Haanrad Technical Analysis

Ben Soer

A00843110

Table of Contents

Functionality

Dynamic Password Generation

When Haanrad first initiates, it starts in *STARTUP* mode. In this mode, Haanrad only listens for DNS requests leaving the system. Upon finding a DNS request packet, Haanrad parses out the first domain in the *QUESTION* section of the DNS header, converts it to a dot notation domain ASCII string and then sends a copy of the packet back to the client. Haanrad only executes this procedure once, thus the client must be previously ready before Haanrad begins execution on the target machine. After sending the packet, Haanrad switches to *FULL FUNCTIONALITY* mode and proceeds to fully operate as a backdoor / ex-filtration system.

Password Based TLS Authentications & Header Authentication

Passwords are sent along with every TLS packet to authenticate with Haanrad and with the client system. For DNS, TCP and UDP packets, specific flags in the TCP, UDP and IP headers are set to identify Haanrad packets. The flags are set as follows:

Protocol	Header	Value
TLS	None	Password in Payload
DNS	DNS Z Bit	1
TCP	TCP RES 1 Bit	1
UDP	IP TOS Bit	1

Password Based Data Encryption

Upon acquiring the password from the network. All data sent to and from Haanrad is expected to be encrypted with the password, ontop of provide authentication as stated previously. In this implementation, Haanrad encrypts all data transferred with a CaesarCipher and uses the password length as part of or entirely the offset value. The encryption methods are as follows:

Protocol	Encryption	Offset Calculation
TLS	Caesar Cipher	Password Length
DNS	Caesar Cipher	Password Length
TCP	Caesar Cipher	Password Length
UDP	Caesar Cipher	Password Length + Destination Port Value

Packet Specific Payload Hiding

All payloads are covertly transferred between the client and Haanrad. Depending on the type of packet used though, specifies the location of where the payload is stored and how much data can be transferred in each packet. The client always sends TLS packets when communicating with Haanrad so as to ensure the highest reliability in communication to Haanrad. This is because the TLS packets can carry the largest payload of 35 bytes and are the only packet type that transports its payload in the body of the packet. The breakdown of TLS and all the other protocols supported are as follows:

Protocol	Payload Location	Payload Max Size
TLS	Application Layer Payload	35 bytes
DNS	DNS ID Header	2 bytes
TCP	TCP Sequence Number Byte 3 & 4	2 bytes
UDP	UDP Source Port Byte 2	1 byte

Popular Process Masking

Haanrad upon start up will immediately rename itself so as to be disguised as another process during execution. Periodically throughout Haanrads execution, it will also rename itself again. Haanrad picks its names by scanning all process names on the system and then tallies them together to determine the most popular name. It then will rename itself to that processes name to look the least suspicious on the victim machine. This functionality is meant to exploit the commonality of many users having numerous internet browsers open, making it look unsuspicious when multiple show up with *ps* or system monitor

File Event And File Synchronization

When operating the ex-filtration functions, Haanrad supports two options: File Event Listening, and File Synchronization. File Event Listening will simply listen in a directory for any changes. Whenever a change in the directory does occur, a message will be sent back to the client that it has happened. During File Synchronization, whenever an event occurs on a file, Haanrad will read the file and transfer its contents to the client. From here the client is designed to emulate the directory Haanrad is hosted on within a local *./sync* folder, maintained by the client. This allows the user to view the files locally from this directory, but also in the same directory format, as it can be found on the exploited system.

Pseudo-Shell Functionality

The pseudo-shell allows the client to interact with a shell on the system Haanrad is hosted on. From here, the user can view system directories and settings from system tools. Haanrad at each command generates a new shell instance, so that no evidence of a command running is visible, after each execution. In order to maintain semi-persistence such as working directory, Haanrad uses a "pseudo-shell" implementation. Specifically when entering *cd* commands, Haanrad will change the working directory variable of itself to emulate that new directory. This allows all future shells to launch in that same directory, emulating the action of a directory change

Protocol Design

The ex-filtration system (Haanrad) uses a simple, minimalist protocol for transferring data between itself and the client and from the client to Haanrad. Communication is not meant to be highly reliable, and functionality has been put in so that failures can be easily detected and cleaned up by both systems.

When sending data, Haanrad uses other packets in the network to send its payload back to the client. The client, always sends all of its contents using TCP TLS packets. Its payload is stored in the body of the TLS packet and is designed to hide as a legitimate TLS packet. This request may or may not succeed though. The TLS packets have been designed to carry up to 35 bytes of data, which is capable of storing the most likely payloads in Haanrad’s various command options. This minimizes the number of packets needing to be sent in a request, increasing the reliability of delivery. For responses, Haanrad uses packets from the network and covertly sends them depending on their type in various header fields or body if it is a TLS packet. Again this is heavily unreliable, and Haanrad will simply blast out these packets at its configured tick speed, set at initialization. By default this is 100 milliseconds which in practice operates at 1 packet per second or less. This variability is present due to the multi-threaded architecture of Haanrad and its multiple components using the timer system. The client then retreives this data from the network. Upon the user requesting a check for new data, the client will then process any received packets if they were successfully transferred.

Packet Design

The Haanrad ex-filtration system and client use a custom packet design which is cutup into the headers or payloads of the packets sent by each system. Each packet starts and ends with a custom header, identifying the “HAAN” packet. The header “{HAAN” identifies the beginning of the packet, and the footer “HAAN}” identifies the ending of the packet. After the header is a packet type byte which identifies and gives context to the payload that follows the 8 bites in the HAAN packet. The currently implemented codes are as follows:

Bit Value	Command Name	Purpose
00000000	DUD	Does Nothing. An Echo/Debug Packet
00000001	CMD	Execute payload in a shell
00000010	FILE	Create listener for path in data. If a listen for the path already exists, remove it

11111110	CMDANSWER	Payload contains response to a CMD packet
11111101	FILEANSWER	Response to a create or remove listener FILE command. Also is a response to a file event. A file event using this command type will not contain file data. Only meta
00000011	FILEDWNLD	Payload contains raw file download information event. Meta information was sent before this in a FILEANSWER packet
00000101	FILESYNC	Create a sync listener for path in data. If the sync listener for the path already exists, remove it. This listener is to sync the files it gets events for. File events will trigger a sync event which will download the file to the client

Following the packet type byte, is then the corresponding payload. “HAAN” packets do not enforce a max length, so any data of any length can be stored within this single packet. When transferred this packet is then cutup and reassembled on the other system. From here the client or haanrad and properly parse the packet and retrieve the required information in order to act on the included data. The identification of the footer label “HAAN}” is then used to identify the end of the packet.

For TLS packets, the authentication password is included in the payload, in the same location as the “HAAN” packet. In this instance, the password precedes the “{HAAN” header in the payload. This password is sent in every TLS packet and is stripped out of the payload before the “HAAN” packet payload is parsed.

Detection

For network administrators and security analysts, Haanrad presents an interesting challenge in its detection. Haanrad sends all of its data using packets from the network, and thus blends in with the flowing network traffic. Additionally, Haanrad picks its masking name based on the currently running processes on the victim machine. Haanrad will name itself with one of the most popular systems and then periodically will reevaluate and rename if necessary.

With this though, an effort to detect Haanrad would be best noticed by its resource usage. Haanrad is not very lightweight due to the use of libpcap and its multi-threaded architecture. In total, Haanrad operates with 4 threads running in parallel, using up substantial CPU resources. Additionally, if not configured correctly, Haanrad will blast all of its data out, causing large spikes in network activity. A second location to identify Haanrad in the network, is the presence of duplicate packets. Since Haanrad physically copies and tweaks existing packets, much of the original packet information is still present. Analysts who observe network traffic can find a pattern of duplicate packets within relative closeness to eachother. An analyst would be able to observe the duplicate packets being sent to the same IP address and thus identifiable as suspicious or malicious activity.

In order to terminate Haanrad from operating on a victim machine, restarting the target computer is the simplest solution. Haanrad does not possess any implementation to revive itself upon restarting. Alternatively, Haanrad can also be terminated from console by searching through the process lists. During high activity, Haanrad will likely sit high on the list due to libpcap. This method would require more caution as Haanrad will rename itself periodically, and thus could confuse the administrator in which process is the ex-filtrator.

To avoid a Haanrad ex-filtration system from being installed in your network, keeping track of contents entering and leaving the office or network is the main solution. Haanrad needs to be manually compiled on the victim machine and requires a number of non-standard libraries in its compilation. This requires manual intervention and likely will be installed via someone physically operating the machine. Haanrad also cannot quickly be installed due to these requirements, and thus is not likely to be installed via a USB or automated trigger. Detection of a Haanrad system in your network is likely a sign members of staff are committing forms of espionage or making a thoughtout intentional effort to damage the network.