TLS and TOS Analysis of Trellix Real Time Monitoring

Alex Asch
UC San Diego

Delaware Wade
UC San Diego

Longtian Bao
UC San Diego

Abstract

The usage of real-time monitoring agents such as Trellix HX raises important questions regarding privacy practices in data handling and retrieval, monitoring scope, and the implications of closed-source software operating with root privileges on user machines. In the worst case, these solutions can serve as another threat vector on a given device, since their programs run with root privilege, and are networked. We sought to first compare and evaluate privacy terms and protections between two comparable solutions, Trellix HX and Norton 360, to ascertain the extent of user data usage, as well as assumed liability on behalf of the product, and the user. Additionally, we performed a network analysis of several components of the software, in order to ascertain what client device data is being collected, as well as potential vectors for exploitation. We conclude that a large amount of private user data is being sent to Trellix and Qualys servers through this active monitoring tool.

1 Introduction

With the Trellix HX real time monitoring software suite being endorsed, and possibly mandated by UCSD for all faculty devices, it is important to understand the privacy implications of the data collected by these products and the user data protections and agreements given to its users.

To identify the user data that is collected, and to analyze the legal protections given to users in regards to this data, we perform two analyses. First, we compare the TOS of the Trellix software suite to a comparable solution, to break down specific guarantees and terms in the TOS in context of endpoint management software. Additionally, we aim to trace and analyze data sent over the network by the Trellix software suite of products, to determine the extent of user data being collected.

Our primary comparison contrasts Norton 360, a popular consumer security software, with the Trellix software currently deployed at our institution across four key features. We found that Trellix employs a significantly more aggressive and detailed approach regarding software limitations and user rights compared to Norton's more open-ended terms.

To obtain the user data sent, we utilize a bpftrace-centric approach, which allows us to bypass the transport layer security (TLS) used by the Trellix software components in their communication to the server. By combining long-term runs of the software suite on virtual machines with this tracing, we were able to ascertain the scope of user data being exfiltrated, and discovered comprehensive scanning of processes, files and kernel configuration data. Considering that the Trellix software suite runs at root level, we believe that these findings highlight a serious privacy implication for end users at UCSD.

2 Trellix TOS comparison

2.1 Perpetual Data Rights vs. Term-Limited Licensing

Table 1 compares the Trellix software we are analyzing with Norton 360, a popular security software. The first notable difference is how they treat user data. Trellix's EULA grants the company a "non-exclusive, irrevocable, worldwide, perpetual right and license to use, reproduce and disclose Threat Data for our business purposes." [1] In contrast, Norton's License and Services Agreement states that for user-provided "Submissions," the company is granted "permission to use, reproduce, copy and translate your Submission on a worldwide basis, for the term of protection of the Submissions by IP rights." [2] Trellix's statement is much more aggressive giving the platform both perpetual and irrevocable license to use and disclose "threat data". By contrasting the terms "Threat Data" and "submission", while both categories are open to interpretation of the company, "submission" seems to more closely refer to data submitted by the user, as opposed to "Threat Data" which can encompass a larger scope.

Table 1: Comparison of key terms and conditions between Trellix and Norton 360 software agreements

Feature	Trellix	Norton 360
Rights to Your Data	Claims a perpetual and irrevocable right to use customer "Threat Data" for its business purposes [1].	Is granted a license to use customer submissions for the duration of the a limited time [2].
Compliance & Fees	Reserves the right to audit software usage and charge specific "out-of-compliance fees" if licensing is inadequate [1].	The consumer agreement does not contain a comparable clause for non-compliance [2].
Termination Rights	Reserves the right to terminate specific software features at any time, stating, "Upon the End-of-Life date of a... feature... Your right to use... the Software feature shall terminate" [1].	Reserves the right to terminate a user's access at any time and for any reason, with or without cause [2].

2.2 Assumption vs. Disclaimer of Infringement Liability

Trellix reserves the right to enforce compliance by stating: "If the systems report or Your prepared Software deployment verification report indicates that You are out of compliance with the license terms of Your Grant Letter and this Agreement, You agree to purchase the additional licenses and pay Us the applicable reinstatement fees associated with the licenses and Support." This means that users must purchase additional licenses and pay out-of-compliance fees when non-compliance is detected. Trellix provides detailed statements regarding compliance requirements and associated fees, whereas Norton does not. This difference may also be related to the common use cases of Trellix vs Norton, with Trellix being a more enterprise focused solution.

2.3 Termination Rights

Both Norton and Trellix imply termination of services or features for any reason for paying users. Trellix states "Upon the End-of-Life date of a... feature... Your right to use... the Software feature shall terminate". This means that if a feature is marked End-of-Life, Trellix may immediately drop support of the feature, without any regards to notification or compensation. In contrast to this, Norton has a clause in which they can terminate a user's access at any time without cause. Thus, while the Trellix TOS termination right may seem extreme, this type of practice is commonplace in endpoint security solutions.

3 Trellix TLS analysis

3.1 Obtaining TLS traffic

The Trellix installer bundles two individual pieces of software: `xAgent` and `Qualys Cloud Agent`. Our first task was to capture the unencrypted TLS traffic these programs send over the network. We utilized Ubuntu/Debian Virtual Machines to install the software and begin testing. Initial scans using wireshark revealed DNS queries for two hostnames: `hexmfv994-hx-agent-1.hex01.helix.apps.fireeye.com` and `qagpublic.qg1.apps.qualys.com`. We attempted to use standard proxy tools such as `mitmproxy` to intercept traffic, however, due to TLS pinning, we were only able to obtain unencrypted TCP keep-alive packets from this method. A possible next step could be to attempt to find logged pre-master secret keys and use them to decrypt captured traffic, but a more straightforward approach is to intercept the unencrypted data as it passes to shared library functions. From here, we tried `ltrace` and `strace` to capture system call traffic from the software. These yielded no useful results, and usage of `ldd` to view the dynamic dependencies of `xagt` and `qualys-cloud-agent` confirmed that custom `libssl.so` and `libcrypto.so` were being used. This is most likely to ensure FIPS compliance [3]. Since the system libraries for `libssl` and `libcrypto` were not being accessed, neither of the above yielded useful results. In the end, we used `bpftrace` scripts to trace the calls into the scripts' custom `libssl` libraries to obtain a long-term traffic log. From our initial log for qualys, we found `lzma` encoded content that we were unable to decompress using solely the data from our TLS dump. To overcome this, we traced the calls to `liblzma` using `bpftrace` as well to obtain the raw data. The final hurdle we overcame was accessing the memory stored at pointer locations `bpftrace` returned from function arguments. We piped the outputs to a custom python script that scraped the data at each memory address into a file. Due to the program operating concurrently on the same locations, some data was altered or corrupted due to race conditions, but the majority of the unencrypted packet data was recovered successfully.

3.2 Metadata, Endpoints, Certs

By analyzing the arguments passed to the custom `libssl.so` library, we were able to uncover all data sent over the network by `xagt` and `qualys-cloud-agent`. For the `xagt` endpoint, we found a certificate revocation list (1) for the `sfServer.hexmfv994-hx-agent-1.`

Table 2: List of Trellix xAgent Endpoints and Descriptions. All subdomains from xagt TLS traffic are for the hostname `hexmfv994-hx-agent-1.hex01.helix.apps.fireeye.com`

Subdomain	Endpoint	Description
sfServer	POST /poll	The most commonly called endpoint. It is polled every 10 minutes, where the client sends a version number(1.0), a status code, versioning info, and date/time. The server then responds with the version id again, current time, cluster id, re-polling timing info, a crl and nonce, and a task struct. Most times this task list is empty, but there were some runs in which we were able to see tasks exchanged. For the entire duration, there is an x-cid header attached, which remains a constant value.
sfServer	GET /content/v1/config/config_id/default	Likely polls for a default client configuration, approximately every 30 minutes. We were unable to get more than a 304 from this endpoint. The configuration ID remained the same for the entire duration of the run
sfPKI	GET /pki/crl/distro	Get a Certificate Revocation List. Only managed to capture a 304 response in our run.
sfPKI	GET /pki/crl/comms	Get a Certificate Revocation List. Only managed to capture a 304 response in our run.
sfPKI	GET /content/v1/intel/ioc/linux-xagent_linux	Retrieves indicators of compromise for the current user device (intel, linux). We captured a response for this, however the content-type was application/vnd.fireeye.hx.payload.v1, and we were unable to retrieve further information from this.
sfPKI	GET /content/v1/intel/mal/exclusions	Retrieves what is assumed to be the scan exclusion list for xagt. We were able to retrieve a large supposedly x-gzip encoded body, but were unable to decompress it with standard decompression or forensics tools. The documentation [4] supports the idea of a centrally managed fetched exclusion list. Additionally, the content-type was a proprietary application/vnd.fireeye.hx.payload.v1 type, and we were unable to retrieve further information from this.
sfPKI	GET /content/v1/intel/mal/hx_exclusions	Retrieves what is assumed to be the scan exclusion list for xagt. We were able to retrieve a large supposedly x-gzip encoded body, but were unable to decompress it with standard decompression or forensics tools. The documentation [4] supports the idea of retrieving an exclusion list from the remote provider. Additionally, content-type was a proprietary application/vnd.fireeye.hx.payload.v1, and we were unable to retrieve further information from this.
agentMSG	POST /msg/v1/lo	Frequently polled endpoint, where the local Trellix install sends system health information, including the hostname, health status, and a frequently empty timestamp. Usually we get an empty response for the server, except for in one instance we got an x509 CRL, which is included in the beginning of the traffic analysis.

Table 3: List of Qualys Cloud Agent Endpoints and Descriptions. The Hostname for all endpoints found in qualys-cloud-agent TLS traffic is `qagpublic.qg1.apps.qualys.com`

Endpoint	**Description**
POST /Qlys/CloudAgent/status	This endpoint is contacted roughly every hour, with the following non-standard headers. X-QLYS-Authorization, with an auth type QHmacV2Auth, then <client_ID> and a colon and what seems to be an authentication key, Q-PROTOVER with a version number 1.0, X-Correlation-Id with a unique bytestring, Q-PROTOCLNTPLATFORM with the client platform, Q-PROTOTYPE with type SCAN, Q-CUSTID with the customer ID, X-QLYS-Timestamp, which is a date-time field, Q-PROTOCLNTARCH, which is the client architecture, Q-CLNTID, which is what we refer to as the machine ID, and Q-PAYLOADHASH which is the hash of the data on this request. The body of this is lzma encoded json, which is a status field, with a state, contextid, and time.
POST /Qlys/CloudAgent/eppagentevent/v1.6/ customer/<customer_id>/agent/<machine_Id>	We observe one call to this endpoint, with the same custom headers as above, except with Q-PROTOTYPE as IOC. The content observed to be sent with this is just lzma encoded client ID and hostname.
POST /CloudAgent/v1.6/customer/<customer_ID>/ agent/<machine_ID>/CAPI	This endpoint contains only the X-QLYS-Timestamp and X-QLYs-Authorization custom headers. However, the body contains a large amount of Customer Data, and the response includes configuration data from the qualys server which is further outlined below
POST /CloudAgent/v1.6/customer/<customer_ID>/ agent/<machine_Id> Manifest/ <manifest_Id> /Delta/<Delta_val> /fragment/1/finalize	We observed one call to this endpoint during our run, which contains the custom timestamp, authorization, and client architecture fields. The reponse was an lzma encoded octet stream of varying length, but all 10000 bytes or more. We were able to retrieve the contents of these calls, which will be noted in a later section.
/CloudAgent/v1.6/customer/<customer_ID>/ agent/018616e5-0ab7-4e58-95fc-ed795fdf227d/command/eventId	Due to limitations in our bpftrace and memory retrieval, while we were unable to get the request type used, we noted that there were requests to this endpoint, with a seemingly empty body and only the custom timestamp and auth headers. While this was called many times, we were unable to further see the contents of this exchange.
GET /CloudAgent/v1.6/customer/<customer_ID>/ agent/<machine_ID>/Manifest/9aa1548c-5cb6-458c-95d7-3b284dbb3946?scope=Global	We noted one call to this endpoint machine, with varying manifest tags across machines. Included were the authorization and timestamp headers. We were able to simply curl a get request to one of these endpoints to retrieve the below anaylized sqlite file, without any additional headers or restrictions.

hex01.helix.apps.fireeye.com host. A certificate for the same host (2) was found in the xagt logs, output of the xagt -log-export command from the Trellix documentation.

Further wireshark analysis demonstrated that the servers communicate using the following cipher suites:

- qualys-cloud-agent (185.125.190.32): TLS 1.2 TLS_AES_256_GCM_SHA384 (0x1302)
- xagt (52.6.220.203): TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = VA, L = RESTON, O = FIREEYE, OU = PRODUCT, CN = PRODCA
        Last Update: Jun  1 18:05:17 2025 GMT
        Next Update: Jul  1 18:05:17 2025 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                9F:EE:7F:37:01:49:AC:02:99:32:4C:98:BF:D8:68:73:8A:16:AD:BA
            X509v3 CRL Number:
                1748801117
No Revoked Certificates.
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        7f:4b:7c:af:69:12:24:c9:61:3b:a4:e2:0d:b8:1f:15:94:fe:
        cc:de:ac:1b:e2:d9:28:81:5a:a8:25:00:0f:bd:b8:44:77:d3:
        21:9b:2e:d4:d5:5f:20:76:91:61:9b:b3:a9:4e:e0:e5:1b:1f:
        d2:96:b2:25:44:25:38:50:0b:3a:73:35:ef:57:15:db:a9:fb:
        d5:7d:fa:73:fc:3a:76:b6:34:46:0e:f6:6b:61:dd:05:13:0b:
        79:90:f5:5c:03:f3:c3:99:93:62:25:eb:b8:f5:0e:91:d3:7a:
        e4:17:73:53:1d:ed:e9:64:09:bf:40:5b:f8:1c:9c:09:e6:43:
        cf:5e:4c:b3:1d:0d:91:f1:d2:94:4c:e6:0d:ec:65:b0:c1:29:
        d6:ee:1d:33:63:0e:a3:9b:4b:b2:19:50:8f:ef:3b:f7:dc:d9:
        5a:68:dc:e0:57:1d:26:bc:ba:1e:8b:0a:e0:1f:56:f6:3a:b1:
        6a:3d:ad:56:d3:40:ed:63:85:83:62:02:c5:5c:b5:da:25:35:
        06:5b:12:49:44:bf:4e:c9:c3:ec:65:d7:d3:20:d8:52:f0:31:
        79:07:10:34:70:ef:bd:89:9a:70:19:90:dd:cb:00:a8:99:a4:
        e2:3a:1a:5c:47:1c:d3:13:7b:d6:df:df:0d:e8:e5:87:8e:2b:
        1e:9e:3e:40
```

Figure 1: xAgent CRL

3.3 Traffic Trace for Trellix xAgent

A full analysis of the endpoints for the xAgent remote server host can be found in table 2.

3.3.1 Task Structures in /poll

While capturing traffic using bpftrace from xagt, we were able to observe scanning task information as exchanged below. For most observations (296), the task struct response was empty, and xagt took no further actions. However, we also recorded 14 observations of a task structure being exchanged between the server and local xAgent client. The exchange takes the form of a JSON response, where the id, state, script uri, base uri, manifest uri, and partialmanifest uri for the task are sent, in addition to the standard poll response [5]

After receiving this response, the client fetches the script from the above script_uri. While we were able to observe the contents of this exchange, and the server response of a binary/octet-stream, we were only able to observe that the

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c2:c6:90:86:bf:88:65:0d
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, ST = VA, L = RESTON, O = FIREEYE, OU = PRODUCT, CN = PRODCA
        Validity
            Not Before: Apr 21 23:13:44 2020 GMT
            Not After : Apr 21 23:13:44 2040 GMT
        Subject: C = US, ST = VA, L = RESTON, O = FIREEYE, OU = PRODUCT, CN = PRODCA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:e6:63:ee:69:da:8c:d6:d2:b2:45:21:93:e3:ae:
                    6f:2b:18:4b:72:27:22:32:b3:e1:8e:fd:b7:d9:c0:
                    31:3e:5b:1d:5b:1e:cc:e4:dd:1e:45:e0:58:97:9f:
                    ad:2c:ff:86:5b:bc:70:c2:2c:b4:f7:24:ad:51:99:
                    c5:d4:6d:7b:58:db:ab:5d:2e:f5:6b:40:9e:9a:c6:
                    32:fb:4a:60:88:22:fb:dd:8a:fc:ae:8c:58:9c:1d:
                    7c:40:f6:61:0b:0e:76:d3:9e:cb:e0:ad:fb:40:5a:
                    95:61:36:ff:65:3a:33:40:65:4f:4f:65:a3:7b:c4:
                    cb:a1:19:a9:b5:ee:62:28:17:85:a9:64:de:a7:25:
                    6f:8f:82:69:b1:7d:89:b2:36:66:ee:66:22:43:0f:
                    a3:04:58:2c:fa:c6:d0:d5:99:42:74:ec:7d:7a:e8:
                    94:2a:ff:73:25:20:06:b0:75:b5:e2:33:2e:44:55:
                    09:ce:d5:88:e2:09:94:31:c5:78:98:c2:bd:8f:ad:
                    b4:51:2b:2f:86:94:f2:14:df:5a:56:18:b7:ab:42:
                    d2:5a:44:53:54:26:82:b9:56:1f:a1:ff:88:c8:68:
                    fa:d9:8a:6a:d0:92:68:ba:33:e2:32:53:05:ee:95:
                    86:25:dc:25:9f:8a:4b:ac:13:0d:74:fb:5d:20:40:
                    d8:8f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                9F:EE:7F:37:01:49:AC:02:99:32:4C:98:BF:D8:68:73:8A:16:AD:BA
            X509v3 Authority Key Identifier:
                9F:EE:7F:37:01:49:AC:02:99:32:4C:98:BF:D8:68:73:8A:16:AD:BA
            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        38:58:c3:3f:6f:22:94:8d:9b:a3:74:89:ce:86:c5:60:5e:e9:
        da:ac:8c:a0:bb:ca:b5:e2:b2:59:46:e9:b7:b9:e7:d8:89:0c:
        d5:6d:e9:9e:94:9e:4d:f8:c7:12:fe:68:72:3f:45:9c:fa:7c:
        0f:2b:2b:bb:f7:e7:8d:5d:91:17:bc:fb:71:56:35:0a:4e:dc:
        a7:5a:37:d6:06:3c:50:ad:77:2f:83:dd:16:78:91:4c:3a:47:
        61:c1:8a:c5:b7:40:b2:07:7f:7e:45:67:0e:32:5d:e4:50:69:
        b2:de:2f:4d:05:3a:ef:ba:22:1c:45:62:71:b8:15:7b:51:fc:
        a9:d1:b0:54:3d:2c:bf:4f:b1:68:f7:9e:8c:cf:fd:2f:2b:0b:
        b6:57:94:05:b1:fe:6d:ab:46:c7:d1:f0:cf:09:ce:cd:e9:0f:
        c7:3e:60:7b:36:c0:74:51:79:1b:8c:94:40:98:cf:6a:98:0a:
        23:26:16:68:59:b1:09:c7:53:79:ef:b1:1c:3a:e3:6f:1f:11:
        ab:75:0c:7c:ab:ef:66:f5:ad:47:f7:68:a9:4b:af:fa:8a:5e:
        06:e0:06:87:81:98:40:97:c3:ff:ff:75:b0:52:83:fa:e6:01:
        8f:d9:ee:aa:fa:60:03:82:45:7b:9b:f2:fb:22:8a:5c:82:7b:
        cf:8e:0b:fd
```

Figure 2: xAgent Certificate

script endpoint from the server always responds with a 2401-byte length response, where and within the first 160 bytes of the response, we find the strings that match with the `xagt` certificate data, that being RESTON, FIREEYE, PRODUCT, and PRODCAagent. After this, the client responds with PUT requests to the endpoint `/task/<task_id>/data/<data_id>` where the data is a 22-byte string. We notice the same first bytes as the script endpoint RESTON, FIREEYE, PRODUCT, and PRODCAagent, but still were unable to find the type of the rest of the stream. We notice that every script corresponds to 5 data PUT responses, where each response is sent with chunked transfer encoding, where then the server responds with a 100 continue code, and then the client sends the bytestream of data. Given the observation of the `application/vnd.fireeye.hx.payload.v1` content type in other locations, we believe this may be a proprietary encoding or format.

3.3.2 Summary

The `xagt` component periodically communicates (about every 10 minutes) using the POLL endpoint to convey the current status and device info of the client device, in which the server responds with the next poll time, the cluster id, and an often empty task list. Also in the approximately 10 minute timescale, the exclusion and IOC lists are updated from the server. Approximately every 15-20 minutes, the client sends a POST request with its health info to the `msg/v1/lo` endpoint. In a much longer timespan, the client refreshes its CRLs from the pki endpoint.

3.4 Traffic Trace for Qualys Cloud Agent

For the following sections, we will list customer ID and machine id as <customer_ID> and <machine_ID>. We noted that across all of our test machines, customer ID was the same for all test devices, while machine ID was separate per device. A full analysis of the endpoints for the Qualys Cloud Agent remote server host can be found in table 3.

3.4.1 Post request to /CAPI endpoint contents

From the request json on the /CAPI endpoint we note that the Qualys agent is sending the hostname, BIOS Serial number, MotherBoard ID, BIOSHardwareUUID, and MAC address of the client device in its Post request, along with self status of the qualys agent, and configuration IDs. In the response, we observe a version control object with binary download paths, as well as a version, and a self-update configuration flag. In addition, we observe the configured scan interval durations for Inventory, Vulnerability, Autodiscovery correlation_prerequisites, linux_mux_prereq, and linux_ebpf_prereq. The full json body is not included in this section but is in the linked data package. [5].

3.4.2 fragment/1/finalize LZMA content analysis

By tracing the liblzma function lzma_code and reading from the pointer to the input stream, we are able to obtain a slightly mangled but still mostly interpretable amount of data that we believe is being sent through the 4th listed endpoint, although we can only fully confirm that that data is being compressed into lzma format, and there is lzma-formatted output being sent [5]. From this, we observe an massive amount of end user device data being exfiltrated through the `qualys-cloud-agent`. Aside from this user data, we also observe a Qualys Agent Local Health Check Tool Report segment, which shows the backend connectivity, certificate data, and agent communication details for Qualys.

The user data that is being exfiltrated contains the outputs of the commands retrieved from the SQlite database analyzed below, however, we will summarize the most concerning of information found here as well. To begin, we see many instances of variously filtered outputs of the "ps" command, listing the user, pid, pcpu, memory, start and end times for each process on the machine, and CMD used to start it. Additionally, we observe the outputs from multiple versions of the "ls" command for both files and symlinks, `systemctl` output reflecting both currently running services and kernel parameters, `ifconfig` and `netstat` outputs for network mapping and information, mounted disc information from a `df` command, and various system properties such as `meminfo`. By checking the readable portions of scripts we see in this file, we can also observe that the scripts being run correspond to commands in the below listed sqlite database [5]. Since our trace only captured a limited duration, we believe that we can assume that any of the commands revealed in the below sqlite database can and may be run by the `qualys-cloud-agent` during its scanning and detection activities. Thus, we conclude the scope of data being gathered and exfiltrated by `qualys-cloud-agent` is any running process, the name and properties of any file, any network device, any mount point, all kernel paramters, and any running service on the device. This endpoint of the `qualys-cloud-agent` sends an extreme amount of private user data to the server.

3.4.3 /CloudAgent/v1.6/customer/<customer_ID>/agent /<machine_ID>/Manifest/.../?scope=Global

This endpoint is by the Qualys agent, and retrieves a SQLite database. We were able to access this endpoint, and retrieve a SQL file with the following tables.

- AgentInfo
- AgentInfoOS
- FilterOS
- InstalledSoftware
- InstalledSoftwareOS

Table 4: Functions in UnixCommandOS table

Function Detail	Command
retrieves permissions for local group and password files	/bin/ls -l /etc/group /bin/ls -l /etc/passwd /bin/ls -l /etc/shadow
Retrieves network info of current active connections	netstat -an | grep -E ' ^ tcp | udp'| awk '{print $4,$5}'| grep -F -v '' netstat -an -f inet|awk '{print $1,$2}' netstat -an | grep -E '^ tcp | udp'| awk '{print $4,$5}'|grep -F -v ''|
auxww lists all running processes, howver, each call if it is piped to grep to only return specific contents such as [msc]HostService.ini, or '[a]ctivemq'	/bin/ps auxww | grep <sometext>
cat /etc/shadow | grep 'root:'	Gets root user password hash
Searches current enviornment for AWS keys	/usr/bin/env | grep Eo "AWS_ACCESS_KEY_ID= |AWS_SECRET_ACCESS_KEY= |AWS_SHARED_CREDENTIALS_FILE= |AWS_SESSION_TOKEN= |AWS_WEB_IDENTITY_TOKEN_FILE="
Mac only. Searches all installed applications for "Security"	system_profiler SPInstallHistoryDataType | grep "Security"
Gets all Ipv6 fields in ifconfig	/usr/sbin/ifconfig -a | /usr/bin/grep IPv6
Scans all running docker containers	docker ps –no-trunc
NIS service access details	cat /var/yp/securenets

Table 5: Functions in UnixCommand table

Function Details	Command
Retrieves package repo config data	cat /etc/yum.repos.d/* cat /etc/dnf/modules.d/*
Entire command not included for brevity, but searches for Java Service Listeners (or other programs with JSL in name), and retrieves their paths	(timeout 300 netstat -lnp|grep JSL | awk '{print $7}' \| cut -d ls -l /proc/$CHECKJSL/exe|awk '{print $11}'
Retrieve all openshift cluster details	oc get nodes -o wide

Table 6: Functions in MultiPassFunctionsOS table

Function Name	Function Details
`collect_aws_metadata_v2()`	Checks if the current device is an Ec2 instance, and collects metadata and configuration info
`find_java_service_listeners()`	Finds running Java Service Listeners (JSL) and retrieves their executable paths.
`data_collection_running_processes_rpm()`	Searches and correlates running processes with packages
`ai_ml_inventory_data_collection()`	Searches for pytorch, keras, huggingface, and mxnet paths, and returns model file paths

- MultiPassCommandsOS
- MultiPassFunctionsOS
- ProviderCommand
- ProviderMetadata
- ProviderMetadataInfo
- TechnologyFunctionCall
- TechnologyFunctions
- UnixCommand
- UnixCommandOS
- UnixSettings

Of these tables, InstalledSoftware, TechnologyFunctions, TechnologyFunctionCall, ProviderMetadataInfo, ProviderMetadata, ProviderCommand, FilterOS, and AgentInfo were empty.

Of the tables with contents, UnixSettings contained name value pairs of Qualys settings, listed below

- `MANIFEST_TYPE: VM`
- `Version: 2.6.348-3`
- `SCHEMA: 1.0`
- `OS_FILTER_VERSION: 18`
- `PROVIDER_METADATA_VERSION: 2`
- `NEW_SCHEMA: "1.5,1.7,2.0,2.1"`
- `TIMESTAMP: "2025-06-11 17-27-20 UTC"`

AgentInfoOS contains the columns ManifestID, Category, AttributeName, Command, OSName, OSExclude, PreAggregate, and PostAggregate. This file contains a list of names attributes, with the command required to obtain that data on a specific operating system. MultipassCommands contains entiries of ManifestID, Qid, FunctionName, Arguments, OSName, and OSinclude, with the function names seemingly pointing to functions in other tables. The most interesting contents of this database are in the UnixCommandOS, and UnixCommand. These tables contained data in the following format ManifestID, Command, WorkingDirectory, Arguments, PreAggregate, PostAggregate, OSName, and OSExclude, with the two last OS fields being excluded in the Unix Command tables. The Command field for this table contained base64 commands, which we assume to be run by the local Qualys agent in its threat monitoring activities. Thus, by getting the commands from this database, we were able to view a list of what the Qualys agent could be monitoring on local environments in a standard operating setting. Additionally present is the MultiPassFunctionsOS file. This file contains a similar layout with FunctionName, FunctionBody, and DependancyFunctions, where FunctionBody is a base64 encoded script to collect data on the end user device.

From the gathered scripts and functions, we find that the contents of these files likely point to extremely wide reaching scanning of the end user system. While we will not list all commands that we retrieved from the files, we include a subset of these in the above tables 4-6 to demonstrate the extent of the information being scanned by qualys. [5]

3.4.4 Summary

The `qualys-cloud-agent` communicates with the host every hour to establish status and connectivity, as well as send base client machine information such as biosUUID to the server, and retrieves configuration data in response. In a much longer interval, the `qualys-cloud-agent` retrieves manifest data and sends scan data to the server. By tracing the lzma-compressed data by the qualys agent and checking the output of it in relation to the retrieved sqlite file, we conclude that the commands as listed in the sqlite dump are being run on the local machine, and that information is compressed, then sent to the qualys server. We believe this amount of data collected and sent represents a massive breach in user privacy.

4 Further Work

While we were able to find many examples of user data being exfiltrated by the Trellix product suite, we did not find any actionable vulnerabilities. We believe that further work could look into a remote adversary sending or intercepting and modifying the sqlite db in order to send commands that would allow remote control of a client system. The most extreme of this being a compromise of Trellix/Qualys servers allowing for a remote attacker to potentially hijack all clients with the Trellix product installed.

References

[1] Trellix, "End User License Agreement," Sec. 4.2, 5.6, 11.2, 13.3. Accessed: Jun. 11, 2025. [Online]. Available: https://www.trellix.com/assets/legal/trellix-eula.pdf

[2] Gen Digital Inc., "Gen License and Services Agreement," Part 2, Sec. 6, 11, 16. Accessed: Jun. 11, 2025. [Online]. Available: https://www.norton.com/terms-of-sale

[3] Trellix, "Trellix FIPS 140-2 Cryptographic Module Security Policy," Mar. 30, 2023. Accessed: Jun. 12, 2025. [Online]. Available: https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp4492.pdf

[4] Trellix, "Endpoint Security xAgent Administration Guide Release 35.31.0". Accessed: Jun. 11, 2025. [Online]. Available https://docs.trellix.com/bundle/agent_35_ag/page/UUID-3193bc52-42bd-4503-3bef-e17219bbfb22.html

[5] Drive link to zipped file of Sqlite table, bpftrace output, and scripts