Model and implementation of user activity tracking utilizing the TLS Client Hello's `server_name` extension

Mauro M.*

*Independent Researcher, hello@maurom.dev

Abstract

This paper puts forward a feasible, non-intrusive, method of tracking user activity using TLS's Client Hello section of a handshake (specified in the TLS protocol [1]), namely the `server_name` extension.

This method can provide an attacker with relevant information regarding patterns and services utilized inside of the target network, further expanding their understanding of the attack surface, potentially, serving as a tool to determine the timing of an attack or, even, provide an attacker with knowledge of a point of entry to a given system.

It is noteworthy that TLS Client Hellos will be encrypted in a future version of the TLS protocol, rendering this method infeasible in fully updated networks. [2]

1 Method Description

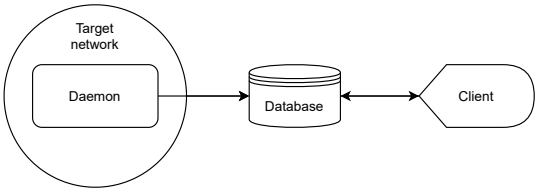

Figure 1. Overview of the proposed method

1.1 Daemon

The daemon handles packet sniffing and filtering, it should be designed in a way that it can act independently within the target network minimizing the number of requests it makes to avoid detection by firewalls. Additionally, the daemon must filter out unnecessary packets, depending on the use-case. It must only commit to the database purely necessary information. For example, if the use-case was to track a user's browsing history, to distinguish patterns, like working hours, domains including the terms "cdn", "source", "API" should not be sent to the database as these are not required. Finally, the daemon should submit the filtered packets to the database. The amount of data submitted, shall again, vary by use-case, but should, in most use-cases, include the MAC address of the source, the URL requested (The value of `server_name` in the `server_name` extension in the sniffed handshake), and timestamp as to when the request was made.

1.2 Data store

This can constitute any data store that can handle a high throughput of data. The proposed model does not require the data store to return data in any special format or preform any operations on the data it receives.

1.3 Client

The client should establish a bi-directional connection with the data store to be able to display the data store's data and allow the end-user to perform operations on the aforementioned data. Furthermore, the end-user should be able to group, and name groups of MAC addresses, which ultimately represent a certain client that connects with a pseudo-random MAC address. Determining what group, if any, a MAC address should be in is not covered by this paper.

2 Method Implementation

The implementation presented herein represents one, of many use-cases of the aforementioned model. The goal of presenting this implementation is to show the real-world feasibility of the model. It concerns simply collecting a user's traffic and displaying it to an attacker, this can be used to determine patterns inside the target network, for example, understanding when a certain device goes offline, consistently, can represent a break for the owner of that device.

2.1 Daemon

The daemon is implemented in Python utilizing the `scapy` module and a wrapper for connecting to the datastore, in this case, a PostgreSQL database.

2.1.1 Packet Sniffing

First, a filter is required to only sniff the client hello section of the handshake, since scapy allows BPF style filters [3] the filter in Figure 2 is used.

```
tcp dst port 443 and (tcp[((tcp[12] &
0xf0) >> 2)] = 0x16 && (tcp[((tcp[12] &
        0xf0) >> 2)+5] = 0x01))
```

Figure 2. BPF filter that filters TLS Client Hellos

This filter verifies that the first byte of the TLS header is 22 and that the sixth byte is 1 this specifies a TLS Client Hello according to the current standard [1]. Applying that filter to scapy's `AsyncSniffer` a list of packets is obtained (we shun scapy's `PacketList` because we do not require any of the additional functionality provided by it). The list is then passed to the filter.

2.1.2 Filtering

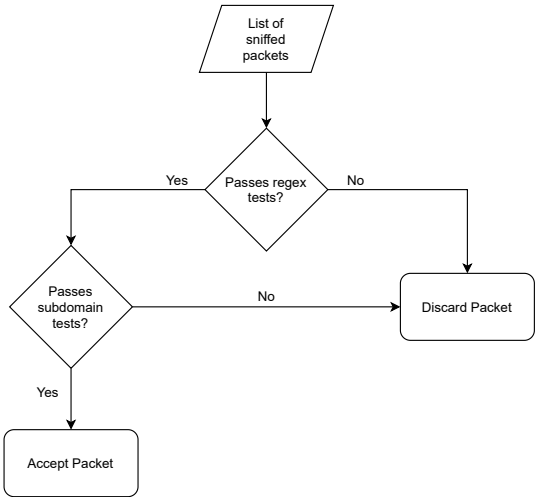

Figure 3. Overview of the two-step filter

The proposed implementation uses a two-step filter described in Figure 3. It consists of a regex check and a sub-domain "validity" check. Together, in testing, these caught 99.9% of domains from a known list of CDN domains ($N = 14300$). The list of filtered domains is then commited to the data store.

2.1.2.1 Regular Expression Check

The regular expression check iterates over the constituents of the array that contains the sniffed packets and checks them against a list of regular expressions which check for strings utilized in common content serving domains.

```
^dist\.|^media\.|^assets\.|^videos\.|
^push\.|^js\.|^widget\.|^plugins\.|^i\.|
^img\.|^fonts\.|^controllers\.|^use\.|
^static\.|^api\.|^apis\.|^cdn\.|^mirror
```

Figure 4. Regular expression used to filter the majority of domains

The main regular expression that checks for these types of sub-domains is represented in Figure 4. In testing, the regular expression alone caught 48% of domains from a known list of CDN domains ($N = 14300$)

2.1.2.2 Sub-domain Check

The sub domain check iterates over an array consisting of the domain split by `"."` it then removes `www` from the array, if it exists. It then removes the last two elements, respectively, the domain name and gTLD. It then proceeds to iterate through the remaining elements of the array and checks if their length is ≤ 3 or if it contains a number. In testing this check alone caught 28.7% of domains from a known list of CDN domains ($N = 14300$).

```
import re

for domain in domains:
   if not re.search(re.compile(regex,
        re.DOTALL), domain)):
       list_ = domain.split(.)
       list_ = list_[:len(list_) - 2]
       list_.remove('www') if 'www' in list_
           else list_

       for element in list_:
          if not (len(element) <= 3 or
              re.search(r"[0-9]", element) or
              element.count('-') >= 2):
             # Accept the domain
```

Figure 5. Example Python implementation of the sub-domain check (adapted for clarity)

2.1.3 Data store

The proposed implementation utilizes a PostgreSQL database which holds the pertinent information from the filtered packets for easy access by the client. It contains two tables `packets` and `groups`, which hold, respectively, the sniffed packets and groups of MAC addresses with their respective name, for access and modification by the client.

2.1.4 Packets Table

In the proposed implementation the `packets` table stores the MAC address of the device that initiated the handshake, the value of the `server_name` extension and a UNIX epoch timestamp [4] with the time at which the handshake was initiated. The schema represented in Figure 6 was used which yields a database like the one represented in Table 1.

```
CREATE TABLE IF NOT EXISTS Packets(
                  mac TEXT,
                  url TEXT,
                  ts TIMESTAMP
                  );
```

Figure 6. Create table query to show the table's schema

MAC	URL	Timestamp (ts)
59:9D:B9:EE:2D:E8	server.local	1609459200

Table 1. Representation of the `packets` table

2.1.5 Groups Table

In the proposed implementation the `groups` table stores named groups of MAC addresses where multiple MAC addresses can be related to a single name. The schema represented in Figure 7 was used which yields a database like the one represented in Table 2.

```
CREATE TABLE IF NOT EXISTS Groups(
                            addresses TEXT[],
                            name TEXT
                           );
```

Figure 7. Create table query to show the table's schema

MACs	Name
{59:9D:B9:EE:2D:E8}	IT Admin

Table 2. Representation of the `groups` table

2.2 Client

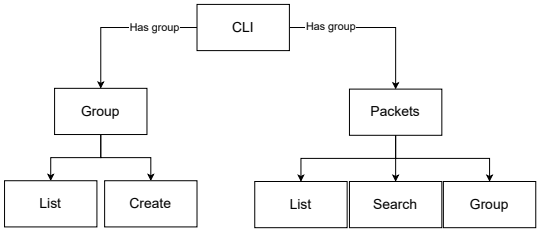

Figure 8. Overview of CLI's command hierarchy

The proposed implementation includes a CLI client which allows the end user to access and manipulate data stored in the data store. For simplicity's sake the CLI was created using the `click` Python package

As can be observed in Figure 8 the CLIs commands are grouped categorically. In the group category the user can list and create groups, whereas in the packets category the user can list, search for a specific MAC address or return all requests belonging to a group.

Additional Information and Declarations

Competing Interests

There are no competing interests

Source code availability

The source code along with supporting documents can be found at `https://github.com/MM-coder/tls-client-hello-activity-tracking`

References

[1] E. Rescorla, "The transport layer security (tls) protocol version 1.3," 08 2018.

[2] K. Oku, C. Wood, E. Rescorla, and N. Sullivan, "Tls encrypted client hello," 10 2020.

[3] S. Mccanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," pp. 259–269, 1992.

[4] T. O. Group, "General concepts," 2001.