# ETH Zürich
# Network Security
# Exercise sheet four: based on lecture 6.

Name:Luca Di Bartolomeo

Handing in this exercise sheet is optional. You can hand in by sending the filled out exercise sheet to networksecurity-handin@lists.inf.ethz.ch.

If you want individual feedback for your solutions, you have to hand in your solution by

**Monday, 29.10.18 10:00**.

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, attach more sheets to the back and number your answers.

| Question: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
|---|---|---|---|---|---|---|---|---|
| Points: | 6 | 3 | 2 | 2 | 2 | 8 | 5 | 28 |
| Score: | | | | | | | | |

1. **Malware obfuscation:** You bought a malware from your local hacker. Of course, you were exited and published it. However, you are somehow disappointed that your malware was detected quite easily. For the next run you want to do better. That is, you want to harden your malware against anti-malware software.
   On a high level, the code you received was the following:

```
void exploit(){
        int a = load_part_a();
        int b = load_part_b();
        if(a==42){
                run_exploit_internal(a,b);
        }
        return;
}
```

Figure 1: Sample exploit code

   (a) (3 points) Create 3 permutations for the code above to obfuscate it from a static binary analysis.

**Solution:** First permutation:

```
void exploit(){
    int a = load_part_a();
    if(a==42){
        int b = load_part_b();
        run_exploit_internal(a,b);
    }
    return;
}
```

Second permutation:

```
void exploit(){
    int b = load_part_b();
    int a = load_part_a();
    if(a==42){
        run_exploit_internal(a,b);
    }
    return;
}
```

Third permutation:

```
void exploit(){
    int a = load_part_a();
    if(a==42){
        run_exploit_internal(a,load_part_b());
    }
    return;
}
```

(b) (2 points) List at least 2 techniques other than modulation of the malware code that can help circumvent the detection of your malware.

**Solution:**

1. You can use a packer; that is compress the malware and put it inside a binary that extracts it and executes it.
2. You can use a crypter; encrypt the malware and decrypt it just before executing it; even a simple XOR encryption will help hiding the malware.
3. You can implement anti-debugging or anti-virtualization features inside the malware; that is trying to understand if the malware is being debugged or is being run in a sandboxed environment and in that case abort execution.

(c) (1 point) After you performed every detection counter measure you could think of, you want to test the detection evasion performance of your malware. How can you do that?

**Solution:** You can download many antivirus tools and test if they are able to flag your code as malicious. There are some tools that can automate this process.
You shouldn't do things such as uploading it on VirusTotal.com as your sample will be shared with many antivirus vendors.

2. (3 points) From the point of view of a firewall, what is the difference between the following two scenarios:

- A server runs a service which has a known vulnerability.
- A user accesses a website and downloads malicious content from that server.

In general, which ones of these scenarios would you think is harder to protect against and why?

**Solution:** The **second** scenario is much harder to protect against.
The reason is that in the first case, the objective of the firewall is clear and well-defined; it should capture and drop packets that are trying to abuse the known vulnerability.
In the second case, however, it's impossible to set up a firewall that can protect against all vulnerabilities of the client's system (especially unknown ones).

3. (2 points) You run a large IT company where developers can take their laptops home to work remotely for their convenience. Despite such a benefit, the developers are grumpy, and this time they are complaining about the strict firewall rules that you impose on egress traffic (i.e., outgoing traffic from the company to the Internet). As you have confidence in your firewall that it can perfectly detect all threats in ingress traffic (i.e., incoming traffic from the Internet to the company), you decide to disable egress filtering and analysis to make your developers happy. Obviously, this isn't a good decision, but is surprisingly frequent. How does egress filtering help protect your network, and how can the adversary attack it now (when egress filtering is disabled)?

**Solution:** Since the developers are bringing their laptop home, they open up to a new set of attacks that weren't possible when they were only inside the company network.
Firstly, they could be able to disable the firewall and surf the internet without it, infecting their computers and bringing infected computers inside the company network. But even if they don't go on the internet, they could still be infected by a malicious usb stick or hardware device that they attach to their laptops.
Those kinds of attacks will never be detected, because outgoing traffic generated by malware will not be monitored anymore). Outgoing traffic by the malware can happen either because of requests to the command and control server, or by the malware sending confidential information to an external agent.

4. The access point router that your ISP gave you functions also as a NAT (Network address translation). That is, your internal IP address gets translated to the router's external one when you communicate with someone outside of your homw network. In addition, the router changes (or translates) the port information in the packets. The router dynamically determines how to translate your port information when you initiate communication (i.e., send the first packet) to the outside.

   (a) (1 point) You want to host a Counter-Strike server, which listens on TCP/UDP port 27015, on a machine that is behind a NAT. How would you enable players to connect to your server? How many different Counter-Strike servers can you host behind the same router?

   **Solution:** You have to setup *port-forwarding* on your router. Port forwarding tells the router that packets received on a specific port should be forwarded to a precise IP inside the NAT network. This enables packets to be received on the correct port by the server. Behind a NAT there could be as many servers as free ports there are (more or less 65000)

   (b) (1 point) You want to connect to a computer which is also behind NAT, without using the technique from the previous step. Both of you can also talk to the common rendezvous server. How can you establish a connection?

   **Solution:** There is another technique called NAT traversal (hole punching). Suppose we have two clients, `C1` and `C2`, that want to establish a connection with the help of a server `S`. Let's divide the process in steps:

   1. `C1` contacts `S`, and `S` records `C1` (external) address and port.

2. `C2` contacts `S`, and `S` records `C2` external address and port. S responds with `C1` address.
3. `C2` tries to contact `C1`; the connection will fail because even if the address is correct, the NAT will find out that the incoming address is different from the original one (which was the address of `S`)
4. `C1` will contact again `S`, and `S` will respond with the address of `C2`.
5. `C1` will contact `C2`, and the connection will successfully establish, because the NAT will see it as an answer for the request sent in step 3.

This is what bittorrent does in the background to establish peer to peer connections.

5. (2 points) What is the response when you send a SYN packet to:
   - open TCP port
   - closed TCP port
   - TCP port filtered by the firewall

**Solution:**

- open TCP port: The server will answer with a SYN/ACK packet.
- closed TCP port: The server will answer with a RST (reset) packet.
- TCP port filtered by the firewall: The server will either not answer, or answer with an ICMP "unreachable" packet.

6. On systems running a Linux kernel, `iptables` is a common tool used to configure firewalling rules. In this question we will look at some such rules and try to examine what they are doing.

   (a) (3 points) What does the following rule literally do? (i.e. explain what it does to the packets).

```
iptables -A FORWARD -o vpn0 -p tcp -m tcp \
        --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 1360
```

**Solution:** That command appends a new rule to the "FORWARD" chain in the standard "Filter" table of iptables.
In particular, the rules sets the maximum segment size to 1360 of certain TCP packets (both SYN and RST packets with the SYN flag set) and forwards those packet via the "vpn0" interface.

   (b) (1 point) What could be a reason to use this rule?

**Solution:** This rule could be useful on servers which block "ICMP Fragmentation Needed" or "ICMPv6 Packet Too Big" packets.
By setting this rule, the clients connecting to the server will set up a TCP connection in which the largst segment will be of the maximum allowed packet size on the server, and will avoid sending packets too big.
Normally, this rule is not needed, as if a packet is too big the client will receive a fragmentation needed message and will proceed in sending the fragmented parts of that packet.

(c) (2 points) Port knocking is a technique through which access to services can be limited to people who first attempt to make a connection to a specific set of ports. What are the security guarantees which such a scheme provides?

**Solution:** This technique does not provide any security guarantees, as its a form of authentication no better than a password sent in clear text over HTTP.

An attacker sniffing the network could easily reproduce the pattern of connections that a client does and successfully use the hidden services of the server.

(d) (2 points) Consider a port knocking scheme where the sequence of knocks is determined by a pre-shared secret between a client and a server. For example, if Bob wants to connect to a service run by Alice, Bob performs a sequence of port knocks derived from the pre-shared secret (e.g., if the secret is 1234, then Bob must knock on ports 1233, 1235, 1234 in this order) before being allowed to connect. In addition, pre-shared secrets are only for one-time uses; that is, different secrets are used for different connections. With the above port knocking sheme, what security guarantees can you provide against passive attackers? What if the attackers are active (i.e., read, create, modify, drop packets)?

**Solution:**

- Passive attacker: the only security guarantees are authentication and integrity, as long as there are enough pre-shared secrets to do every connection. The attacker has no way of guessing which will be the sequence of ports to authenticate to the server. Confidentiality depends on whether the connection after the port-knocking is done over an encrypted channel like TLS or not.
- Active attacker: in this case, the connection is not secure. The attacker could drop all the packets sent by the clients, and record the destination ports of the packets. Later, he can attempt connections to the server to the same ports and in the same order as the client did, and successfully authenticate to the server and use its services. For confidentiality and integrity of the packets, it depends on the encryption of the connection that is used after the port-knocking.

7. The Berkeley Packet Filter provides a mechanism for userspace applications to filter packets in the kernel without needing to copy each packet recieved on the network interface to a userspace application in order to make a decision about whether that packet is of interest to that specific application.

This works by providing a small program in BPF bytecode to the kernel, which then runs this program in the datapath when processing packets to determine if the application wants to receive this packet. The program returns non-zero if the packet is of interest to the application, otherwise returns zero.

For further information about BPF, you might want to consult some documentation such as but not limited to:

- Linux kernel documentation on BPF [1]
- BPF paper, USENIX 1993 [2]

(a) (1 point) What would be an issue for a system that copies every received packet to the userspace application?

[1] https://www.kernel.org/doc/Documentation/networking/filter.txt

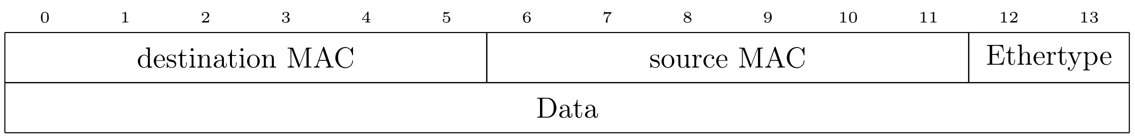
[2] http://www.tcpdump.org/papers/bpf-usenix93.pdf

**Solution:** The system will suffer from performance issues. BPF enables the userland applications to specify in advance in which packets they are interested, thus avoiding the copying of unnecessary data.

(b) (2 points) The `tcpdump` program is a widely used Linux utility to capture packets for administrative or debugging purposes. Explain how `tcpdump` would use BPF to capture TCP packets with the destination port 22 (e.g., when a user issues the command `tcpdump -i eth0 tcp dst port 22`).

**Solution:** `tcpdump` can use BPF to capture only packets corresponding to the filter provided by the user (in this case, TCP packets with destination port 22), avoiding to have to manually check for each packet from interface `eth0`, and substantially increasing performance.

(c) (2 points) Recall that an Ethernet header looks as follows:



It consists of 6 octets each for the source and destination MAC addresses, 2 octets for the Ethertype, a variable-length data, and a CRC. An octet is a unit of 8 bits. You can find the different Ethertypes in [3].

Further recall that the IPv4 header contains the IP protocol type field at the 9th octet (counting from beginning of the IP header). You can find the values for the IP protocol type in [4].

The following BPF program is executed on the received Ethernet frames:

```
        ldh [12]
        jne #0x800, drop
        ldb [23]
        jneq #6, drop
        ret #-1
        drop: ret #0
```

What kind of packets does the above BPF program accept?

**Solution:** The program looks at bytes 12 and 13 of the packet in the first instructions, and checks if they are equal to 0x800 (That corresponds to IPv4 Ethertype). Then loads the 23rd byte of the packets (which corresponds to the 9th byte of the payload, since the Ethernet header is 14 bytes long) and checks if it is equal to 6 (TCP protocol).

Summing it up, this BPF filter captures IPv4 packets that are part of the TCP protocol.

[3] `https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml`
[4] `https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`