Global Information Assurance Certification Paper

QUIC & The Dead: Which of the Most Common IDS/IPS Tools Can Best Identify QUIC Traffic?

GIAC (GCIA) Gold Certification

Author: Lee Decker, deckerl@gmail.com
Advisor: *David Hoelzer*

Accepted: 04/03/2020

Abstract

The QUIC protocol created by Google for use in their popular browser Chrome has begun to be adopted by other browsers. Some organizations have a robust strategy to handle TLS with HTTP2. However, QUIC (HTTP/2 over UDP) lacks visibility via crucial information security tools such as Wireshark, Zeek, Suricata, and Snort. Lack of visibility is due to both its use of TLS 1.3 for encryption and UDP for communication. The defender is at a disadvantage as selective blocking of QUIC isn't always possible. Moreover, some QUIC traffic may be legitimate, and so outright blocking of endpoints that use QUIC is likely to cause more issues than it solves. To complicate matters further, QUIC has begun to appear in Command and Control (C2) frameworks like Merlin as an additional means of hiding traffic.

This paper seeks to establish the current state of open-source detection tools, identify which tools detect the most metrics, and add to current detection capabilities by creating a proof of concept Zeek script to enhance detection.

1. Introduction

The QUIC protocol was created in 2012 by Google engineer Jim Roskind. QUIC improves the performance of web-based applications by using UDP instead of TCP. UDP allows the connection to enhance the performance of web-based applications by reducing the traditional TCP three-way handshake to a single UDP round-trip (Ghedini, 2018). In addition to solving performance challenges, QUIC also supports encryption by default using TLS 1.3. To further complicate matters, the IETF took the original Google QUIC protocol (GQUIC) and improved it. This QUIC protocol expands and diverges from GQUIC (Ghedini, 2018).

Both GQUIC and QUIC create new challenges for information security practitioners. By utilizing both UDP which, is connectionless and TLS 1.3 for encryption, many of the proven packet tools such as Wireshark, Zeek (formerly Bro), Suricata, and Snort loose visibility or functionality. Most QUIC/GQUIC traffic may be legitimate. Google uses it to speed up Youtube, and Microsoft has plans to use it to accelerate SMB/file traffic (Pyle, 2020), so outright blocking of endpoint traffic is likely to create more issues than it solves. To complicate matters further, GQUIC has begun to appear in Command & Control (C2) frameworks to help obfuscate malicious traffic. Russel Vay Tuly added support for GQUIC to the Merlin C2 framework in 2018 to aid penetration testers and defenders.

Both QUIC & GQUIC protocols are works in progress, and implementations may vary among applications. Different libraries support different versions and features (Shah, 2018). Both Wireshark and Zeek's plugin Bro-Quic by Corelight support earlier versions of GQUIC (Google QUIC). The GQUIC plugin by Salesforce supports the current version of Q046 (Yu, 2019).

The lack of support for QUIC is found not only among open-source security solutions but also among commercial proxy solutions like Cisco's Web Security Appliance. Many commercial firewall vendors currently recommend blocking QUIC (Liebetrau, 2018). Chrome and other browsers will default to HTTP/HTTPS using TCP if GQUIC/QUIC isn't available.

Lee Decker, deckerl@gmail.com

Traditional web traffic over TCP requires a three-way handshake. QUIC uses UDP instead of TCP. UDP speeds up web traffic by causing less delay and fewer packets sent (Niroshan,2017). Using UDP instead of TCP provides several benefits, including connection migration, forward error correction, improved establishment latency, and better congestion control.

Various tools must decode QUIC's packet structure to gain insight into its contents. QUIC consists of two different packet types: special and regular.

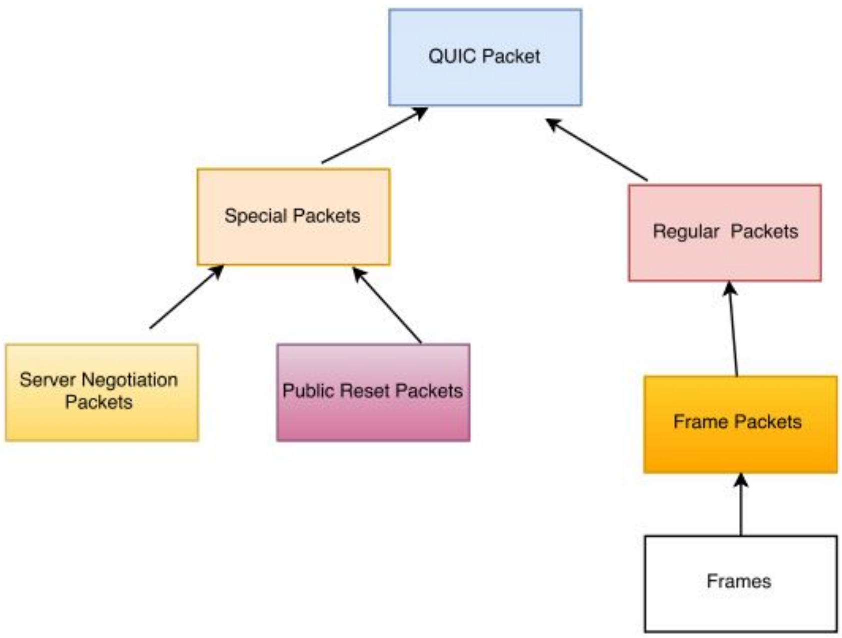

Figure 1 - QUIC Packet Types (Niroshan, 2017)

Both types of QUIC packets begin with a public header between 1 and 51 bytes that provides details concerning the rest of the packet.

Lee Decker, deckerl@gmail.com

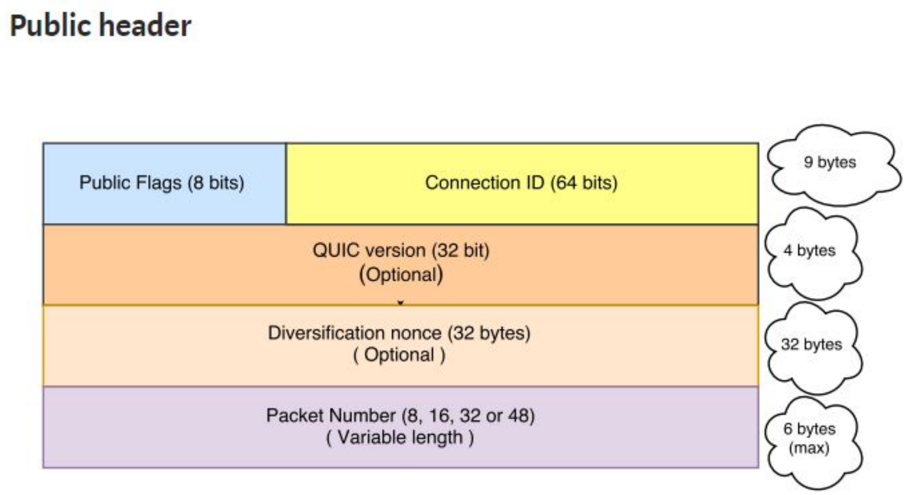

Figure 2 - QUIC Public Header (Niroshan, 2017)

Special packets consist of two types: version negotiation packets or public reset packets. Regular packets consist of frame packets with type and payload information (2017, Niroshan). The researcher's focus will primarily be on these packets, which are un-encrypted and can provide critical information.

The first communication between a client and a new server consists of a *helo* packet, followed by a rejection response packet containing the information needed to establish the connection. The *helo* packet is then resent with the new parameters, and an encrypted channel created. On further communications, the client can use cached information to establish encryption, thus bypassing the un-encrypted packets which are necessary to fingerprint and gather information (2019, Yu).

These *HeloInfo* packets contain up to twenty-eight tags that can be analyzed to gather information about the connection, including user-agent header and server information.

Server rejection packet – *RejInfo* contains up to seventeen tags that provide additional information to aid in the profiling of the packet.

The diagram below illustrates how GQUIC handles the initial handshake and all further handshakes that follow.

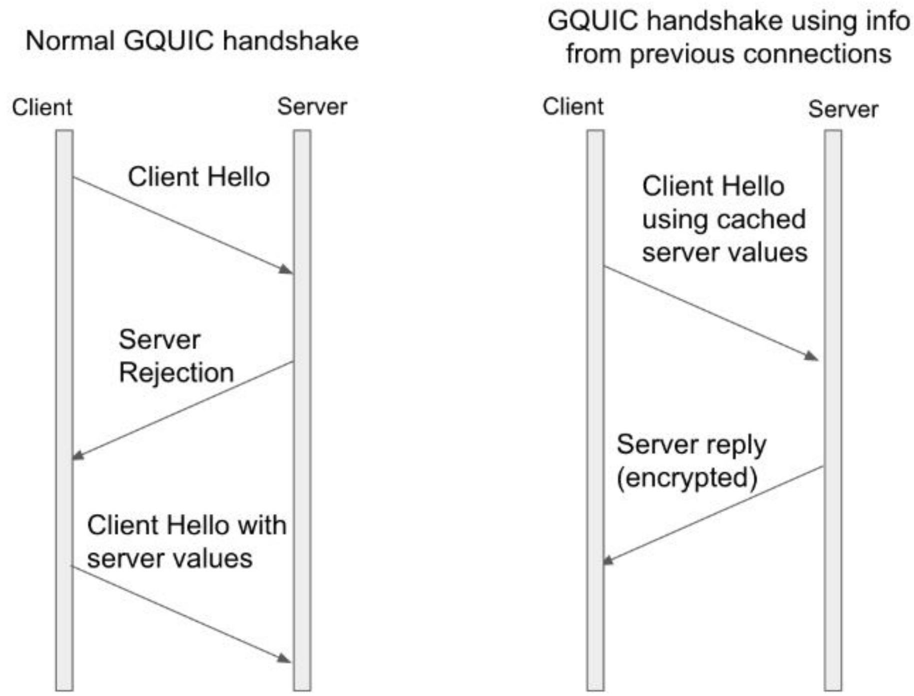

Figure 3 - GQUIC Handshake (Yu,2019)

A study published in 2019 by Jan Ruth and others, shows GQUIC accounts for as much as 40% of Google's traffic. With the more recent adoption of QUIC in Firefox and support from Cloudflare (Ghendi, 2018) and other providers, these numbers will only increase.

While this is good for the public due to speed increases, reduced latency, and easier maintenance in the userspace instead of the operating system (Pearce,2019), it presents challenges in the corporate space.

Information security professionals will need to adapt old tools and develop new techniques to address this blind-spot in corporate systems.

The researcher's goal is to see what commonly available tools have the best support for the current version of GQUIC/QUIC and create a Zeek script to provide additional intelligence. By analyzing the state of the existing open-source tools, the researcher will gather additional information to aid security professionals in both controlling known "good" traffic and identifying and blocking malicious traffic.

Lee Decker, deckerl@gmail.com

2. Research Method

2.1. Lab Design

2.1.1. Overview of Lab

The researcher chose to use Vmware Workstation 15.5.1 build-15018445 to virtualize the infrastructure. Data gathering and analyzation workspaces consist of Virtual machines of KALI Linux 2019.4 and Security Onion 16.04.6.3. These distributions are both readily accessible and stable. To the KALI workspace (based on Debian Linux), the researcher added the following applications – Chrome version 79.0.3945.130 (Official Build) (64-bit), and Firefox version 72.0.2 (64bit).

Security Onion contains the following version of tools: Snort 2.9.15, Suricata 4.15 Bro/Zeek 2.6.4, and Wireshark 2.6.10. Two plugins were then added to Bro/Zeek: corelight/bro-quic and salesforce/GQUIC_Protcol_Analyzer.

Two additional VM's were built based on Kali 2019.4, one to act as the Merlin C2 client and one to serve as the Merlin C2 server. Merlin is currently in beta, version 0.8.0.

2.1.2. "Good" versus "Bad" packets

After being configured to enable QUIC, Google Chrome & Firefox are each used for ten packet captures using tcpdump. These packet captures are "good" or potentially "legitimate" traffic that will be analyzed using various tools.

GQUIC is the default with the current version of Google Chrome. But to ensure GQUIC was enabled, the researcher toggled QUIC under chrome://flags/ (Liebert, 2018). You can then confirm the visited website is using GQUIC and the version used while in developer mode. Then while Google, Youtube, and other sites known to use GQUIC/QUIC are visited, **tcpdump -i eth0 -w filename** is running in an additional terminal window to capture the packets.

QUIC is the default with the current build of Firefox. But to confirm this, the researcher goes to **about:config** and search for **network.http.http3.enabled**. After making the change, restarting Firefox applies the setting. HTTP3 is another name for the newer QUIC protocol, not the GQUIC protocol. Tcpdump captures packets while various sites are visited.

Merlin C2 client and the Merlin C2 server run on separate VM's and ten packet captures are created from the client-side, while various commands run. These packet captures are considered "bad" or potentially "malicious" traffic.

2.2. Tools used for Analysis & Packet Generation

The researcher reviewed the current state of open-source packet analysis tools and frameworks to determine which was best for dealing with the GQUIC/QUIC protocol.

2.2.1. Suricata

Suricata is an open-source network threat detection framework. The engine can act as both an IDS, IPS, and NSM. Packets can also be processed offline, which is the primary use case demonstrated for this research. Although UDP and TLS are both supported by the protocol parser, QUIC is not currently supported. The researcher suspects the information obtained from our packet captures will be limited. Suricata's latest stable version is 5.0.2

2.2.2. Snort

Snort is an open-source IPS, IDS framework. The engine can also process packets offline. Suricata does not support decoders for QUIC, but decoders for UDP and TLS do exist.

Snort is the bases of enterprise products like Cisco Firepower. Cisco's latest recommendation is to block QUIC traffic, forcing browsers back to TCP/TLS. (Maynard, 2018).

2.2.3. Zeek

Zeek is the open-source network security monitor formerly know as Bro. It is a popular framework for extract meta-data from packets, providing analysis, and acting on that meta-data. Zeek contains protocol plugins for both UDP and TLS. Third-party plugins are available to provide additional information on the QUIC protocol. The researcher found plugins from both Corelight and Salesforce via Github. The Corelight plugin is two years old and only supports up to version Q043 of QUIC. Due to the various capabilities listed above, the researcher feels it will be the best tool to identify malicious QUIC/GQUIC packets. Zeek's latest version is 3.0.1

When analyzing packet capture files the -c flag ignores any checksum errors that may occur, and the -r flag is used to read the .pcap file.

2.2.4. Zeek with Salesforce Plugin

This plugin for Zeek was developed in 2019 to provide additional visibility into GQUIC packets. Using BinPAC, the plugin focuses on the non-encrypted packets of GQUIC – the client *hello* and server rejection packets. The plugin allows the gathering of certificates, user-agent strings, and other valuable data used for fingerprinting "good" versus "bad" traffic (Yu, 2019). The researcher expects this tool to the most useful for dealing with QUIC packets currently.

The plugin is first downloaded from Github using the command **git clone https://github.com/salesforce/GQUIC_Protocol_Analyzer**. It can then be configured and installed into Bro using sudo and the following commands - **./configure, make, make install**. The plugin is installed in the /opt/bro/lib/bro/plugins directory. Bro -N verifies a successful installation. The output should look like the following.

```
Bro::SSH - Secure Shell analyzer (built-in)
Bro::SSL - SSL/TLS and DTLS analyzers (built-in)
Bro::SteppingStone - Stepping stone analyzer (built-in)
Bro::Syslog - Syslog analyzer UDP-only (built-in)
Bro::TCP - TCP analyzer (built-in)
Bro::Teredo - Teredo analyzer (built-in)
Bro::UDP - UDP Analyzer (built-in)
Bro::Unified2 - Analyze Unified2 alert files. (built-in)
Bro::X509 - X509 and OCSP analyzer (built-in)
Bro::XMPP - XMPP analyzer (StartTLS only) (built-in)
Bro::ZIP - Generic ZIP support analyzer (built-in)
Bro::AF_Packet - Packet acquisition via AF_Packet (dynamic, version 1.3)
Salesforce::GQUIC - Google QUIC (GQUIC) protocol analyzer for Q039-Q046 (dynamic, version 1.0)
```

Figure 4 - Validating Bro Plugin

2.2.5. Merlin C2 Framework

Merlin is a command and control framework written in the Go programming language by Russel Van Tuyl to aid in red team exercises. It was designed from the start to use HTTP/2 for communications and then updated to allow the use of the GQUIC protocol. This use of encryption creates challenges for IPS/IDS solutions, and the inclusion of GQUIC made it the perfect candidate for the researcher to generate "bad packets" for testing (Villarreal, 2019).

The Merlin framework consists of an agent and a server. The server and agent both must use the command-line switch **-proto hq** to use GQUIC as the communication protocol. Merlin uses the GOQUIC library, which currently supports version Q044 of the protocol.

The latest version of Merlin C2, v0.8.0 beta, contains a known bug, in which the server will die if told to use the GQUIC protocol and the built-in certificate. After a discussion with the author, the researcher learned that if you create a self-signed certificate, this problem is corrected. The following commands generate this certificate.

openssl genrsa -out privatekey.pem 1024

openssl req -new -x509 -key privatekey.pem -out publickey.cer -days 1825 openssl pkcs12 -export -out public_privatekey.pfx -inkey privatekey.pem -in publickey.cer

Figure 5 - Merlin C2 server using GQUIC

Then the server can be started with flags pictured in Figure 5 above. Tcpdump performs packet captures between agent and server for later analysis.

2.3. Wireshark

Wireshark is an established GUI used for packet analysis. The researcher will use Wireshark to examine GQUIC/QUIC packets for information. Wireshark will act as a control to validate what each open-source tool produces. The current stable version of Wireshark is 3.2.1. Versions of Wireshark, as recent as 3.0.3, had challenges examining version Q046 GQUIC packets, which represent most of the traffic currently seen (Yu, 2019).

Security Onion currently provides an older version of Wireshark by default. The researcher will use the more current version installed on Kali when conducting an analysis.

Lee Decker, deckerl@gmail.com

3. Findings and Analysis

Each tool processes the packet captures. The resulting output or data extracted provides information and insight for the researcher to help differentiate legitimate traffic from malicious traffic. It also further illustrates the differences between GQUIC and QUIC and highlights the strengths and weaknesses of each tool.

3.1. Suricata

Suricata is capable of reading .pcap files offline using the -r switch. The researcher updated the installed Suricata rules (**sudo suricata-update**) and downloaded the emerging threats rule set.

Suricata found no concerns in the "good" .pcap files containing chrome traffic.

Figure 6 - Suricata - Chrome Packets

Suricata runs against the packet capture contain the Merlin C2 traffic, and no rules were triggered.

Lee Decker, deckerl@gmail.com

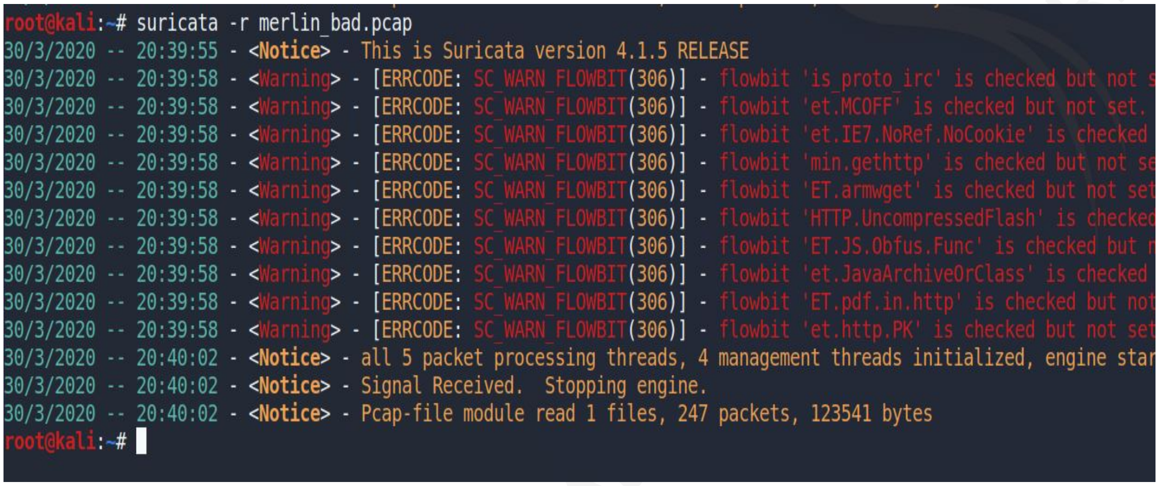

Figure 7 - Suricata Merlin C2 Packets

Suricata was unable to provide any additional information concerning GQUIC/QUIC traffic since neither the legitimate nor the malicious traffic triggered any of the signatures or rules.

The remaining Google Chrome, Firefox, and Merlin C2 packets showed similar findings when processed with Suricata.

3.2. Snort

Snort is capable of reading .pcap files offline using the -r switch. Snort processed eleven thousand four hundred eighty-three packets processed from the chrome_newtest1.pcap. No signatures or rules matched, and the GQUIC traffic appears as UDP packets mixed in with TCP packets of regular traffic. Nine additional Chrome packet captures processed with similar results. The ten Firefox packet captures yielded comparable results.

Lee Decker, deckerl@gmail.com

```
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
====================================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:        11483 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:        11451 ( 99.721%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:        10167 ( 88.540%)
        TCP:         1280 ( 11.147%)
        IP6:           14 (  0.122%)
    IP6 Ext:           14 (  0.122%)
   IP6 Opts:            0 (  0.000%)
      Frag6:            0 (  0.000%)
      ICMP6:            2 (  0.017%)
       UDP6:           12 (  0.105%)
       TCP6:            0 (  0.000%)
     Teredo:            0 (  0.000%)
```

Figure 8 - Snort Chrome Packet

Snort processed the Merlin C2 capture containing 247 packets. No rules were triggered, and all the traffic seen is UDP. An additional nine packet captures from Merlin C2 were processed with similar results.

```
  Total free space (fordblks):          108496
  Topmost releasable block (keepcost):  102368
================================================================
Packet I/O Totals:
   Received:          247
   Analyzed:          247 (100.000%)
    Dropped:            0 (  0.000%)
   Filtered:            0 (  0.000%)
Outstanding:            0 (  0.000%)
   Injected:            0
================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:          247 (100.000%)
       VLAN:            0 (  0.000%)
        IP4:          220 ( 89.069%)
       Frag:            0 (  0.000%)
       ICMP:            0 (  0.000%)
        UDP:          220 ( 89.069%)
        TCP:            0 (  0.000%)
        IP6:            2 (  0.810%)
```

Figure 9 - Snort Merlin C2 Packet

Snort was unable to provide additional meaningful information from either packet capture for similar reasons as Suricata. Neither legitimate nor malicious traffic triggered any rules or signatures.

3.3. Zeek

Zeek, by default, generates the standard set of log files. These log files show the UDP traffic on 443 but can't identify it as GQUIC/QUIC. The IP source and destination information may help map known malicious IP addresses that the traffic is going to. The remaining nine Google Chrome packet captures yielded similar results as did the Firefox traffic.

```
deckerl@deckerl-seconion:~/test1$ cat conn.log | bro-cut | grep udp | grep 443
1581659755.152907       COOqqwsPYO5AiJHl        192.168.189.130 37241   216.58.199.109  443     udp     -
0       Dd      7       3369    6       5126    -
1581659755.940853       C879wv2Hp70qT4v4Y1      192.168.189.130 55407   172.217.24.195  443     udp     -
0       Dd      5       4254    3       4134    -
1581659757.689428       CV05Vd4p9kVIjHBUk7      192.168.189.130 34217   216.58.220.196  443     udp     -
0       Dd      35      7474    36      11542   -
1581659761.734125       C5jhF323fv2Wk5VKzb      192.168.189.130 52550   216.58.220.195  443     udp     -
0       Dd      34      5007    56      75743   -
1581659765.134226       CoTd6gXoYkqymhBA2       192.168.189.130 37515   172.217.163.227 443     udp     -
0       Dd      9       5926    7       5621    -
1581659770.489050       CPnVvh2xIUomQ4clKa      192.168.189.130 54020   172.217.161.163 443     udp     -
0       Dd      11      6585    11      9874    -
1581659771.723965       CSi01NDjvQAqk6pE7       192.168.189.130 56972   216.58.200.78   443     udp     -
0       Dd      6       3004    7       9646    -
1581659785.488005       CDYfje1TFL3GLzzSJb      192.168.189.130 42580   216.58.199.109  443     udp     -
0       Dd      29      7875    46      57958   -
```

Figure 10 - Zeek - Good Packets - Conn Log

Zeek is unable to provide much information from the Merlin C2 packet. No certificate information is processed, and only standard connection information supplied. Merlin traffic has far fewer packets compared to legitimate QUIC web traffic, and there is none of the associated HTTP/HTTPS traffic surrounding it as part of the transaction. The malicious C2 traffic did not generate ssl.log's or x509.log's usually seen with standard TCP encrypted traffic.

```
deckerl@deckerl-seconion:~/merlin_bad_zeek$ cat conn.log | bro-cut | grep udp | grep 443
1582134598.911759       CfZU1G33Td6peY18a4      192.168.189.131 32856   192.168.189.132 443     udp     -
0       Dd      108     63078   107     54927   -
deckerl@deckerl-seconion:~/merlin_bad_zeek$
```

Figure 11 - Zeek - Merlin C2 Packet

3.4. Zeek with Salesforce module

Zeek, with the Salesforce plugin installed, identifies the GQUIC packets and creates an additional log file. This log file includes GQUIC version information, browser head information, as well as an MD5 fingerprint based on the version and tags in the client *hello* packets. This fingerprint can help identify "good" versus known "bad" packets.

Zeek produced similar results for the remaining Google Chrome packet captures but was unable to process the data from the Firefox packets using the newer QUIC format.

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       version server_name     user_agent      tag_count
s
#types  time    string  addr    port    addr    port    count   string  string  count   string  string
1581659755.152907       CIkmEf4bdrY1YjuFyh      192.168.189.130 37241   216.58.199.109  443     46      accounts.google.com     Chrome/79.0.3
_64     17      910a5e3a4d51593bd59a44611544f209        46,PAD-SNI-VER-CCS-UAID-TCID-PDMD-SMHL-ICSL-NONP-MIDS-SCLS-CSCT-COPT-IRTT-CFCW-SFCW
1581659755.187029       CuzcJ42lNrFlrSfIRd      192.168.189.130 42531   216.58.220.196  443     46      www.google.com  Chrome/79.0.3945.130
7       910a5e3a4d51593bd59a44611544f209        46,PAD-SNI-VER-CCS-UAID-TCID-PDMD-SMHL-ICSL-NONP-MIDS-SCLS-CSCT-COPT-IRTT-CFCW-SFCW
1581659755.789413       ClMStb4fhCDJdLCgc2      192.168.189.130 58483   216.58.220.202  443     46      fonts.googleapis.com    Chrome/79.0.3
_64     17      910a5e3a4d51593bd59a44611544f209        46,PAD-SNI-VER-CCS-UAID-TCID-PDMD-SMHL-ICSL-NONP-MIDS-SCLS-CSCT-COPT-IRTT-CFCW-SFCW
1581659755.940853       COtUTl1uKCG6J5UAXb      192.168.189.130 55407   172.217.24.195  443     46      fonts.gstatic.com       Chrome/79.0.3
_64     17      910a5e3a4d51593bd59a44611544f209        46,PAD-SNI-VER-CCS-UAID-TCID-PDMD-SMHL-ICSL-NONP-MIDS-SCLS-CSCT-COPT-IRTT-CFCW-SFCW
1581659757.689428       CU37LI2K0nJyjyrvCi      192.168.189.130 34217   216.58.220.196  443     46      www.google.com  Chrome/79.0.3945.130
7       910a5e3a4d51593bd59a44611544f209        46,PAD-SNI-VER-CCS-UAID-TCID-PDMD-SMHL-ICSL-NONP-MIDS-SCLS-CSCT-COPT-IRTT-CFCW-SFCW
1581659757.837817       CU37LI2K0nJyjyrvCi      192.168.189.130 34217   216.58.220.196  443     46      www.google.com  Chrome/79.0.3945.130
```

Figure 12 - Zeek - SalesForce Plugin - Good Packets - Gquic Log

The Salesforce plugin extracts the same data from the Merlin C2 packet capture, but there is minimal information to differentiate the regular browsing traffic from the bad C2 traffic. Web browsing has a mix of TCP & UDP connections, whereas the C2 framework does not. The CYU tags found here match the ones mentioned by the Salesforce GitHub site as being associated with Merlin C2. The researcher's use of a self-signed certificate did not result in different CYU tags for Merlin C2.

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       version server_name     user_agent      tag_count
s
#types  time    string  addr    port    addr    port    count   string  string  count   string  string
1582134598.911759       CnuBau4Z1RoXnRMvA9      192.168.189.131 32856   192.168.189.132 443     44      192.168.189.132 -       9       9
9d1f095f5d      44,PAD-SNI-VER-CCS-PDMD-ICSL-MIDS-CFCW-SFCW
#close  2020-02-18-03-39-01
deckerl@deckerl-seconion:~/merlin_bad$ cat gquic.log | bro-cut cyu cyutags
97fc27484cbf1ec48be2f99d1f095f5d        44,PAD-SNI-VER-CCS-PDMD-ICSL-MIDS-CFCW-SFCW
deckerl@deckerl-seconion:~/merlin_bad$
```

Figure 13 - Zeek - Salesforce Plugin - Merlin C2

Lee Decker, deckerl@gmail.com

3.5. Wireshark

Wireshark needs to be a version greater than 3.0.3 to help decode the latest version of QUIC packets. Wireshark 3.05 was installed on Kali to examine the researcher's packet captures. QUIC profiles from Cellstream were tested but did not provide additional information with the researcher's setup.

When examining the known "bad" packets from the Merlin C2 capture, only the initial client hello packets are unencrypted and provide information like what has been extracted by the Zeek Salesforce plugin.

The data field contains multiple tags, including version, encryption algorithm, padding, and many others. These fields help establish how the server and client will handle the traffic stream.

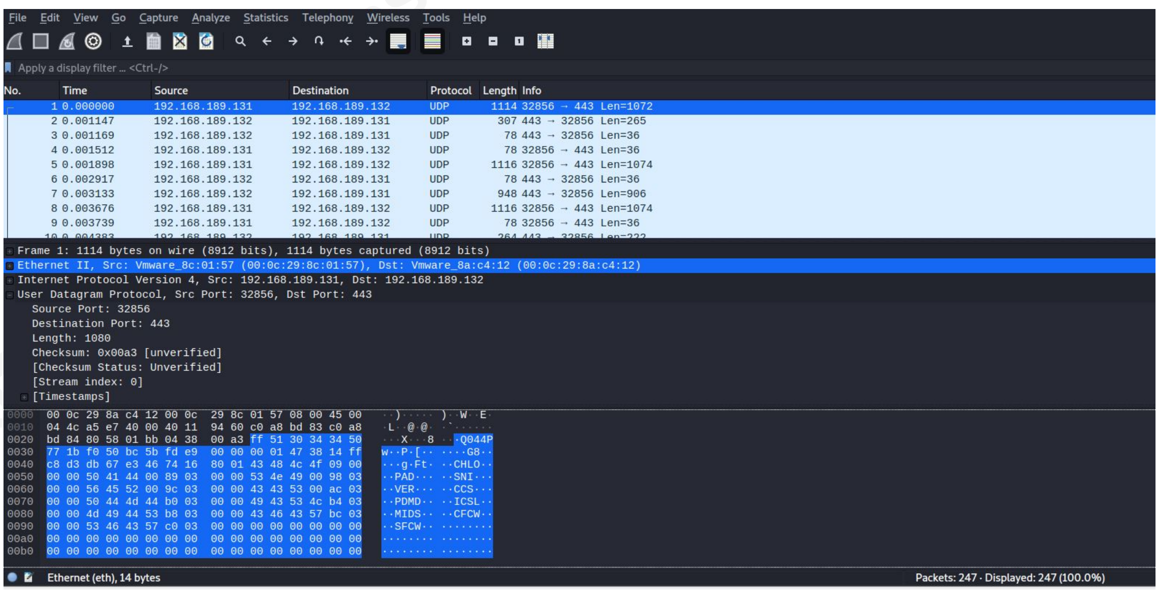

Figure 14 - Wireshark - Merlin C2 packet

Using Wireshark to examine the "good" traffic from Google & YouTube yields DNS traffic, HTTP and HTTPS traffic, andG QUIC traffic. GQUIC appears sporadically as some, but not all Chrome sites use it. The client *helo* packets contain similar information to the "bad" traffic.

Lee Decker, deckerl@gmail.com

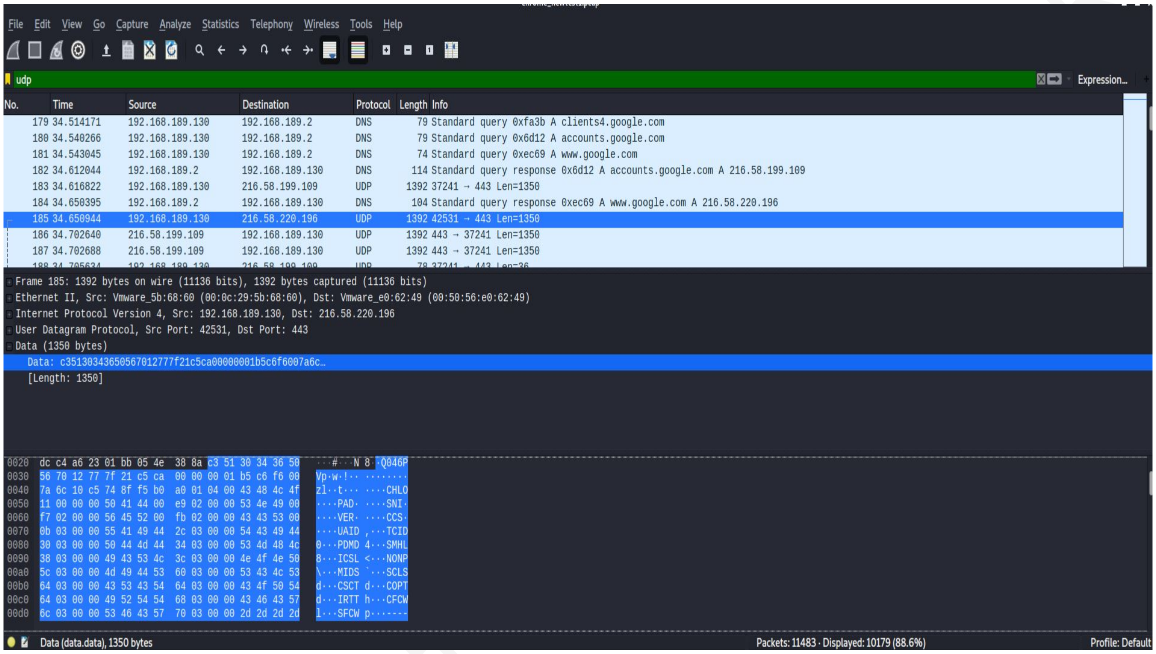

Figure 15 - Wireshark - Chrome packet

4. Future Research

While collecting data for this research, challenges arose with the third-party plugins and the latest version of Zeek. The researcher was unable to get either Bro-quic or the GQUIC plugin to work with the current version of Zeek.

GQUIC is an evolving standard, and the Salesforce plugin has supported up to the current version of Q046. Additional work may be needed to update the plugin as well as any related scripts as the protocol continues to evolve.

The Salesforce plugin was unable to process QUIC traffic used by Firefox. Further research is required to adapt the plugin to QUIC, as it is the newer standard seen from non-Google browsers.

As with traditional TCP encrypted traffic, analysis of the connection information, and meta-data is key to finding malicious traffic. Additional research will be needed to adapt reputation, beaconing, and other methods to QUIC/GQUIC traffic analysis.

As Microsoft moves QUIC beyond HTTP traffic, incorporating it into the SMB protocol in future builds of Windows (Pyle, 2020), additional analysis and tools will be needed.

4.1. Developing a proof on concept Zeek script

Zeek is a collection of scripts and can be extended and customized as needed. The Salesforce GQUIC plugin adds four new events. Gquic_packet, gquic_client_version, gquic_helo, and gquic_rej. It also adds two new constants: PublicHeader and HelloInfo.

The researcher will create a script using these elements to more reliably identify Merlin C2 traffic when it is mixed in with legitimate traffic.

After looking at the various data sets gathered, two identifying features of malicious GQUIC traffic, like Merlin C2 are relevant.

1) Malicious traffic showed up far less frequently than legitimate traffic.
2) Google's GQUIC tags were consistent in the limited sample set.

From this premise, the pseudo-code followed: Building on the Salesforce plugin, when Zeek identifies a new GQUIC packet, it adds the ip address (id_orig.h), and tag set (CYU) tag to an array and a counter starts. The counter increments the next time the same ip and CYU appear. By looking at both variables, we can account for an infected machine that is generating both legitimate and malicious packets.

The researcher can then filter the common Google tags or sort for least seen tags, which could aid an investigator in identifying infected machines.

A sample of the Merlin C2 packet capture was merged with a Google Chrome packet capture using mergecap. The script uses this packet capture to count each combination ip address and GQUIC tag.

4.2. Zeek Script Proof of Concept

```
@load base/protocols/conn
@load base/protocols/http
```

Lee Decker, deckerl@gmail.com

```
## There is likely a far more elegant way to do this

type GQUIC_Hosts:record {
          host_ip: addr;
          gquic_tags: string;
       number_seen: count;

};

global profiles: table[string] of GQUIC_Hosts;
global x = 1;
global start = 1;


event gquic_hello(c:connection,
is_orig:bool,hdr:GQUIC::PublicHeader,hello:GQUIC::HelloInfo)
{
   local packet_match = 0;
   if (start == 1) {
          profiles[c$uid] = [
          $host_ip=c$id$orig_h,
          $gquic_tags = hello$tag_list,
          $number_seen = x
          ];
          start +=1;
}
### Test new packet to increment counter if source and tags match ##
     for (keys in profiles) {
        if (profiles[keys]$host_ip == c$id$orig_h && profiles[keys]$gquic_tags ==
hello$tag_list) {
               local y = 1;
               y = profiles[keys]$number_seen;
               y +=1;
               print "match found";
               print c$uid;
               profiles[keys] = [
                      $host_ip = c$id$orig_h,
                      $gquic_tags = hello$tag_list,
                      $number_seen = y
                      ];
            packet_match = 1;
          }
## for loop
     }

            if (packet_match == 0) {
```

```
                print "add new packet";
                print c$uid;
                profiles[c$uid] = [
                $host_ip = c$id$orig_h,
                $gquic_tags = hello$tag_list,
                $number_seen = 1
            ];


    }

#   print c$uid;
#   print c;
#   print x;
#   x+=1;


# print start;
# print x;
 print profiles[c$uid];
## GQUIC event
}
```

5. Conclusion

As hypothesized, most open-source tools tested provided minimal or no information on QUIC/GQUIC packets, and as a result, could not detect malicious versus non-malicious packets. Third-party plugins for Zeek proved to be the most valuable at extracting data from the non-encrypted GQUIC packets. The researcher was able to create an initial proof of concept Zeek script to help identify the malicious packets among the legitimate Google traffic, using the work previously done by Salesforce.

Security professionals will better be able to defend their networks in the future from malicious GQUIC traffic, by understanding the current state of the security tools, where QUIC is going, and how tools like Zeek plugins perform.

However, the recommendation of the researcher would be to block and monitor GQUIC/QUIC traffic from enterprise networks until further tools develop. These protocols work in the consumer space but create challenges in the enterprise security space.

Lee Decker, deckerl@gmail.com

References

Adams, Tommy. (2018, August 7th). Processing experimental protocols against IDS.

Retrieved from URL https://www.sans.org/reading-room/whitepapers/detection/processing-experimental-protocols-ids-38565

Liebert, Etienne (2018, June 22nd) How Google's QUIC Protocol Impacts Network Security and Reporting.

Retrieved from URL https://www.fastvue.co/fastvue/blog/googles-quic-protocols-security-and-reporting-implications/

Lynegar, J / Thomson M. (2019, December 16th). QUIC:A UDP-Based Multiplexed and Secure Transport.

Retrieved from URL https://quicwg.org/base-drafts/draft-ietf-quic-transport.html

Maynard, Jason. (2018, April 22nd). Cisco Firepower Threat Defense 6.2.3: Block QUIC force TCP TLS/SSL Decryption.

Retrieved from URL https://www.youtube.com/watch?v=QYgE7TDoWlM

Niroshan, Anuradaha. (2017,Oct 1st). Understanding the QUIC wire protocol.

Retrieved from URL https://medium.com/@nirosh/understanding-quic-wire-protocol-d0ff97644de7

(2017, October 2nd). How to Block QUIC.

Retrieved from URL http://community.lightspeedsystems.com/documentation/how-to-block-quic/

Ghedini, Alessandro. (2018, July 26). The Road to QUIC.

Retrieved from URL https://blog.cloudflare.com/the-road-to-quic/

Google QUICK analyzer/detector for Bro.

Retrieved from URL https://github.com/corelight/bro-quic

Pearce, Catherine / Vincent, Carl. HTTP/2 & QUIC. Teaching Good Protocols to do bad things.

Retrieved from URL https://www.blackhat.com/docs/us-16/materials/us-16-Pearce-HTTP2-&-QUIC-Teaching-Good-Protocols-To-Do-Bad-Things.pdf

Pyle, Ned. (2020, March 3rd). SMB over QUIC: Files Without the VPN.

Retrieved from URL https://techcommunity.microsoft.com/t5/itops-talk-blog/smb-over-quic-files-without-the-vpn/ba-p/1183449

Ruth, Jan, / Poese, Ingmar / Dietzel, Christoph / Hohlfeld, Oliver. (2019, February 24th). A First Look at QUIC in the Wild.

Retrieved from URL https://arxiv.org/pdf/1801.05168.pdf

Shah, Urvekkumar Chandravadan. (2018). Flow-based Analysis of QUIC Protocol.

Retrieved from URL https://is.muni.cz/th/q69ug/Shah_Thesis.pdf

Tuyl, Russel Van. (2018, July 31st). Merlin Adds Support for the QUIC protocol.

Retrieved from URL https://medium.com/@Ne0nd0g/merlin-adds-support-for-the-quic-protocol-ee5f8a1e8955

Villarreal, Ryan (2019, January 15th). Merlin the (C2) Wizard!

Retrieved from URL https://bestestredteam.com/2019/01/16/merlin-the-c2-wizard/

Yu, Caleb. (2019. August 13th). GQUIC Protocol Analysis and Fingerprinting in Zeek.

Retrieved from URL https://engineering.salesforce.com/gquic-protocol-analysis-and-fingerprinting-in-zeek-a4178855d75f