CS790 - Privacy Enhancing Technologies
Assignment 2 - Tor Detection

Shantanu Welling

April 2025

Contents

Overview

This plugin is a Lua-based post-dissector for Wireshark that detects whether the system is communicating over the Tor network. It uses known characteristics of the Tor TLS handshake and publicly known relay IP addresses to efficiently identify suspected Tor-related traffic. Upon detection, it adds a label and coloring to the packet list for easy visualization.

Approach

The plugin performs detection using a combination of two techniques:

- **IP Matching:** It checks if either the source or destination IP matches a known list of Tor relays (loaded from a local `.txt` file which is dynamically manually updated at runtime).
- **TLS Fingerprint Matching:** It inspects TLS Client Hello packets for the specific sequence of 18 cipher suites used by Tor clients, totaling 36 bytes.

If either method identifies potential Tor traffic, the plugin marks the packet as suspicious.

Implementation Details

- **Postdissector Protocol:** Defined using Lua's `Proto()` and registered via `register_postdissector()` to scan all packets post-dissection.
- **Cipher Suite Matching:** The plugin parses the `tls.handshake.ciphersuites` field which returns a `FieldInfo` object containing the the raw byte information about the cipher protocols' code and validates the presence and order of 18 known Tor cipher suites.
- **Relay IP Lookup:** The plugin reads a text file of Tor relay IPs and flags any traffic with matching IPs.
- **Packet Annotation:** If detected, the plugin adds a "Tor!" label in the custom protocol tree and appends a note to the `Info` column.

Wireshark Integration

- A custom column **Tor Status** displays "Tor!" for detected packets.
- The `pinfo.cols.info` field is updated to reflect detection inline.
- Optional: The plugin can be paired with a display filter and color rule using `tor_detect.status == "Tor!"` for enhanced visibility.

Performance Considerations

- The plugin avoids false positives by verifying both IP and cipher suite patterns.
- It stores previously flagged Tor relay IPs (flagged using TLS handshake cipherspec fingerprinting) to optimize future detections. In case it sees a packet which was previously flagged (either as source or destination), then it flags those packets as Tor too.

Heuristics Considered (but rejected)

While designing this plugin, I explored several heuristics to identify Tor traffic based on known behavioral patterns. Two of the prominent ones were:

- **Known Tor Port Numbers:** Tor relays commonly use certain ports such as:
  - **443** – TLS/HTTPS (used to blend in with regular web traffic)
  - **9001** – Default port for Tor relay traffic
  - **9050** – Default Tor SOCKS proxy port
- **Packet Size Heuristics:** I hypothesized that specific Tor-related packets—especially TLS Client Hello messages or relayed Tor application data—might consistently have fixed payload sizes that could be used as signatures for detection.

However, upon further analysis of captured traffic in Wireshark:

- Tor traffic did not always use the expected port numbers. For instance, I observed traffic on port **8080**, which invalidated port-based detection as a reliable method. This behavior is likely due to Tor's attempt to evade censorship and blend with other traffic.

- Packet size analysis turned out to be unreliable. Although some Tor-related handshake packets had consistent sizes, several unrelated (non-Tor) packets matched the same lengths. This resulted in a high number of false positives and made packet size a poor standalone heuristic.

Conclusion: These heuristics were ultimately discarded in favor of more reliable and unique identifiers such as the specific cipher suite fingerprint used in TLS Client Hello messages and known Tor relay IP addresses.

Supplementary Note: Dissection Mechanisms in Wireshark

In Wireshark, dissectors are scripts or plugins that analyze packets and extract protocol-specific information to display in the GUI. There are two types of dissectors:

- **Normal Dissectors:** These are responsible for parsing specific protocols as packets are captured or opened. For example, the TLS dissector parses TLS handshakes, and the HTTP dissector parses HTTP headers. Normal dissectors are protocol-specific and get called in the order defined by Wireshark's protocol stack. They also need to be associated with a specific port (kinda like a protocol-port tuple unique identifier).
- **Postdissectors:** These are Lua plugins that run *after all standard dissectors* have finished processing a packet. They analyze the final results of dissection and can extract information across different protocols.

  Postdissectors are used when:

  - You want to add a layer of analysis that relies on multiple protocols and ports.
  - You are correlating information across packets or protocols or ports.
  - You are not dissecting a new protocol but enhancing Wireshark's view (e.g., labeling suspicious packets, computing latency, etc.).

In this plugin, a *postdissector* was the ideal choice because we:

- Leverage existing dissectors like `tls` to extract fields such as cipher suites.
- Analyze higher-level behavior (e.g., is this packet Tor-related?) rather than implementing a new protocol.
- Add metadata (like "Tor!" labels) across packets without interfering with native dissector order or performance.