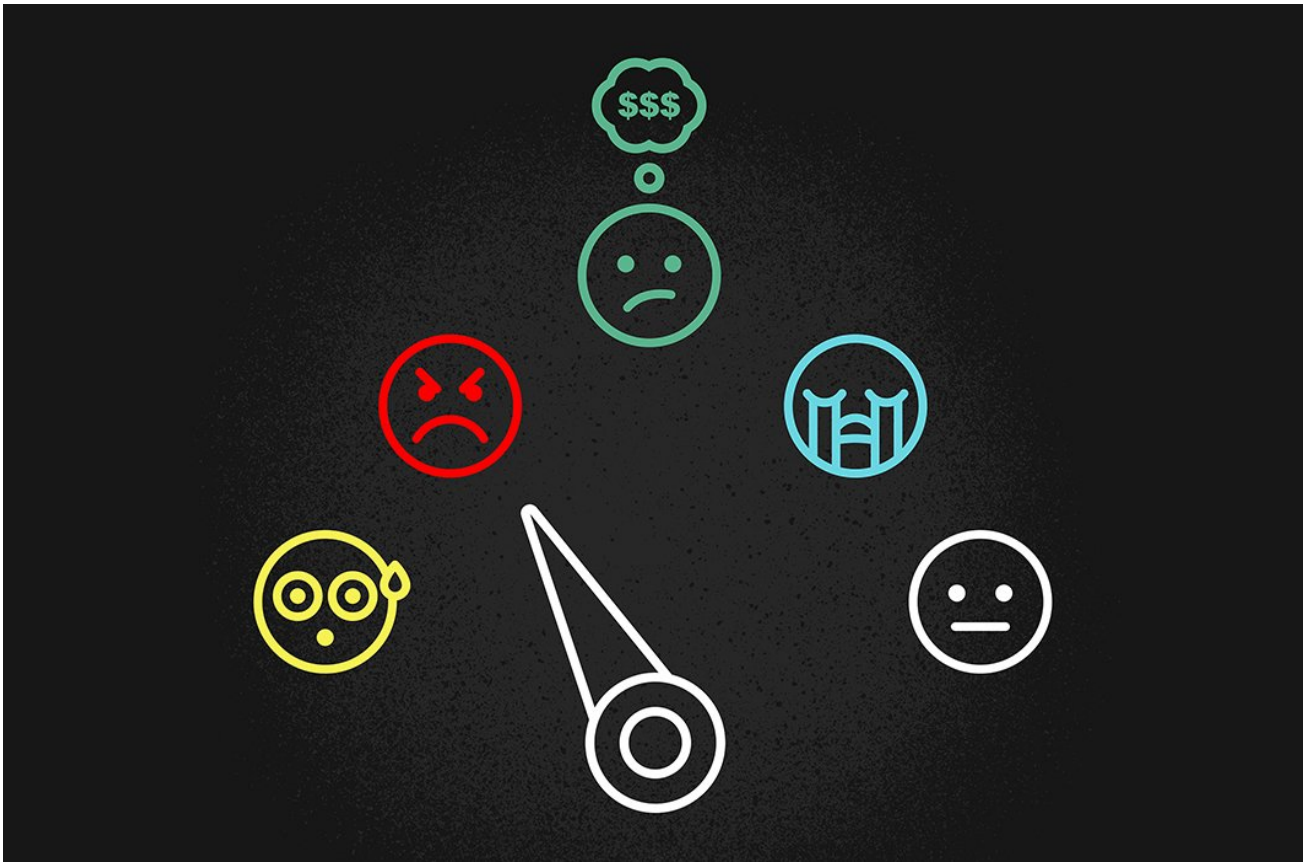
Learn more about Falcon Prevent Next-Gen AV.

Download the white paper: Guide to AV Replacement: What You Need to Know Before Replacing Your Antivirus Solution.

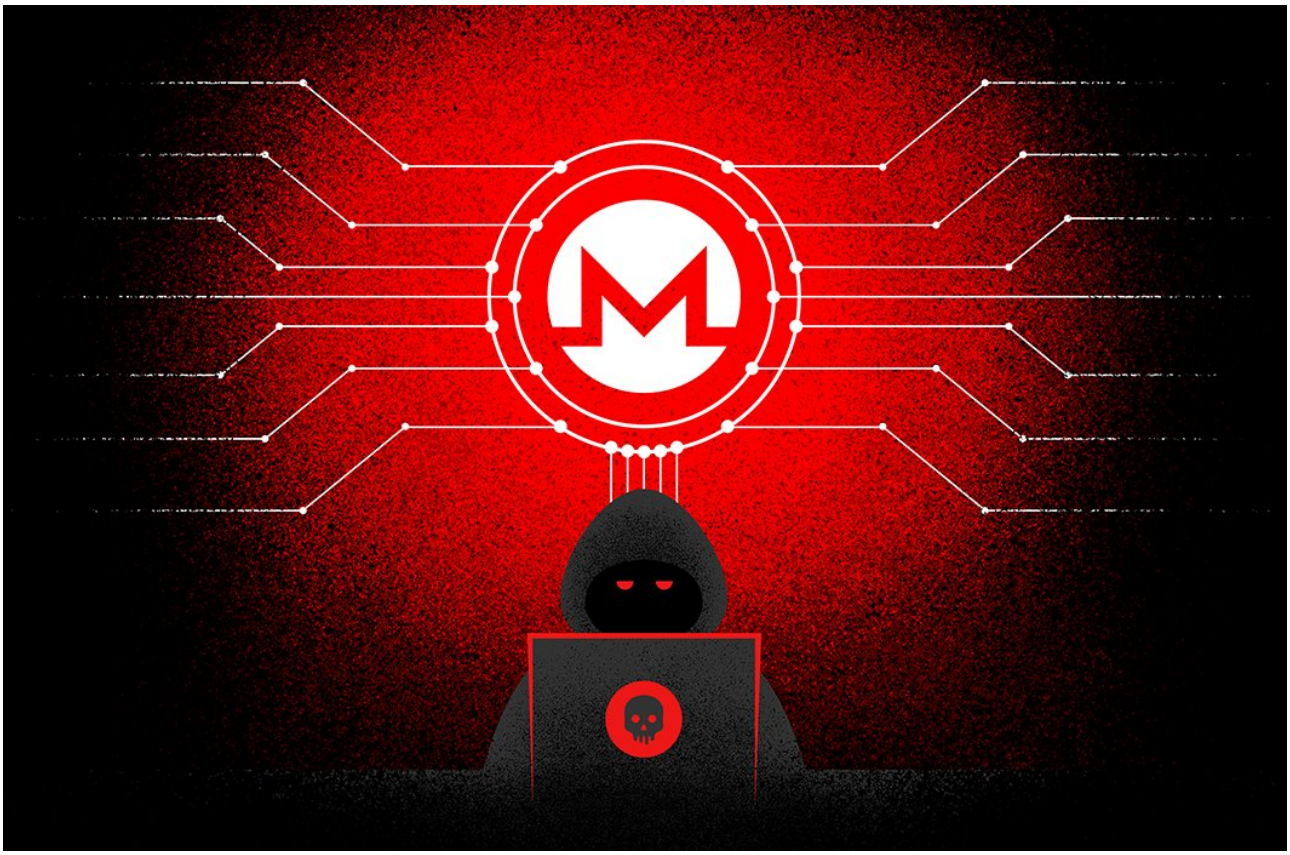
Related Content



Compromised Docker Honeypots Used for Pro-Ukrainian DoS Attack



Navigating the Five Stages of Grief During a Breach



LemonDuck Targets Docker for Cryptomining Operations