Design and implementation of a firewall system for working in unsafe environments

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Financial Technology and Computing
main track

presented by
Bin Yong

under the supervision of
Prof. Student's Advisor
co-supervised by
Prof. Student's Co-Advisor

September 2023

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Bin Yong
Lugano, Yesterday September 2023

Abstract

Design and implementation of a firewall system based on Raspberry Pi. The firewall monitors network activities and encrypts some plaintext traffic. It helps device owners take back control of their own devices. It is designed for someone who needs to work in unsafe environments. The system sacrificed some compatibility to pursue better security but still achieved a balance between security and convenience. A sensitive target, like an investigative journalist, could be a potential user of this device.

Acknowledgements

Contents

Contents vii

List of Figures xi

List of Tables xiii

1 Introduction 1

2 Hardware 3
- 2.1 Theory of the cost . . . 3
- 2.2 Market research on commercial firewalls . . . 3
- 2.3 Choosing hardware . . . 5
- 2.4 Deployment and operating system . . . 5

3 Packet filtering and forwarding 9
- 3.1 Design . . . 9
- 3.2 Implementation . . . 10

4 Dynamic Host Configuration Protocol (DHCP) 11
- 4.1 Introduction . . . 11
  - 4.1.1 The issue . . . 11
- 4.2 Design . . . 11
- 4.3 Implementation . . . 11

5 Domain Name System (DNS) 13
- 5.1 Introduction . . . 13
- 5.2 Design . . . 13
- 5.3 Implementation . . . 13
  - 5.3.1 Outgoing requests . . . 13
  - 5.3.2 Inside firewall . . . 13
  - 5.3.3 Incoming requests . . . 14

6 Transport Layer Security (TLS) 15
- 6.1 Management of certificates . . . 15
  - 6.1.1 Introduction . . . 15
  - 6.1.2 Design . . . 16
  - 6.1.3 Implementation . . . 16

6.2 Mismatch between the changing best current practice and the unchanged implementation . . . 16
6.2.1 Problem . . . 16
6.2.2 Solution . . . 16
6.3 Proxy . . . 16
6.4 Process view of traffic . . . 20

7 Plaintext connections **21**
7.1 HTTP . . . 21
7.1.1 Implementation . . . 21

8 Date time synchornization **25**
8.1 Introduction . . . 25
8.1.1 The issue . . . 25
8.2 Design . . . 25
8.3 Implementation . . . 26

9 Behavioral simulation **27**
9.1 Introduction . . . 27
9.2 Design . . . 27
9.3 Implementation . . . 27
9.3.1 Action files . . . 28
9.3.2 HoodExecutor . . . 28
9.3.3 Browser . . . 28
9.3.4 Hood action script . . . 29

10 Dispatcher and firewall rules **31**
10.1 Initial state of firewall rules . . . 31
10.2 On udev add event . . . 31
10.3 On system startup . . . 31
10.4 On NetworkManager-dispatcher pre-up event . . . 32
10.5 On NetworkManager-dispatcher up event . . . 33
10.6 On NetworkManager-dispatcher down event . . . 33

11 Installation **35**
11.1 Checking target system . . . 35
11.2 Disable wireless . . . 36
11.3 Disable GPU . . . 36
11.4 Other configurations modified . . . 37

12 Showing network activities **39**

13 Reasons of not to use **41**
13.1 IPv6 . . . 41
13.2 WiFi . . . 41
13.3 Bluetooth . . . 41
13.4 GPU . . . 41
13.5 Honeypot . . . 41

14 Conclusions **43**
14.1 Possible improvements for future 43
14.1.1 LSM and seccomp 43
14.1.2 Compile time hardening 43
14.1.3 Network stack fingerprinting 43
14.1.4 Further use to libcomposite 43

15 The attacks encountered during the time I was working on this thesis **45**
15.1 Malicious hardwares 45
15.2 Sounds 45

A Simple scripts and services created by firewall **47**
A.1 before-network.service 47
A.2 hood-network-services.service 47
A.3 domain_blacklist.txt 47
A.4 ip_subnet_blacklist.txt 47
A.5 nftables.conf 47

Glossary **51**

Bibliography **53**

Figures

2.1 Hardware deployment view 6

6.1 Process view of TLS proxy 17
6.2 Process view of TLS traffic from user 20

7.1 Process view of OCSP over HTTP traffic from user 22
7.2 Process view of normal HTTP traffic from user 23

9.1 Logic view of hood-actor.py 30

Tables

1.1 Some facts 1

2.1 The first page of search results of "hardware firewall" on amazon.de. Captured at 5 P.M. of December 26th, 2023, Lugano 4
2.2 Statistics of table 2.1 5

6.1 Allowed cipher suites values 18

8.1 Command line arguments of hood-timesync.py 26

9.1 "type" field of action file objects 28
9.2 "properties" object of action file 28
9.3 "task" object of action file 28

10.1 Firewall rules to be added for LAN ports 32

11.1 Command line arguments of install.sh 35
11.2 Configurations added to target to disable wireless 36
11.3 Files deleted from target to disable wireless 36
11.4 Configurations modified to target to disable GPU 36
11.5 Files deleted from target to disable GPU 36
11.6 Configurations modified to target to disable GPU 37

12.1 Log locations of network services. 39

Chapter 1

Introduction

With the development of surveillance and management tools, people are losing controls of their own devices. Here are some facts that general publics may underestimated their impcats:

1. Any device with a microphone (sometimes a speaker may do the same) can 'hear' sounds people are making around.
2. Any device with a camera can 'see' what is happening around.
3. Any device connected to internet can upload what they know and receive commands from remote.
4. Any device may have been hacked and functioning as not what they were designed.
5. Some devices are designed with surveillance functionality.

Table 1.1. Some facts

Based on the facts above, some devices are naturally born with surveillance abilities. For example, smartphones and smart speakers are both able to hear voices and communicate with the internet. Most people do not have the skills to investigate and monitor what they actually uploaded to a remote server. To make them able to respond to "Hi, Siri." or "Hey, Alexa.", those devices have to process every sound they receive when they are quiet. Some are even worse: Intel Management Engine (Intel ME) or Intel Converged Security and Management Engine (Intel CSME) and Intel Active Management Technology (Intel AMT) for example. It's obvious that they are made for management, a function that is unnecessary to most individual owners. According to their official documents. "its power stats are independent" means that simply powering off the computer could not stop this management. "up before the main operating system" means that most users could not even notice it and could do nothing about it, and if something goes wrong, people cannot fix it by simply reinstalling operating system. It can access "LAN/Wireless LAN", check table 1.1. "present on most Intel platforms, including client consumer and commercial systems, workstations, servers, and IoT (Internet of Things) products." means there is almost no escape in their ecosystem. The details of its documents also show the capability of running remote commands (Serial Over Local-Area Network), the capa-

bility of accessing local storage "Universial Serial Bus Redirect", "Keyboard, Video and Mouse" remote control over network.

So many things are out of the control of the owners of the devices. People need something to restrict the management and to show what is happening to the network. The goal of this work is to help with that. It aims to increase the chance of survival of the user from hacking, eavesdropping, digital fingerprinting, and unwanted managements. It uses a screen to display selected real-time internet activities. The system limits network activities and applies encryption standards.

Using hardware firewalls has several advantages over software firewalls. Firstly, a hardware firewall runs completely independent to high-risk user devices, where unsafety thrives: CPU with possible hidden instruments, firmware with possible infections, motherboards with doubtable proprietary management technologies, and applications with possible surveillance or unpublished vulnerabilities. Second, it brings convenience to the user: the configuration of this portable device restricts everything behind it, so people do not need to do the time-consuming configuration work on different software and operating systems repeatedly. Third, it brings possibilities: Without the help of a firewall, untrusting a built-in root certificate on mobile devices is hard and the methods with root or jailbreak involved may void the warranty. With the help of a firewall, managing certificates is no longer a problem.

Chapter 2

Hardware

2.1 Theory of the cost

Commercial firewalls are usually more expensive than development boards that contain the same level of hardware functionality. On one hand, It's acceptable because they provided additional value through their software and services and all of the development and maintenance of the product, the service, and the company itself are costing money. On the other hand, it also indicates that an open-source solution would reduce the lower bound of the cost of accessing a hardware firewall.

2.2 Market research on commercial firewalls

Despite the firewall implemented in this paper is not a general-purpose firewall, the difference in cost between market products is still worth knowing. From tables 2.1 and 2.2, the prices of hardware firewalls are centered around 250.74 EURO and most of them are more expensive than the price of a Raspberry Pi 4B (68.79 EURO) selling at the same website. The only exception is TP-LINK ER605. Its price is 55.5 EURO which is still higher than the price of a Raspberry P 3B on the same website. After checking the details of this exception, its central management design and the corresponding cloud-based controller both show that it is designed for a totally different threat model. Its exceptional price could be explained as the device is under the control of the company. However, spending money on a hardware firewall usually implies the reluctance to give control of devices to a company. To sum up, using Raspberry Pi would cost less than usual general-purpose firewalls.

Name	Price
TP-LINK ER605 5 Port Dual/Multiple WAN VPN Router (up to 4 Gigabit WAN Ports, Highly Secure, Omada SDN, Central Management, Intelligent Monitoring, Firewall) Black, Ideal for Office Network	€55.49
Micro Firewall Appliance, Mini PC, Pfsense Plus, Mikrotik, OPNsense, VPN, Router PC, Intel N4505, HUNSN RS41k, AES-NI, 4 x 2.5GbE I226-V, Console, Type-C, HDMI, DP, SIM Slot, 4G RAM, 32G SSD	€188.99
ZyXEL ZyWALL 350Mbps VPN Firewall, Recommended for up to 10 Users [USG Flex 50]	€259.00
Micro Firewall Appliance, Mini PC, pFsense Plus, Mikrotik, OPNsense, VPN, Router PC, Intel Alder Lake-N 12th Gen N100, HUNSN RJ42, 4 x 2.5GbE I226-V, 2 x HDMI, DP, TF, Type-C, 8G DDR5 RAM, 128G SSD	€279.99
KingnovyPC Firewall Micro Appliance, 4 Port i226 2.5GbE LAN Fanless Mini PC N5100, 2* DDR4, HDMI, DP, RJ45 COM, 4*USB Gigabit Ethernet AES-NI VPN Router Openwrt Barebone	€146.53
New J4125 Quad Core Firewall Micro Appliance, Mini PC, Nano PC, Router PC with 8G RAM 128G SSD, 4 RJ45 2.5GBE Port AES-NI Compatible with Pfsense OPNsense	€279.00
Cisco Meraki Go - 5 Port Security Gateway - Power Supply to EU Standard GX20-HW-EU	€102.53
Protectli Vault FW4B - 4 Port, Firewall Micro Appliance/Mini PC - Intel Quad Core, AES-NI, 4GB RAM, 32GB mSATA SSD - Compatible with pfSense/OPNsense etc	€284.87
Firewall Hardware, Pfsense, OPNsense, Mikrotik, VPN, Network Security Appliance, Router PC, Intel Atom N2600, HHUNSN RS31, 4 x Intel Gigabit LAN, 2 x USB, COM, VGA, Fan, 4G RAM, 32G SSD	€221.99
Firewall Mini PC with 2.5G Gigabit LAN, Firewall Micro Appliance Celeron J4125,4 x I225 Gigabit LAN,Fanless Mini PC AES-NI,12V,WiFi,HDMI,RS232 COM,USB 3.0,8GB RAM/128GB SSD	€265.00
FORTINET FortiGate 40F Hardware - Next Generation Firewall Protection and Security	€520.90
Micro Firewall Appliance, Mini PC, VPN, Router PC, Intel Alder Lake-N 12th Gen N100, HUNSN RJ42, 4 x 2.5GbE I226-V, 2 x HDMI, DP, TF, Type-C, Barebone, NO RAM, NO Storage, NO System	€288.99
Cisco Systems Go Router Firewall Plus | Cloud Managed | VPN | Cisco [GX50-HW-EU], White	€318.09
Zyxel Secure Cloud-Managed Router/Firewall with AXE5400 Tri-Band WiFi Subscription Free Network Security, Managed via Nebula APP/Ideal for Small Offices/Small Branches. [SCR 50AXE]	€173.5
Micro Firewall Appliance, Mini PC, VPN, Router PC, Intel Alder Lake-N 12th Gen N100, HUNSN RJ46, 6 x 2.5GbE I226-V, 2 x HDMI2.1, TF, Type-C, Barebone, NO RAM, NO Storage, NO System	€242.99
HSIPC 11th Gen i3 1115G4 Firewall Micro Appliance, Mini PC, Nano PC, Router PC (16G 256G) With 6 RJ45 2500M, AES-NI, HDMI USB3.0 Console, Compatible with Pfsense OPNsense	€384.00

Table 2.1. The first page of search results of "hardware firewall" on amazon.de. Captured at 5 P.M. of December 26th, 2023, Lugano

Statistics of the prices in search results			
Mean	250.74	Median	262.0
Min	55.49	Max	520.89
Standard deviation		109.98	

Table 2.2. Statistics of table 2.1

2.3 Choosing hardware

The historical reason why Raspberry Pi is used is that Pi is the only single board computer (SBC) I have when this firewall started as a hobby project to solve the aggressive hack problem that I was facing. The decision is unchanged after the start of this thesis mainly because of the time costs of trying new SBCs: Since its first release in 2012, Raspberry Pi has upgraded its hardware models and software multiple times, has accumulated a lot of real-world applications, and has maintained an active user community with a large number of experienced users behind it. All of them show that by using Raspberry Pi instead of a cheap new brand, reduced time cost could be expected on fixing the bugs of the SBC itself or waiting for the answers from the community because previous users have asked and answered many questions and have found and fixed many problems. Besides, its large number of users could also help in spreading the firewall project and help in pointing out security flaws of the new upgrades to the computer.

Joy-IT RB-TFT3.5 was the only screen that I had. The dtbo file of the Waveshare 35A screen also works with this hardware. Thus, the cheaper one in the market should be able to replace another. The resolution of 480x320 is enough to print logs. I do not have the material or the budget to try other screens with lower resolutions.

2.4 Deployment and operating system

Hood is designed to work as a USB ethernet device so people do not need to connect many cables. See fig. 2.1.

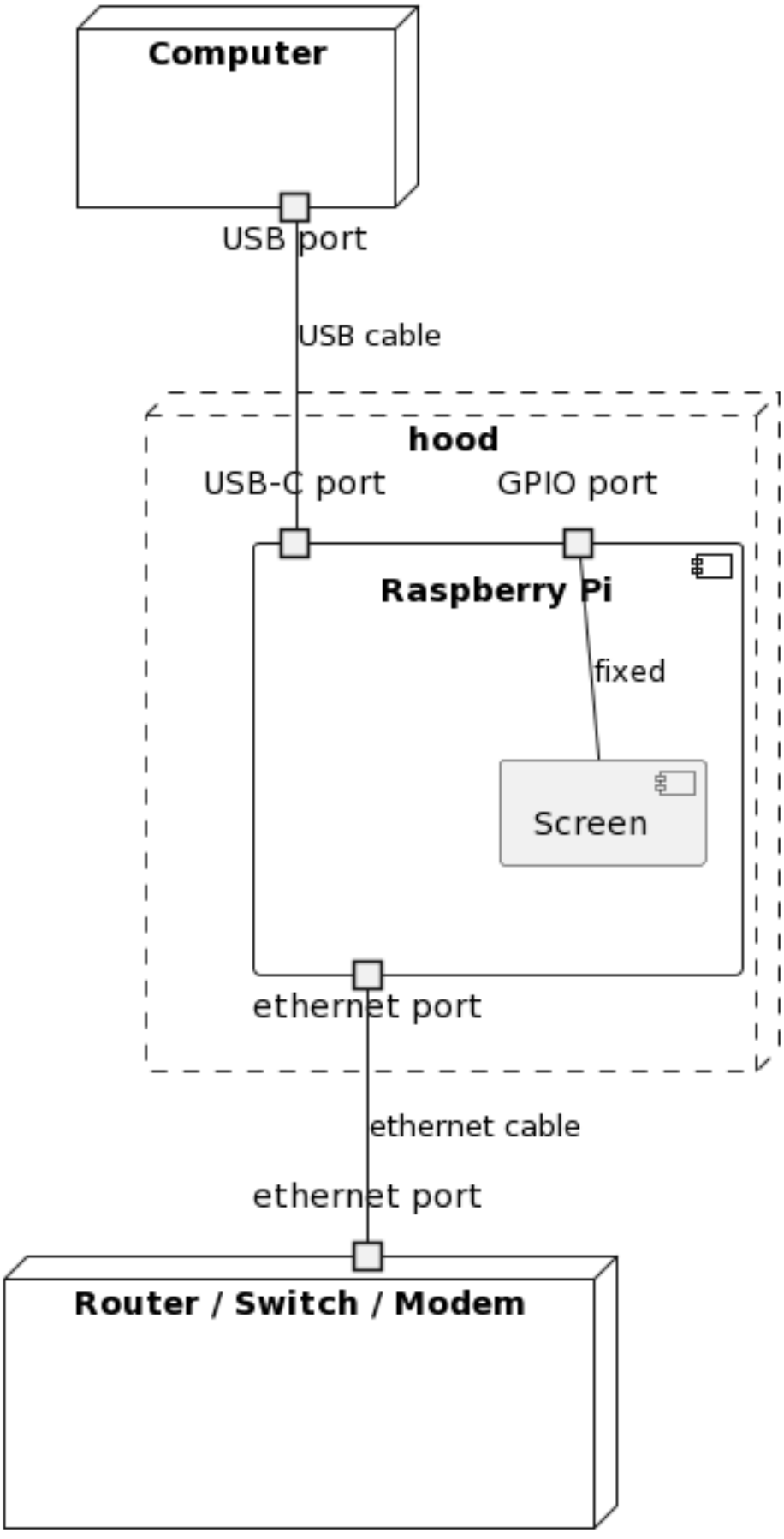

Figure 2.1. Hardware deployment view

To make Raspberry Pi work as a USB device, the support from operating systems (OS) is required. Linux provides USB gadget mode. Using the combination of CDC-ECM and RNDIS mode, most Linux, Windows, BSD, and macOS could be supported. FreeBSD USB stack has a device mode and provides 3 virtual network interface templates but none of them works with Microsoft Windows[Project, 2023]. OpenBSD has renowned security-focused nature but it does not contain a device mode in its USB stack. Thus, to support as many devices as possible, Linux is selected as the base OS of this firewall device.

Chapter 3

Packet filtering and forwarding

3.1 Design

As the general principle, all traffic passed through should be checked against eavesdropping. Plaintext protocols should be channeled through strong encryptions or discarded if unable to ensure their integrity.

A modern browser can satisfy most of the needs to interact with the internet for work. So, allowing HTTP and HTTPS connections could make most of the work done when facing a hostile network environment. Thus, everything unnecessary to get this done is blocked. The firewall rules should be carefully made to filter out as much as possible and as early as possible, because everything on all layers of the Open Systems Interconnection model (OSI model) is potentially vulnerable and the less network traffic being passed to the following components, the less chance a vulnerability can affect the firewall.

Proxies should be used to forward network traffic. There are advantages of using a proxy rather than forwarding packets like a router. First, a proxy can run at the user level while packet forwarding runs at the kernel level. When there is a vulnerability, a userland one naturally be less harmful than a kernel one. Second, proxies are easier and also safer to configure. It's hard to configure operating system firewalls well. When the rules are not strict enough unexpected things may happen. For example, allowing any connection to 127.0.0.1 to succeed without a log, could cause potential problems if a malicious DHCP server assigned the address 127.0.0.1 to a physical network interface. Also, simply filtering packets by port numbers, addresses, and interfaces could not guarantee the packets being forwarded to the remote actually used by the protocols we are expecting. An attacker can simply let the receiving end of a reverse shell listen to an allowed port to bypass this kind of defense. By contrast, a proxy only works for the protocols that it can understand. when you are configuring a DNS proxy, the chance that your misconfiguration makes an attacker able to make HTTP requests over this proxy is very rare. They have to hack the firewall or create tunnels over those protocols to make their reverse shell work, which increases the difficulty of their operations. Third, the packets from the proxy are encoded again by the Linux network stack so the fingerprinting techniques that target TCP and IP properties are useless to the devices under its protection.

3.2 Implementation

Nftables is integrated into modern Linux systems. It can classify and filter network traffic. The firewall uses it to filter network packets. UDP 67 or 68 port and ARP packets are allowed for assigning IP addresses. UDP 53 is allowed from users to the firewall device for DNS requests, TCP 80, and TCP 443 are allowed for HTTP and HTTPS traffic. All other packets are filtered as early as possible.

Different proxies and services are created to check connections, to forward traffic, and to apply encryptions. Details are included in the chapters of the corresponding protocol. See figs. 6.2, 7.1 and 7.2. Several configuration files are created for blocking IP subnets and domain names, they will be processed by proxies and services. See appendices A.3 and A.4.

Chapter 4

Dynamic Host Configuration Protocol (DHCP)

4.1 Introduction

DHCP is widely used to dynamically assign a ip address to a new device connected to a network.

4.1.1 The issue

A client can expose its host name to the network from the 'client identifier' option or 'sname' field in the protocol. When in the same local network, the host name exposed by the DHCP protocol and other protocols can be used to locate the devices of the victim. If the name of a computer is straightforward enough, like "Yongbin's MacBook Pro", the network administrator will be able to know the name of the owner of the computer at first glance. The hardware address exposed to the network can also expose more information than just the address. Hardware address or Media Access Control (MAC) address can be used to reverse lookup the manufacturer and the manufacturer can help identify the owner. For example, the network administrator may look at the MAC address of the device to find out that the manufacturer is company A and if company A only sells its products in Country B and if C is the only person who came from Country B then the chance that the device belongs to C would be high.

4.2 Design

Firewall should randomize both host name and MAC address. The randomized computer name should look normal to prevent being identified easily.

4.3 Implementation

NetworkManager is used to manage physical connections of the firewall. By adding 'ethernet.cloned-mac-address=random' and 'wifi.cloned-mac-address=random' to the 'connection' section of

NetworkManager.conf, the MAC address randomization is done by NetworkManager. The 'ifconfig interface link random' command can also randomize the MAC address of an interface manually.

Host names of the firewall device are randomly generated while starting the system. The relevant code is inside rc.local. The name is crafted to look normal. The pattern of default computer names of Microsoft Windows and common names like iPad are used. Since the implementation of the client of the protocol could also be vulnerable, the combination of AppArmor and dhclient to mitigate this problem.

Dnsmasq is used to assign the IP address of the users of the firewall it also tells the devices behind the firewall to use the firewall as the DNS resolver.

Chapter 5

Domain Name System (DNS)

5.1 Introduction

DNS is a protocol that translates the human-readable domain names to the IP addresses of the remote server. The default configuration of many devices is to use plaintexts to do requests, which could easily be monitored by simply recording the packets and could easily be attacked by methods like DNS spoofing.

5.2 Design

From the network administrator's view, all DNS queries sent out from the firewall should be encrypted. From the user's view, nothing should be specially configured.

5.3 Implementation

5.3.1 Outgoing requests

hood-name-service.py is a name-resolving service created to resolve domain name queries from firewall services. It is a local RPC service that resolves domain name queries via public DNS-over-HTTPS (DoH) services. DoH can mitigate both passive surveillance and DNS spoofing attacks[Hoffman and McManus, 2018]. DoT (DNS-over-TLS) is another DNS encryption standard that can do almost the same as DoH. The reason why use DoH instead of DoT is that DoT uses a unique server port number, 853, that can be easily filtered and identified as an encrypted DNS connection, while DoH, with the help of using the same port number and protocol as HTTPS, makes it much harder to be identified and filtered than DoT.

5.3.2 Inside firewall

A dnsmasq is started locally to resolve everything to localhost to make the firewall proxies system-wide.

5.3.3 Incoming requests

A dnsmasq is started to resolve everything to the firewall device to make the firewall proxies apply to devices behind it.

Chapter 6

Transport Layer Security (TLS)

6.1 Management of certificates

6.1.1 Introduction

All modern major operating systems manage a collection of trusted certificates issued by certificate authorities (CAs) they choose. Those certificates are used to prove the validity of a public key. When making a TLS connection, local applications check the path of the certificate provided by remote with a local trusted collection to ensure secure connections.

The issue

Almost all TLS clients do not alert their users when the remote website presents a different certificate, which gives CAs the power to do man-in-the-middle (MITM) attacks. Even if a CA has no intention to do evil, its private key could still have been stolen. Thus, none of them are strictly trustworthy. However, the whole TLS is based on it, if we trust none of those authorities, there will be almost no website we can use and without TLS, things will only be worse.

Exsting solutions

SSL-pinning can prevent MITM attacks from a trusted CA when the user knows the remote endpoint should use the certificate from another authority. However, configuring SSL-pinning to all the endpoints is hard and time-consuming for average users, and without a basic trusted environment, people are unable to know whether the configuration has been done correctly. Even if the configuration is correct, it is still not strange for an endpoint to switch to another CA. Thus, to a firewall, this technique cannot be used as a general solution to the issue.

Removing suspicious CAs from the trusted list could protect people from MITM attacks initiated by those CAs. However, some systems and devices do provide the function of untrusting a CA. For example, an iPhone needs a computer to enable such configuration and most Android variants do not provide such entry in settings.

6.1.2 Design

The firewall should trust only the CAs that are being widely used on the internet. People should be able to further decide which certificates to trust. One can consider untrust following when making such decisions: the CAs of the place you are staying, the CAs of the place you come from, and their enemies and allies.

6.1.3 Implementation

Since SSL pinning can only be used on a small number of endpoints by an advanced user who can clearly know what is this meant to be, it is not implemented now. Manually management of the certificate can be done by Linux commands. The certificates of all outgoing TLS connection remote points are checked by TLS proxy.

6.2 Mismatch between the changing best current practice and the unchanged implementation

6.2.1 Problem

The mismatch problem has existed everywhere whenever a public protocol standard updated a fix to a problem. There are delays between a deficiency of a protocol being published and the majority of the implementation fixed the problem. A living example is that in the latest version of Python 3.12.1 released on December 2023, as part of its built-in libraries, the implementation of the function create_default_context() from the SSL library is still creating a context that supports TLS 1.0 and TLS 1.1 by default [Foundation, 2023], while they have already been deprecated by the best practice rfc8996 since March 2021 [Moriarty and Farrell, 2021].

6.2.2 Solution

An idea to save the clients that contain severe deficiencies is to attack their deficiencies and then hijack and channel their communication to remote endpoints through the TLS connection with the current best practices applied. Applying this methology to solve the downgrade attack of Secure Socket Layer (SSL) version 2.0 was the initial motivation of the works on the TLS proxy, but due to the drastic changes to the protocol since SSL 2.0, the final implementation is instead to use the proxy to check whether the best practice is applied.

6.3 Proxy

MITM attack

Despite using MITM attack to inspect or protect encrypted connections has been used in firewalls and proxies for many years [FORTINET, 2021], the TLS proxy used in this firewall should not implement the same functionality. The reason is that when the firewall itself is compromised, the ability to decrypt TLS sessions will cause a disaster to the devices behind it: the attacker will be able to easily modify the encrypted data transferred over a MITM proxy while the devices behind the firewall will alert nothing about the modification. However, if the

firewall could not do MITM attacks at all, the compromised firewall is still unable to see and modify encryptions, because the certificate verification of the devices behind this firewall is still working.

Filtering certificates

The key_share extension sometimes makes the certificate invisible to the proxy. Thus, to filter certificates, the proxy handshakes with the remote server with a separate connection to verify the certificate used by the remote host. A cache of trusted endpoints is used to reduce delays caused by certificate check connections.

Checking connections

The proxy should parse handshake packets. It should check algorithms provided by ClientHello and ServerHello, to ensure strong encryption. Strengthen the connection by modifying handshake packets without MITM is not possible because the HMAC of all handshake packets is used for encryption.

Implementation

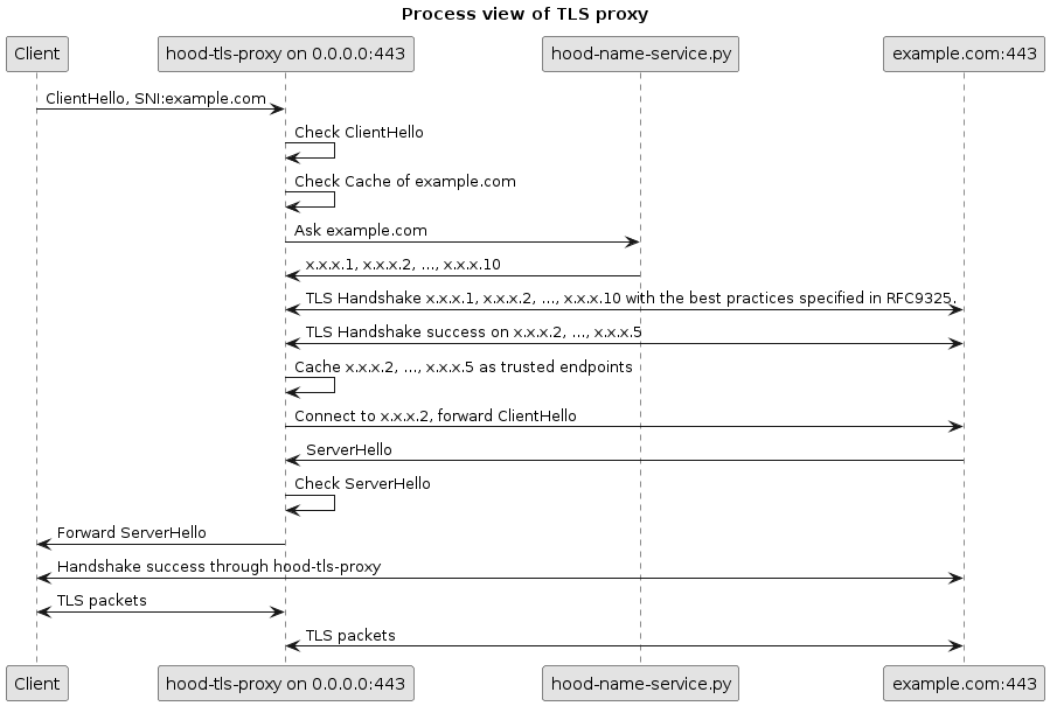

Figure 6.1. Process view of TLS proxy

Programming language Since TLS proxy is in charge of processing and forwarding the majority of the traffic of the firewall, c++ is used to develop the proxy due to its high-performance capability.

ClientHello Upon receiving ClientHello from a client, the proxy first checks the version of the protocol, and the suppor_versions extension to see if the client supports establishing a safe TLS channel, TLS 1.1 and TLS 1.2 are the only allowed versions [Sheffer et al., 2022], if a ClientHello shows that the client supports neither, it will be discarded. After the version check, the proxy checks the ciphersuites field, if it contains none of the values listed in 6.1 [Sheffer et al., 2022], it will be considered unable to be safe and discarded. If the ClientHello passes all of the security checks, the proxy will extract the Server Name Indication (SNI) from ClientHello to determine the actual destination of the request.

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

Table 6.1. Allowed cipher suites values

Name resolving and trusted endpoint The proxy first checks if there are any cached trusted endpoints available, if not it uses a RPC call to hood-name-service.py to resolve the domain name. Then TLS handshakes following the same restrictions to the ClientHello check are made to all of the addresses returned from the name service. The handshake process will not only ensure remote endpoint meets the requirements (versions and ciphersuites) of applying the best practices described in rfc9325, it will also check whether the remote endpoint could provide a valid certificate of the hostname from a trusted CA of the firewall.

ServerHello After forwarding the ClientHello to the server, a ServerHello will be responded and the proxy will check the final result of the handshake applies to the best practice. If a naughty attacker made the negotiation result unsafe, TLS 1.0 or the cipher suite RSA_WITH_NULL_MD5 for example, the connection will be blocked.

6.4 Process view of traffic

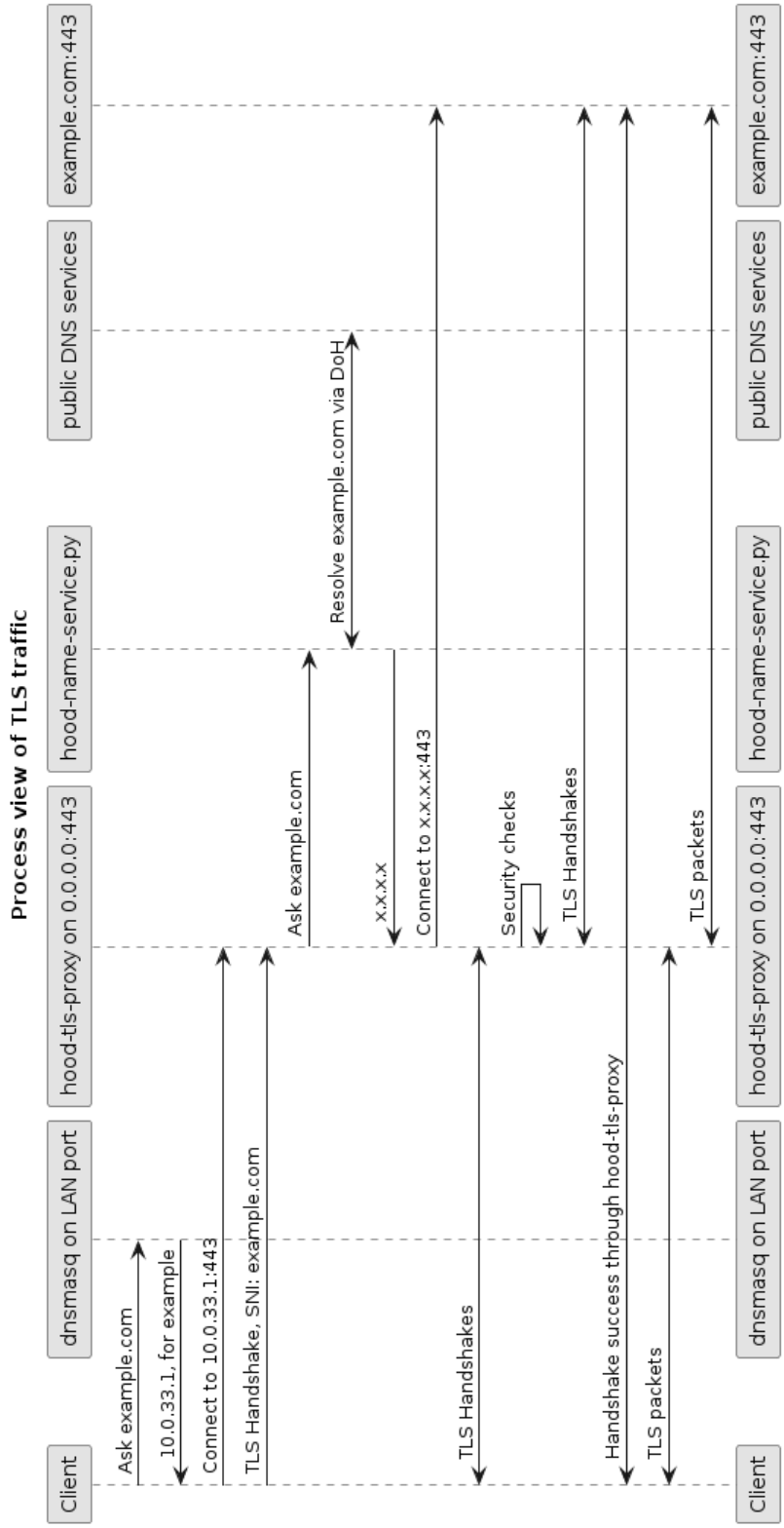

Figure 6.2. Process view of TLS traffic from user

Chapter 7

Plaintext connections

7.1 HTTP

7.1.1 Implementation

hood-http-handler.py is the proxy created to filter and protect HTTP plaintext connections. The proxy checks the host name with a known list of OCSP (Online Certificate Status Protocol) and CRL (Certificate Revocation List) servers. Only the connections to those servers are allowed to be plaintext because those connections usually be OCSP over HTTP connections, which have no reason to be protected. The connections to other servers are tunneled via TLS protocol through the hood-tls-proxy service. See figs. 7.1 and 7.2.

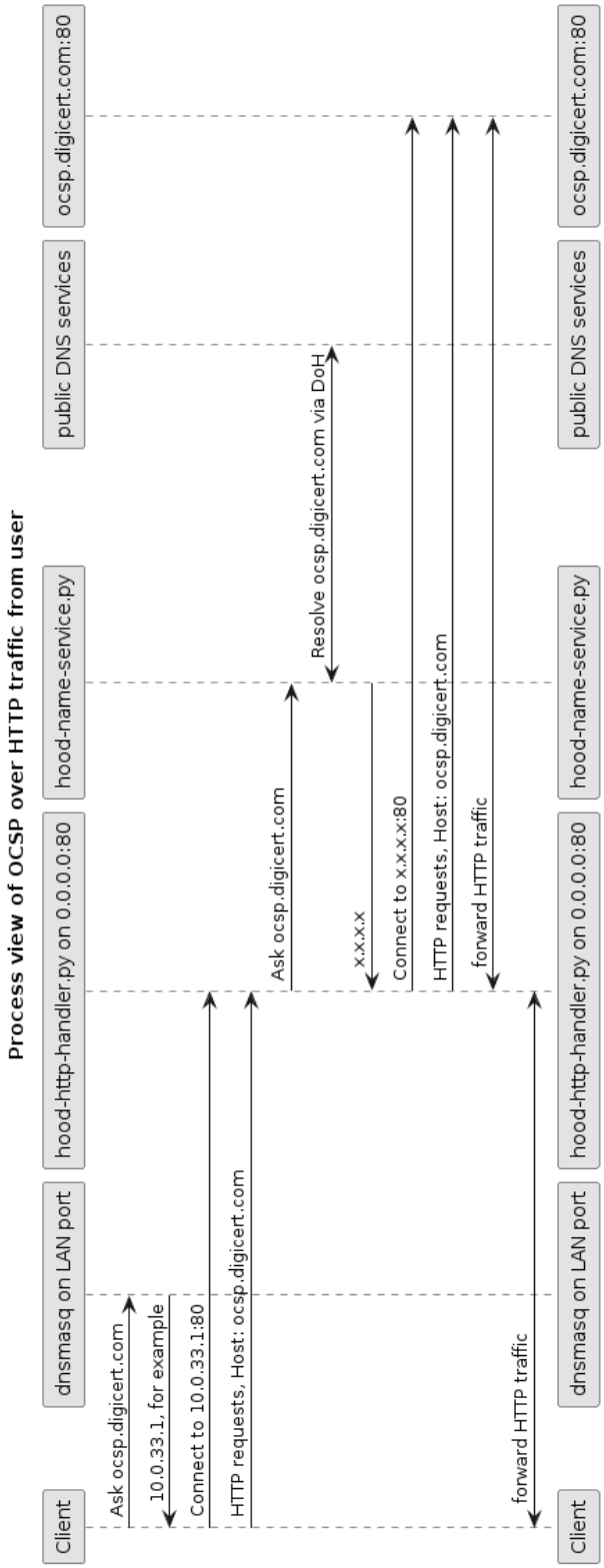

Figure 7.1. Process view of OCSP over HTTP traffic from user

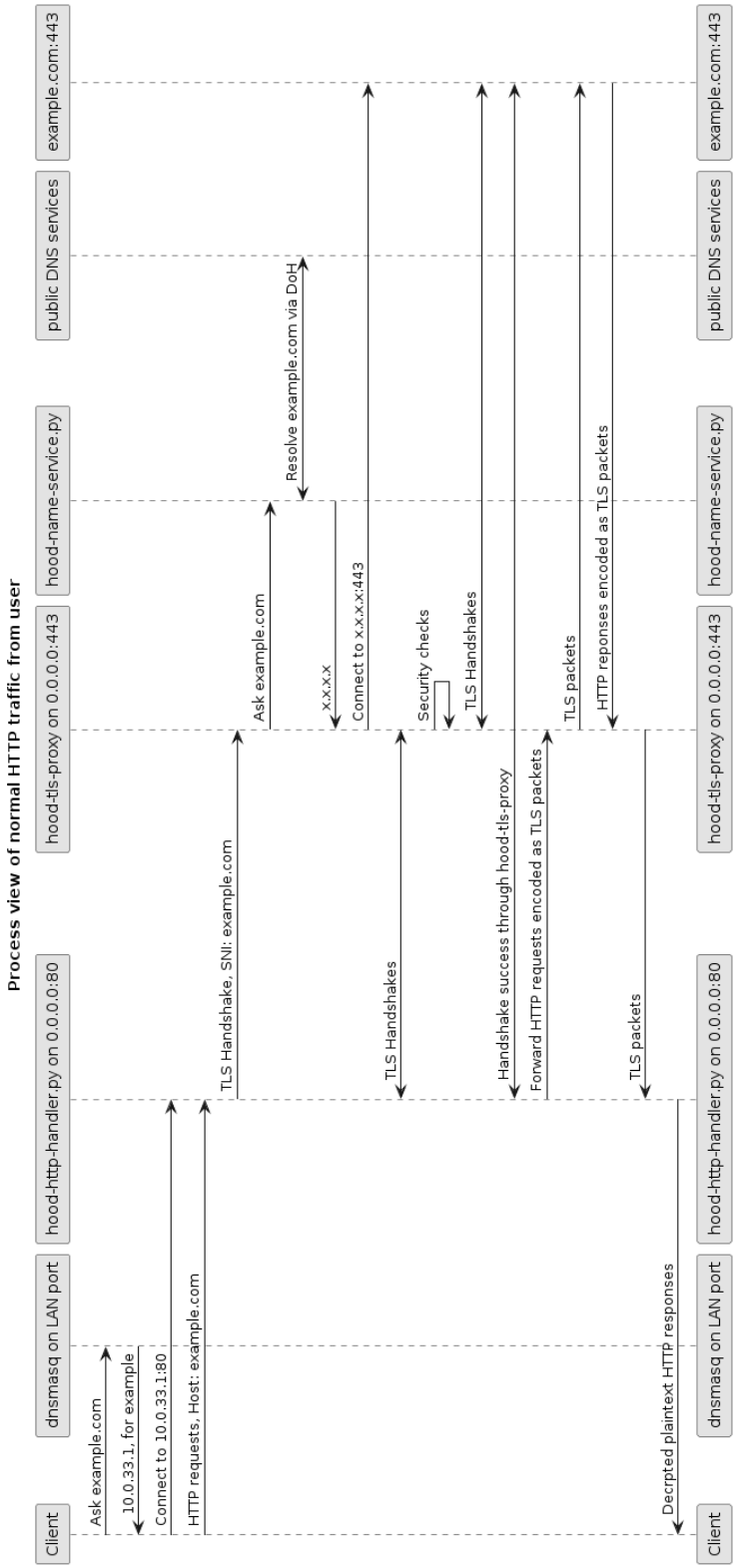

Figure 7.2. Process view of normal HTTP traffic from user

Chapter 8

Date time synchornization

8.1 Introduction

A correct time is required for TLS protocol to check certificates. Network Time Protocol (NTP) is used by most modern systems to synchronize the clock with a remote server.

8.1.1 The issue

A problem with using hardware like a Raspberry Pi is that it does not have a battery to keep the clock ticking after the power source is cut. So a time synchronization mechanism is a must for this firewall.

Many operating systems are pre-configured to use an NTP server which has a unique domain name that can be used by a network administrator to identify the operating systems being used by the client.

Both the unique port number (UDP 123) and the unique domain names (time.windows.com, time.apple.com, and ubuntu.pool.ntp.org) make NTP a protocol that can be easily identified and targeted. Various attacks can be done to the protocol itself[Kwon et al., 2023]. The difference in NTP behavior between different operating systems has been used in OS fingerprinting and tethering detection osandtether. Not to mention that the operating system information being exposed by an NTP domain name can also help an attacker target the NTP client being used by the device.

8.2 Design

Give up on NTP, use HTTP/HTTPS instead. Use the "Date" header from the response of an HTTP server to synchronize date time with acceptable accuracy. To hide from security detections, the attack vectors targeted to HTTP/HTTPS protocol usually do not want to make mistakes in the header. Even if an attacker targeted the method used by the hood firewall, It's hard to say whether this request is made for time synchronization. Thus the date field of an HTTP response is generally trustworthy.

8.3 Implementation

The tool is implemented in a Python script named hood-timesync.py. It accepts two command line arguments as listed in table 8.1. It at first tries to receive an HTTP response through a TLS connection from the domain name specified by the host argument. If it fails, the reason could be the website is down or the system time is too different from the actual time and made a valid certificate unable to pass the validation. Thus, a plaintext connection to the host name provided by the last resort address argument is made as the second attempt. After the second attempt, the tool starts another attempt to receive time from the domain of the host argument via TLS connection, because TLS has less chance to be attacked and because the reason why the first step failed could be the website is temporarily down, the reason could also be the huge time difference made the validation of the certificate failed. In both cases, a HTTPS request to the domain may become possible to succeed after the second attempt. An Epoch time is used as an 'anchor' time. Based on the assumption that time only moves forward, any time before this value will be rejected.

Name	Type	Default	Description
–host	str	www.bing.com	The website to request time from
–last-resort-host	str	1.1.1.1	The website used as the last resort, should be an IP address
–time-anchor	int	1703703361	An Epoch time that being anchored as past. Based on the assumption that time only moves forward, any time before this value will be rejected. Set 0 to disable this test. Example: -time-anchor=$(stat /etc/os-release -c %W)

Table 8.1. Command line arguments of hood-timesync.py

Chapter 9

Behavioral simulation

9.1 Introduction

In previous chapters, the fact that network activities can be used to detect the user has been proved in different protocols. However, the efforts to reduce the leak of information on those protocols are not enough when facing a more sophisticated network analyzation. For example, the connections to Windows update servers and App Store servers can be used to detect Windows and Apple devices citeposandtether, but the firewall has no reason to stop those connections. Even if the firewall blocked all OS-specific connections, the domain leaked from TLS handshakes can also tell the eavesdroppers what websites a person is watching and then may infer what the person is doing. Thus, the leak of the information seems to be inevitable.

9.2 Design

A service to simulate different network activities is designed to solve this issue. The goal is to hide user network activities inside the flood of fake network activities to increase the difficulty of an analyzer to produce useful reports. However, the risk of using this service is to produce a false alarm immediately and make the user blocked for tethering, make the user fired for browsing unrelated websites, etc. To prevent the firewall from being compromised by browser vulnerabilities, browser-based simulation tools, like Puppeteer, should not be used for implementation. It should also make plaintext traffic to attract attention from attackers and eavesdroppers to make them less focused on the actual user traffic.

9.3 Implementation

The service is implemented in a Python script named hood-actor.py. It can parse and execute action files. To simulate the behaviors of a human user or a computer service. All implementations inside the script are based on the base modules of Python, to avoid the version management of third-party libraries being involved in the installation process of the firewall.

9.3.1 Action files

Action files are in JSON format. An action file contains an array of objects. All objects in the array have a "type" field to let the parser know how to process it. Supported object types are descripted in tables 9.1 to 9.3.

Value	Description
properties	Properties of the action
task	Tasks to be done in the action

Table 9.1. "type" field of action file objects

Field	Type	Description
type	string	"properties"
safe_age	integer	Minimum age safe to do this
proper_age	list of two numbers [min, max]	Proper range of age to do this.

Table 9.2. "properties" object of action file

Field	Type	Description
type	string	"task"
name	string	Name of the task, must be unique in current file
before	list of strings	Names of tasks that should run no earlier than this.
after	list of strings	Names of tasks that should run before this.
delay_between_actions	list of two numbers [min, max]	Range of seconds to wait between actions.
actions	list of strings	list hood action script

Table 9.3. "task" object of action file

9.3.2 HoodExecutor

HoodExecutor is a general-purpose multi-thread task executor for Python. It can run and schedule tasks that have dependency relationships and delay requirements in parallel. It has a thread pool of workers to execute tasks and a single thread for scheduling delayed tasks. It is required to simulate the paralleled requests sent out from real-world browsers without a third-party HTTP client based on asyncio. It is also the cornerstone of fulfilling all the execution requirements of the tasks and actions in an action file.

9.3.3 Browser

A fake browser is created to simulate network activities of browsing. It records the cookies specified in the HTTP response headers and sends them out like a real browser. It parses the HTML content of the webpage to extract the resources to be loaded and load them in parallel

just like a real browser. It uses headers that are used by real browsers and uses different headers on different resources just like a real browser. Cookies are stored in memory and used in the running session. However, due to security concerns, and also to reduce the performance impacts caused by the actor, the fake browser will neither render the webpage [CVE-2023-6707, CVE-2023-6351, CVE-2023-6869, CVE-2023-5170] nor evaluate scripts [CVE-2023-6702, CVE-2023-5997, CVE-2023-5728, CVE-2023-5723] used by the webpage.

9.3.4 Hood action script

A hood action script is a script to describes actions to be done to the actor. Each line of the script can be seen as three parts: namespaces, commands, and arguments. For example, "hood:browser:goto:http://www.example.com". "hood:browser:" is to refer browser namespace from the hood namespace, "goto" is a command inside the browser namespace to tell the interpreter to use the browser to load a webpage. "http://www.example.com" is the parameter of the "goto" command. In this example, it represents the URL of the webpage to be loaded.

hood:browser:goto command

Its usage is "hood:browser:goto URL". Its effect is to use a fake browser (See section 9.3.3) to load a URL. When the special value "random_link" is used as the argument, the actor randomly picks a link from current webpage of the browser to go to, but nothing will be done if the current webpage contains no links.

hood:loop command

Its usage is "hood:loop:CONDITION:COMMAND". It is used to declare a loop. Multiple conditions can be declared in the CONDITION part by syntax "key=value[,key=value]". Currently, only two types of conditions are implemented: "exit_chance" and "delay". "exit_chance" describes the chance of the loop to be stopped. "exit_chance=0.33" means the loop has a 33% chance of termination at the beginning of each loop, "0" means it will never have an end. "1" means the loop will never be executed. "delay" describes the range of delay between each loop. "delay=0-2.5" means the delay between each loop is from 0 seconds to 2.5 seconds. A Just-In-Time compiler is implemented to convert the CONDITION field into Python code. The COMMAND part is the hood action script to be executed in the loop.

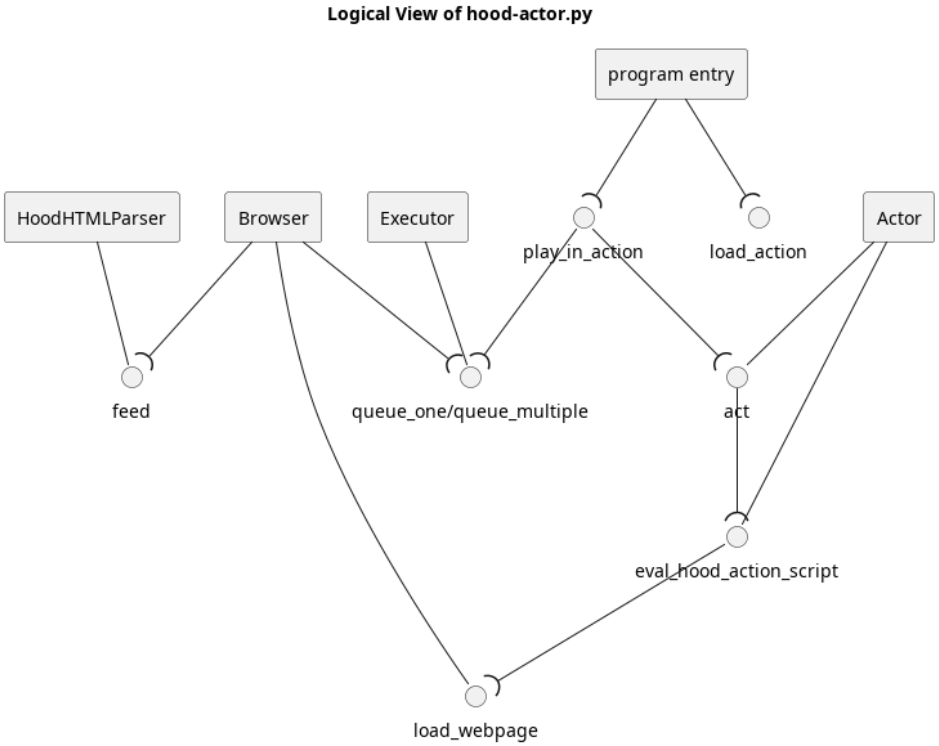

Figure 9.1. Logic view of hood-actor.py

Chapter 10

Dispatcher and firewall rules

Dispatcher is a shell script named 02-hood-dispatcher. It interacts with the changes in the network status or the system status to dynamically configure firewall rules and to start or stop network services. Its callers are three systemctl services: before-network.service (See appendix A.1), udev.service, and NetworkManager-dispatcher.service.

10.1 Initial state of firewall rules

The initial state of firewall rules (See appendix A.5) is basic because the design of nftables made the rules with interface name involved requires to be done dynamically. It only allows IPv4 and ARP protocol at layer 3. It allows anything to lo interface but rejects anything else to communicate from 127.0.0.1 or to 127.0.0.1. It allows UDP/TCP packets in connection track established and related state. It allows the device to send or receive ICMP fragmentation-needed packets. Anything else is ignored.

10.2 On udev add event

netdev filter

A default netdev filter will be added to both LAN and WAN interfaces. It has to be added dynamically because both its ingress and egress filters require the name of the device. The filter accepts only ARP and IP packets to pass through. The firewall rules from other places already achieved the same thing but it is added because it is the earliest place to filter out a packet and the earlier a packet is filtered the less chance the packet to cause a problem.

10.3 On system startup

Upon system startup, the dispatcher will be called by before-network.service. It will start two dmesg instances one is for redirecting firewall-related logs to a log file another is for formatting the log to a more human-readable form. In the iptables era, it was used for initializing iptables rules.

Since the udev event could be triggered at a very early stage of the startup, so early that the nftables service may not be available, and adding rules may fail. The dispatcher tries to add a netdev filter on all available network instances again because the timing before-network.service is sufficient to ensure the availability of nftables.service.

10.4 On NetworkManager-dispatcher pre-up event

This event means that a network interface is connected to the network but is not yet fully activated. The dispatcher checks the device path with the device specified during the installation process to know whether it is a WAN port. The device path is the real path of the network device. For example, on Linux, the device path of eth0 is the result of "realpath /sys/class/net/eth0".

WAN port

For the WAN port network interface, the dispatcher allows UDP packets to be sent out from the local 68 port to the remote 67 port and to be received from the remote 67 port to the local 68 port for DHCP protocol.

LAN port

For LAN port network interfaces, the dispatcher at first picks a randomized start number $\in [2, 255]$ as the third byte of the IPv4 address, and all following LAN ports use the result of

$$2 + (LastUsedNumber - 1) \mod 254$$

as the third byte. The randomization of the start number is made after each system starts. LAN subnet randomization could help increase the difficulty of malicious code in detecting the firewall from the LAN port.

After randomization, the following firewall rules are added:

- Accept the incoming TCP connections through the LAN interface if the destination IP address is the LAN IP address and the destination port is 80 or 443. For hood-http-handler.py and hood-tls-proxy
- Accept the incoming UDP packets through the LAN interface if the destination IP address is the LAN IP address and the destination port is 53. For the dnsmasq instance.
- Allow UDP packets to be sent out from the local 67 port to the remote 68 port and to be received from the remote 68 port to the local 67 port. For the dnsmasq instance.

Table 10.1. Firewall rules to be added for LAN ports

A dnsmasq instance is started for this network interface, to assign IP addresses to connected devices, to let connected devices use the IP address of the LAN port of the firewall as the DNS server, and to respond to the IP address of the firewall LAN port for DNS requests.

10.5 On NetworkManager-dispatcher up event

The dispatcher only reacts to this event for the WAN port. It at first waits for DHCP to finish. After DHCP is done, the dispatcher removes DHCP related accept rules from nftables and adds a rule to accept outbound TCP connections from the WAN instance with the source address as the DHCP result to remote 80 and 443 ports. This rule is for HTTP and HTTPS protocols. After that, the dispatcher restarts the hood-network-services.service (appendix A.2). It will also check the existence of /do_upgrade or /run_once files to do a system upgrade or to run the script.

10.6 On NetworkManager-dispatcher down event

The dispatcher removes the firewall rules dynamically added for this instance and kills the dnsmasq instances for this interface.

Chapter 11

Installation

Installation now can only be done from Linux. install.sh is the shell script that is in charge of the installation process. It copies and applies the scripts, executables, and configuration files to the target Raspberry Pi OS filesystem. It accepts the command line arguments listed in table 11.1.

Name	Default	Description
usb_tether=	1	Share network to computer via USB cable
harden_only=	0	Only apply hardening parts. Let the target SBC can still used as a computer.
disable_wireless=	1	Disable WiFi and Bluetooth.
disable_gpu=	1	Disable GPU.
target=	/	The target root/device to install firewall.
wan_port_ device_path=	/sys/devices/ platform/scb/ fd580000.ethernet/ net/eth0	The path of the device to be used as WAN port.

Table 11.1. Command line arguments of install.sh

11.1 Checking target system

Multiple checks to the target path are done before the beginning of the installation process to avoid potential harm to the user's computer when they specify a wrong target path. It first checks the type of the target, if the target path is not a directory target will be treated as a device or file and will be mounted as the pattern of a live system of Raspberry Pi OS. If the target is a file, then the target will be mounted as a disk image. If the target does not exist, "/dev/" prefix will be added and will be treated as a name of a device. Then if the content of /boot/firmware/config.txt of the target does not contain the string "dtparam", the target will not be treated as a Raspberry Pi OS filesystem and the installation process will be aborted.

11.2 Disable wireless

Multiple methods are used to ensure the complete disablement of wireless devices. See tables 11.2 and 11.3.

Configuration file	Contents added
boot/firmware/config.txt	dtoverlay=disable-bt dtoverlay=disable-wifi
/etc/modprobe.d/bin-y-disable-wireless-blacklist.conf	blacklist bluetooth blacklist btbcm blacklist hci_uart blacklist i2c_brcmstb blacklist i2c_dev blacklist brcmfmac blacklist brcmutil blacklist cfg80211

Table 11.2. Configurations added to target to disable wireless

- /lib/firmware/brcm/*
- find /lib/linux-image*/broadcom -type f
- find /usr/lib/modules/ -name bluetooth

Table 11.3. Files deleted from target to disable wireless

11.3 Disable GPU

Multiple methods are used to ensure the complete disablement of GPU. See tables 11.4 and 11.5.

Configuration file	Contents modified
boot/firmware/config.txt	Removed: dtoverlay=vc4-kms-v3d dtoverlay=vc4-fkms-v3d
/etc/modprobe.d/bin-y-disable-gpu-blacklist.conf	Added: blacklist v3d blacklist drm blacklist drm_panel_orientation_quirks

Table 11.4. Configurations modified to target to disable GPU

- Any directory under /usr/lib/modules/ with the name "gpu"

Table 11.5. Files deleted from target to disable GPU

11.4 Other configurations modified

Configuration file	Contents modified	Purpose
/boot/firmware/cmdline.txt	Added: ipv6.disable=1 apparmor=1 security=apparmor	disable IPv6 enable AppArmor
/etc/modprobe.d/bin-y-blacklist.conf	Added: blacklist ipv6 blacklist hci_uart blacklist i2c_brcmstb blacklist i2c_dev	disable IPv6, i2c, and UART
/boot/firmware/config.txt	Added: enable_uart=0	disable UART
/etc/nftables.conf	See appendix A.5	nftables rules
/etc/hosts	OMITTED	provide address of some DNS services
/etc/sysctl.conf	OMITTED	sysctl related hardening
/etc/rc.local	OMITTED	host name randomization
/etc/pki/nssdb/cert9.db /etc/pki/nssdb/key4.db /etc/ca-certificates.conf	OMITTED	manage trusted certificates
/etc/NetworkManager/NetworkManager.conf	OMITTED	MAC address randomization
/etc/apt/sources.list /etc/apt/sources.list.d/raspi.list	OMITTED	use HTTPS mirrors for apt
files related to systemd services	OMITTED	enable hood services, enable required system services, and disable unsafe services

Table 11.6. Configurations modified to target to disable GPU

Chapter 12

Showing network activities

Network activities of the firewall system are logged to multiple places by multiple services (see table 12.1). All logs are simplified to human-readable forms and aggregated to a tty device to let users access them easily. If the firewall was installed with the parameter harden_only=1, aggregated logs will be sent to tty8, and if the installation parameter harden_only was 0, aggregated logs will be sent to tty1.

Service	Log location
hood-name-service.py	/var/log/hood-name-service.log
hood-http-handler.py	/var/log/hood-http-handler.log
hood-tls-proxy	/var/log/hood-tls-proxy.log
hood-dispatcher	/var/log/hood-dispatcher.log
nftables	dmesg

Table 12.1. Log locations of network services.

Chapter 13

Reasons of not to use

In most cases, the reason to not use something is to reduce the attack surface.

13.1 IPv6

13.2 WiFi

13.3 Bluetooth

13.4 GPU

13.5 Honeypot

Chapter 14

Conclusions

14.1 Possible improvements for future

14.1.1 LSM and seccomp

Use LSMs, like AppArmor, and seccomp to protect all as many processes as possible. The programs that do not support seccomp can be changed by LD_LOAD_LIBRARY.

14.1.2 Compile time hardening

Use strict compiler options to harden everything, including kernel, like what Gentoo Linux is doing now, to try to mitigate some unpublished vulnerabilities, and also to increase the difficulty of attacking the firewall itself.

14.1.3 Network stack fingerprinting

Spoof network stacks, like TCP stacks and TLS stacks, make the firewall and the devices behind it to be less detectable.

14.1.4 Further use to libcomposite

Libcomposite can make the device show multiple roles to a host (a computer), which means the device can at the same time work as a USB mass storage device or as a USB CDROM, both of which can be used as a media of a Live DVD. Then the computer will be able to boot a live system from it. For a computer that is password protected to boot from USB, we can also use this small device as a PXE server to make that computer boot from the network of the USB device.

Chapter 15

The attacks encountered during the time I was working on this thesis

15.1 Malicious hardwares

Normal attackers will use the network, Bluetooth, or WiFi for communications between malware and the host. For example, casting the screen to another device via WiFi or sending keyboard inputs via Bluetooth. Those kinds of attacks can be detected by an RF detector. RF shielding fabric, RF detectors, and signal jammers could be used to fight against such kinds of attacks.

However, there are still other methods. Some even without wireless communications. Despite I had already bought an RF detector to alert me of wireless communications between unknown hardware, unknown attackers still managed to monitor my progress with some modified hardware. Take the devices that I am using to develop this project for example. Raspberry 4B has a USB chip that can communicate with the power source. If the charger is specially crafted with the ability to forward the USB connection to the remote endpoint via power lines, then, with the help of the malware installed on the Pi itself, data can be leaked silently via power lines even without any network connections to the computer. The same story can also happen to the portable screen that I am using now. I have found two counter measurements to this kind of attack: One is to use a USB-C to DC adapter when connecting a USB-C charger to the laptop. Another is to tape the two pins in the middle of the male USB-A port to prevent data communications.

15.2 Sounds

Another attack that can bypass a computer without a wireless device, is to use AI / ML to identify the sounds of keyboard hits. The detected types can be sent to remote via power lines or mobile phones. The sound could even be recorded from the room of a neighbor which makes this attack more stealthy than other methods. I use cardboards to extend the pillar of the key cap to shorten the key travel to lower the volume of the sound of the key hit to counter this attack but I am uncertain about the effect because I have no attack tools to test this. AliPay

also used to use sound waves to transmit data between phones and vending machines. My Raspberry Pi recently started to emit strange noises from the speakers on the screen, which could be because of the same kind of technology being used by hackers.

Appendix A

Simple scripts and services created by firewall

A.1 before-network.service

before-network.service is a systemd service being enabled during installation process and run before network is available to system. It is created to trigger dispatcher (See chapter 10) to initialize firewall rules and network configurations.

A.2 hood-network-services.service

It is a service for running hood network services. It runs hood-network-services-runner.sh which starts hood-http-handler.py, hood-name-service.py, hood-tls-proxy, and a dnsmasq client for DNS queries made from the firewall system.

A.3 domain_blacklist.txt

It is a text file with lines of regular expressions. Lines starting with # are treated as comments and are ignored. Domains matching any one of the regular expressions in the file will be blocked.

A.4 ip_subnet_blacklist.txt

It is a text file with lines of IP subnets. Lines starting with # are treated as comments and are ignored. IP addresses matching any one of the subnets in the file will be blocked.

A.5 nftables.conf

```
#!/usr/sbin/nft -f
```

```
flush ruleset

table ip filter {
  chain input {
    type filter hook input priority filter; policy drop;
    iif lo accept
    ip daddr 127.0.0.1/8 log prefix "[HOOD D]" flags all drop
    ip saddr 127.0.0.1/8 log prefix "[HOOD D]" flags all drop
    meta l4proto udp ct state {established, related} log prefix "[HOOD A]" flags all a
    ct state {established, related} accept
    icmp type {destination-unreachable} icmp code {frag-needed} accept
    log prefix "[HOOD D]" flags all drop
  }
  chain forward {
    type filter hook forward priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain output {
    type filter hook output priority filter; policy drop;
    oif lo accept
    ip daddr 127.0.0.1/8 log prefix "[HOOD D]" flags all drop
    ip saddr 127.0.0.1/8 log prefix "[HOOD D]" flags all drop
    meta l4proto udp ct state {established, related} log prefix "[HOOD A]" flags all a
    ct state {established, related} accept
    icmp type destination-unreachable icmp code {frag-needed} accept
    log prefix "[HOOD D]" flags all drop
  }
}

table ip6 filter {
  chain ingress {
    type filter hook input priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain prerouting {
    type filter hook input priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain input {
    type filter hook output priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain forward {
    type filter hook forward priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain output {
```

```
    type filter hook output priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain postrouting {
    type filter hook forward priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
}

table bridge filter {
  chain ingress {
    type filter hook input priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain prerouting {
    type filter hook prerouting priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain input {
    type filter hook output priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain forward {
    type filter hook forward priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain output {
    type filter hook output priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
  chain postrouting {
    type filter hook postrouting priority filter; policy drop;
    log prefix "[HOOD D]" flags all drop
  }
}

table netdev filter {
}


table arp filter {
  chain input {
    type filter hook input priority filter; policy drop;
  }
  chain output {
    type filter hook output priority filter; policy drop;
  }
```

```
}
```

Glossary

Bibliography

FORTINET. Security profiles. In *FortiOS - Administration Guide*, chapter 9, pages 1026–1029. June 2021. URL http://docs.freebsd.org/en/books/handbook/usb-device-mode/.

Python Software Foundation. ssl - tls/ssl wrapper for socket objects. In *Python 3.12.1 documentation*. December 2023. URL http://docs.freebsd.org/en/books/handbook/usb-device-mode/.

Paul Hoffman and Patrick McManus. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018. URL https://www.rfc-editor.org/rfc/rfc8484.

Jonghoon Kwon, Jeonggyu Song, Junbeom Hur, and Adrian Perrig. Did the shark eat the watchdog in the ntp pool? deceiving the ntp pool's monitoring system. In *USENIX Security'23*, USENIX Security Symposium. USENIX, August 2023. URL https://www.usenix.org/system/files/usenixsecurity23-kwon.pdf.

Kathleen Moriarty and Stephen Farrell. Deprecating tls 1.0 and tls 1.1. RFC 8996, RFC Editor, March 2021. URL https://www.rfc-editor.org/rfc/rfc8996.

The FreeBSD Documentation Project. Chapter 28.usb device mode/usb otg. In *FreeBSD Handbook*, chapter 28. July 2023. URL http://docs.freebsd.org/en/books/handbook/usb-device-mode/.

Yaron Sheffer, Peter Saint-Andre, and Thomas Fossati. Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls). RFC 9325, RFC Editor, November 2022. URL https://www.rfc-editor.org/rfc/rfc6176.
1