Quick&Dirty Heartbleed Detection Using Bro

event connection_state_remove(c: connection) {
    if (port_to_count(c$id$resp_p) != 443) return;
    if (c$conn$orig_bytes < 200) return;
    if (c$conn$orig_bytes > 300) return;
    if (c$conn$resp_bytes < 33000) return;
    if (c$conn$resp_bytes > 66000) return;
    if (c$conn$duration < 50 msec) return;
    if (c$conn$duration > 800 msec) return;
    # Log...
    # ......

Jakub Svoboda, Jan Vykopal
Masaryk University
Introduction

- Heartbleed
  - OpenSSL vulnerability publicly disclosed on April 7
- Bro Network Security Monitor
  - Open-source DPI-capable event-driven extensible IDS

- Heartbleed detection using Bro implemented within a few hours
- Bro enables going back in time
Hearbleed Vulnerability

- Client sends a heartbeat request with incorrect information about the requested size.
- OpenSSL does not perform bounds check and blindly copies data from the request to the reply based on the client-provided information.
- Client can send a short request, claiming to be a long request.
- OpenSSL voluntarily copies its own memory into the reply, which can disclose the private key.
What is Bro?

- Capable of deep packet inspection - unlike NetFlow.
- Extensible using the Bro scripting language.
- Event-driven – packets are transformed into events with context; events can be handled by scripts.
- For more details see http://www.bro.org
Bro Architecture

- NIC
- PCAP
- Packet decoder
- Scripting engine
- Event generator
- logs
- execute
- files
Fixing Servers

- Task: find vulnerable servers on our network and get them fixed.
- System admins(?) did a terrific job testing all our servers with the ssltest.py tool.
- We analyzed the traffic they generated, i.e. passively detect vulnerable servers, and notify responsible admins.
Hearbleed Attack Profiling

- A python tool started circulating soon.
  - Originally available at http://s3.jspenguin.org/ssltest.py, then taken down.
- We used it, analyzed attack attempts, and created a profile of successful attack.
- We found out that the request and reply are of a certain size.
- The profile is purely flow-based. (Only metadata about the connection are processed.)
- Our method uses **passive detection** - unlike other tools
  - e.g., ssltest.py and https://github.com/titanous/heartbleeder.
Successful Hearbleed Attack Profile

- Our Bro code detects the successful attack.
- The displayed sizes are on L4.

```c
event connection_state_remove(c: connection) {
    if (port_to_count(c$id$resp_p) != 443) return;
    if (c$conn$orig_bytes < 200) return;
    if (c$conn$orig_bytes > 300) return;
    if (c$conn$resp_bytes < 33000) return;
    if (c$conn$resp_bytes > 66000) return;
    if (c$conn$duration < 50 msec) return;
    if (c$conn$duration > 800 msec) return;
    # Log...
    # ......
```
Going Back In Time

- Bro Input Framework allows reading of existing logs.
- The logs contain all the information we need.
- A shell script & another Bro script allowed retrospective detection based on the very same criteria.
- Bro had already been running and producing logs => we could go a week back.
- Luckily, no successful attack prior to April 7 was detected.
Shortcomings

• Our method is simple, but...

• cannot detect subtle attacks where the attacker requests just a few bytes more than the request.
  – The reply/request size ratio then merges with other traffic.

• On the other hand, it's great for discovery of vulnerable servers on your network.
  – Finds vulnerable servers script kiddies already found.
  – Finds the most probable successful attacks (ssltest.py).
Summary

• **Quick** prototype **implementation** and deployment using **Bro** IDS.

• **Passive** identification of attacked vulnerable servers.

• Legal aspect: **someone else** attacked the servers, we only monitor the generated traffic.