



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Emanuele Iannone  
Ph.D. Student  
Software Engineering (SeSa) Lab  
University of Salerno

[eiannone@unisa.it](mailto:eiannone@unisa.it)  
<https://EmanueleIannone3>  
<https://emaiannone.github.io>



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

**Next on this lecture**



Fabio Palomba

## *Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Methodological Perspective*

- (1) How does a general predictive task work?
- (2) How can MSR support vulnerability prediction?
- (3) Which data processing activities are required for predictive tasks?
- (4) What are the pitfalls when developing a vulnerability prediction model?
- (5) What are the current limitations and challenges you are called to face?



Emanuele Iannone

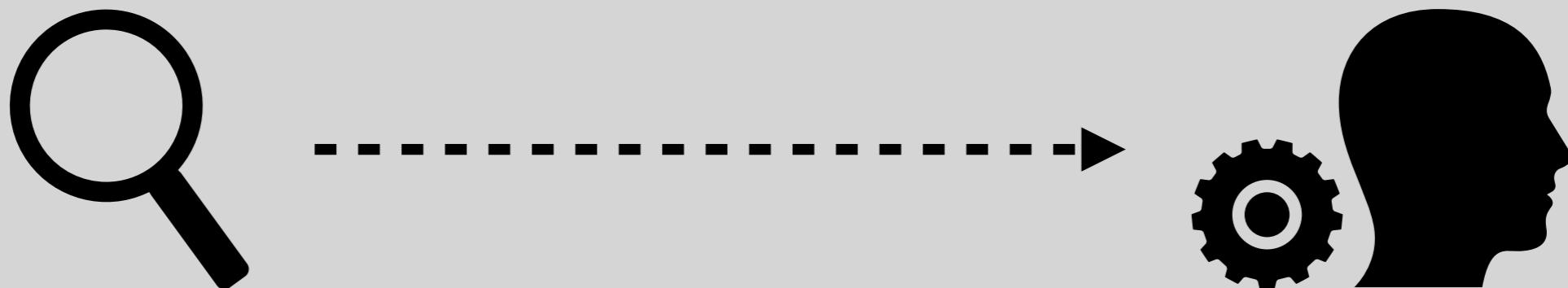
## *Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

- (1) What kind of data do we have to collect?
- (2) How do we query the data sources?
- (3) How can we make mining smart and efficient?
- (4) How do we process the collected data?
- (5) How do we prepare the data for the prediction models?

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

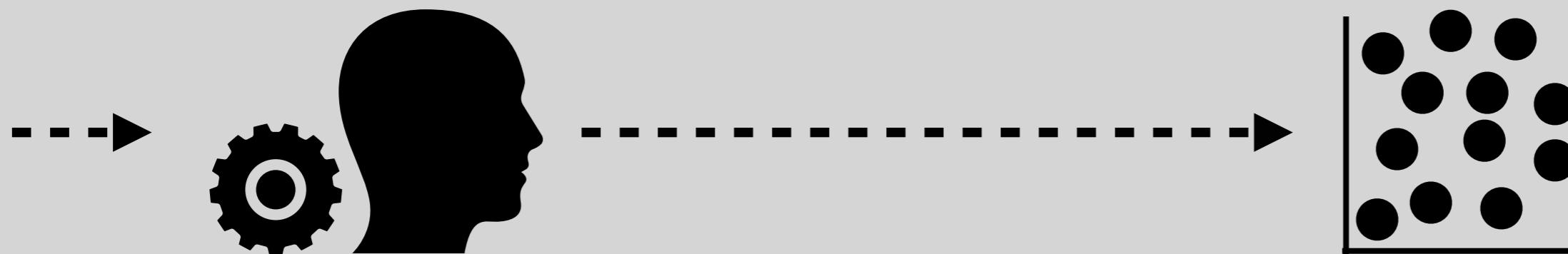


Predicting vulnerabilities needs...  
**machine learning**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

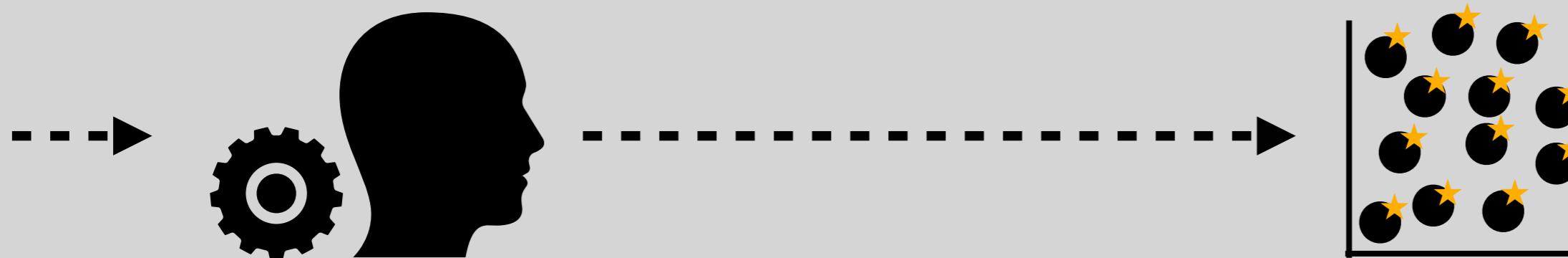


But machine learning needs...  
**lots of data...**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

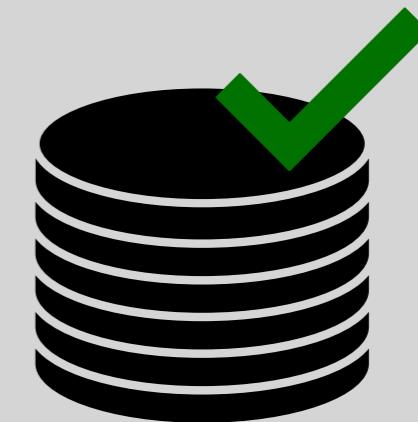
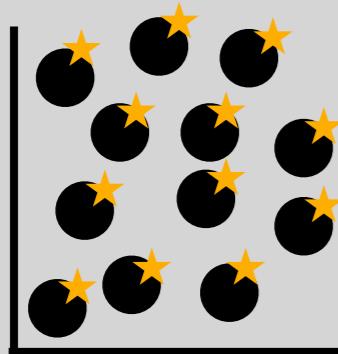


But machine learning needs...  
**lots of data...**  
**possibly of high quality**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

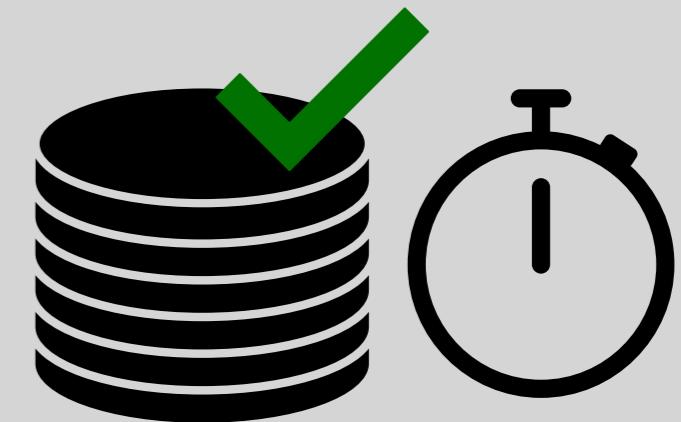
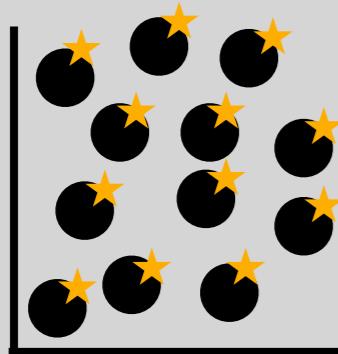


But high quality data needs...  
**reliable data sources...**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



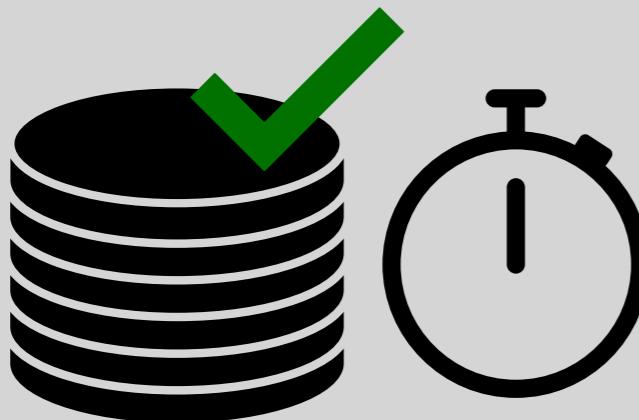
But high quality data needs...  
**reliable data sources...**  
and **time!**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Mining data **IS** time-consuming! We must be smart and efficient to **minimize the times we go back to our steps** and re-do everything.



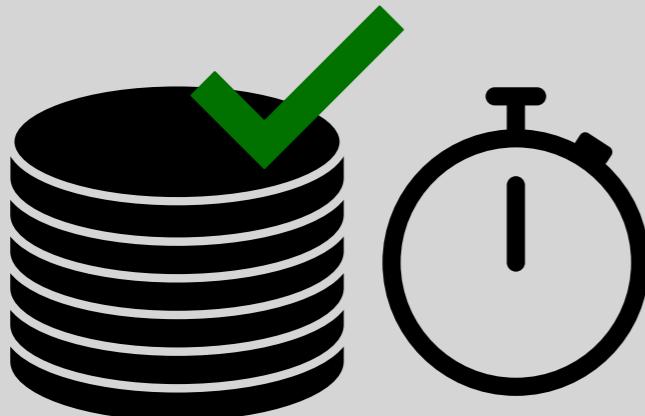
# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Mining data **IS** time-consuming! We must be smart and efficient to **minimize the times we go back to our steps** and re-do everything.

To minimize such a risk, it is good to ask ourselves:



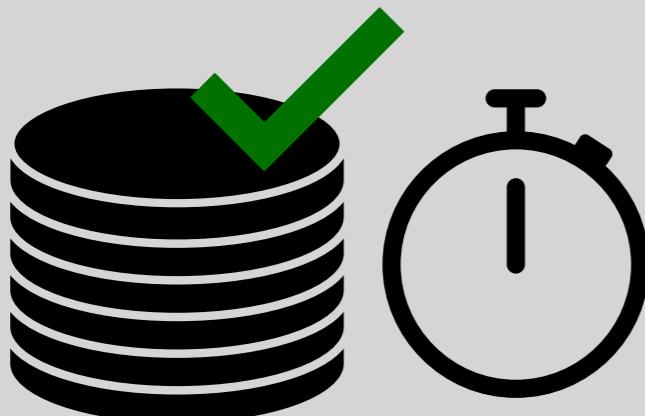
What kind of predictions do I want to make?  
What do I want to achieve?  
How do I plan to use the collected data?

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Mining data **IS** time-consuming! We must be smart and efficient to **minimize the times we go back to our steps** and re-do everything.

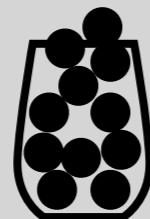


To minimize such a risk, it is good to ask ourselves:



What kind of predictions do I want to make?  
What do I want to achieve?  
How do I plan to use the collected data?

Answering these questions helps to avoid collecting:



**Too much data**  
(pointless workload)



**Too few data**  
(no good models)

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file  
in a specific project is vulnerable.**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



Build a VPM that determines whether **a C file** in a specific project is vulnerable.

The object of the classification is files written in C: the ML models will train on a set of C files and make their predictions on C files. We have to mine data associated with C files; the more, the better.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



Build a VPM that determines whether a C file  
in a specific project is vulnerable.

We can collect historical data regarding that specific project, go for synthetic C files (transfer learning), or even a mixture. This is a **methodological decision** that must be taken seriously.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



Build a VPM that determines whether a C file  
in a specific project **is vulnerable**.

We need information that helps a classifier **recognize the differences** between a vulnerable C file and a “safe” C file. In other words, we need the data that will be used to **encode/extract the features**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file  
in a specific project is vulnerable.**

Let's assume we find a dataset  
with records of past vulnerable C  
files observed in a project.

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

A small icon of a computer screen showing a code editor interface with a cursor.

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

- **Lines of Code**
- **# Functions**
- **# System Calls**
- **Tokens Stream**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000



```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

- Lines of Code ✓
- # Functions ✓
- # System Calls ✓
- Tokens Stream ✓

**Candidate Features**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

Any data that could be used to extract/encode features should be collected. We can store the “high-level” data to save storage space, and only later can we run the algorithms to extract the features.

- **Lines of Code** ✓
- **# Functions** ✓
- **# System Calls** ✓
- **Tokens Stream** ✓

**Candidate Features**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

Any data that could be used to extract/encode features should be collected. We can store the “high-level” data to save storage space, and only later can we run the algorithms to extract the features.



**Source Code** ✓

**Candidate Features**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

→ **Author's name**

→ **Source Code** ✓

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



**Author's name** 

Metadata is not necessarily useful as features. The author's name does not make sense and could be harmful: it might give the classifiers a “shortcut” to make distorted predictions (exploiting **spurious correlations**).



**Source Code** 

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

→ **Author's name**

! Features are not the only reason why we mine data. There are **other purposes** for which we can use the mined data.

→ **Source Code**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000



```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



**Creation Date.** It can be used to filter out specific files.  
For instance, we could drop **outdated files**, as we think they are not reliable enough for today's predictions.

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



**Creation Date.** It can be used to filter out specific files. For instance, we could drop **outdated files**, as we think they are not reliable enough for today's predictions.

It's not rare that we re-use the same "high-level" data for multiple purposes, i.e., data selection and features.



**Source code (again).** It can be used to filter out additional files. For instance, we could drop **empty files, not parsable files, or files with no functions**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



**Creation Date.** It can be used to filter out specific files.  
For instance, we could drop **outdated files**, as we think they are not reliable enough for today's predictions.



It's not rare that we re-use the same "high-level" data for multiple purposes, i.e., data selection and features.



**Source code (again).** It can be used to filter out additional files. For instance, we could drop **empty files, not parsable files, or files with no functions**.



Data Selection

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Let's assume we find a dataset with records of past vulnerable C files observed in a project.

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



But wait, there is more!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```

}



But wait, there is more!

Let's assume we find a dataset with records of past vulnerable C files observed in a project.



**Source code (again).** It can be inspected with a static vulnerability analyzer to obtain the *Nr. warnings*. This metric can be used to **set our ground truth**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Let's be a bit more practical with an example...



**Build a VPM that determines whether a C file in a specific project is vulnerable.**

Author: Emanuele  
Date: 01-09-1996  
Bytes: 20,000

```
int main() {  
    doStuff();  
    doSth();  
    return 0;  
}
```



But wait, there is more!

**Source code (again).** It can be inspected with a static vulnerability analyzer to obtain the *Nr. warnings*. This metric can be used to **set our ground truth**.



**Label Assignment**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

**Next on this lecture**

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

**Candidate Features**

Data intended to **be used for extracting features**, either manually or automatically.

LOC, #Functions, #System Calls, Token Stream, Past Faults, Nr. maintainers

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

**Candidate Features**

Data intended to **be used for extracting features**, either manually or automatically.

LOC, #Functions, #System Calls, Token Stream, Past Faults, Nr. maintainers

**Data Selection**

Data intended to **support the selection of valid instances**, i.e., those that will be seen by the models.

Creation Date, Size, Vulnerability Type/CWE

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

## Candidate Features

Data intended to **be used for extracting features**, either manually or automatically.

LOC, #Functions, #System Calls, Token Stream, Past Faults, Nr. maintainers

## Data Selection

Data intended to **support the selection of valid instances**, i.e., those that will be seen by the models.

Creation Date, Size, Vulnerability Type/CWE

## Label Assignment

Data intended to **support the process of determining the labels** to assign to each instance.

Analysis tool's report, Vulnerability Insertion Date, Vulnerability Type/CWE

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

**Next on this lecture**

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

**Candidate Features**

In this respect, I want to introduce the first of my personal *10 commandments*, i.e., 10 lessons I learned while working on mining data for vulnerability prediction.

**Data Selection**

**Label Assignment**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

Candidate Features

In this respect, I want to introduce the first of my personal *10 commandments*, i.e., 10 lessons I learned while working on mining data for vulnerability prediction.

1

**Thou shalt not be too eager to terminate the mining**

Data Selection

There will **always** be some data that you forgot to collect, and you'll regret it. Take your time, and think!

Label Assignment

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To sum up, we mine data for three main reasons:

Candidate Features

In this respect, I want to introduce the first of my personal *10 commandments*, i.e., 10 lessons I learned while working on mining data for vulnerability prediction.

1

**Thou shalt not be too eager to terminate the mining**

Data Selection

There will **always** be some data that you forgot to collect, and you'll regret it. Take your time, and think!

Label Assignment

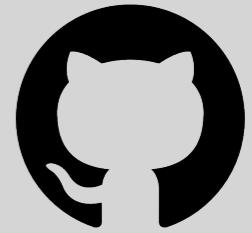


When in doubt, **collect all data available** if it does not cost you too much. If you have no time, do it later... think incrementally!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Fabio showed us many possible data sources: NVD, CVE, GitHub, etc. Depending on many factors, we might have to **query multiple sources**.

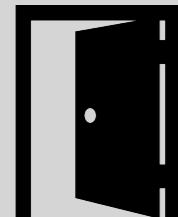
# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Fabio showed us many possible data sources: NVD, CVE, GitHub, etc. Depending on many factors, we might have to **query multiple sources**.



Let's assume we opt for NVD and want to collect **ALL existing CVEs**! The real deal now is to **find an “entry point”**: an *interface* allowing the retrieval of the stored data.

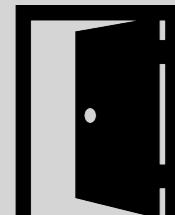
# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Fabio showed us many possible data sources: NVD, CVE, GitHub, etc. Depending on many factors, we might have to **query multiple sources**.



Let's assume we opt for NVD and want to collect **ALL existing CVEs**! The real deal now is to **find an “entry point”**: an *interface* allowing the retrieval of the stored data.



Reliable data sources often come with **public web APIs**, allowing the retrieval of data—in JSON format—with simple HTTP requests. Whenever they exist, it's a good sign.

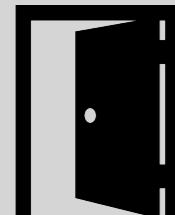
# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Fabio showed us many possible data sources: NVD, CVE, GitHub, etc. Depending on many factors, we might have to **query multiple sources**.



Let's assume we opt for NVD and want to collect **ALL existing CVEs**! The real deal now is to **find an “entry point”**: an *interface* allowing the retrieval of the stored data.



Reliable data sources often come with **public web APIs**, allowing the retrieval of data—in JSON format—with simple HTTP requests. Whenever they exist, it's a good sign.



Let's go check the dedicated page on the NVD website: <https://nvd.nist.gov/vuln/data-feeds>

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

## NVD Data Feeds

### NOTICE

For additional information on the NVD API, please visit the [developers pages](#).

All NIST publications are available in the public domain according to Title 17 of the United States Code, however services which utilize or access the NVD are asked to display the following notice prominently within the application: "This product uses data from the NVD but is not endorsed or certified by the NVD." You may use the NVD name in order to identify the source of the data. You may not use the NVD name, to imply endorsement of any product, service, or entity, not-for-profit, commercial or otherwise.

For information on how to cite the NVD, including the database's Digital Object Identifier (DOI), please consult [NIST's Public Data Repository](#).

## APIs and Data Feed Types

The following table contains links and short descriptions for each API or data feed the NVD offers. Please read how to keep up-to-date with NVD data when using the traditional data feeds.

Users of the data feeds provided on this page must have an understanding of the XML and/or JSON standards and XML or JSON related technologies as defined by [www.w3.org](http://www.w3.org).

Type	Description
CVE and CPE APIs	An alternative to the traditional vulnerability data feed files. The APIs are far more flexible and offer a richer dataset in a single interface compared to the JSON Vulnerability Feeds and CPE Match Feed.
JSON Vulnerability Feeds	Each vulnerability in the file includes a description and associated reference links from the CVE® dictionary feed, as well as CVSS base scores, vulnerable product configuration, and weakness categorization.
CPE Match Feed	A feed that provides the product/platform applicability statement to CPE URI matching based on the CPEs in the official CPE dictionary.
RSS Vulnerability Feeds	An eight day window of security related software flaws.
Vulnerability Translation Feeds	Translations of vulnerability feeds.
Vulnerability Vendor Comments	Comments provided by vendors regarding a particular flaw affecting within a product.
CPE Dictionary	dictionary containing a list of products.
Common Configuration Enumeration (CCE) Reference Data	Reference data for common configuration items.



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

## NVD Data Feeds

### NOTICE

For additional information on the NVD API, please visit the developers pages.

All NIST publications are available in the public domain according to Title 17 of the United States Code, however services which utilize or access the NVD are asked to display the following notice prominently within the application: "This product uses data from the NVD but is not endorsed or certified by the NVD." You may use the NVD name in order to identify the source of the data. You may not use the NVD name, to imply endorsement of any product, service, or entity, not-for-profit, commercial or otherwise.

For information on how to cite the NVD, including the database's Digital Object Identifier (DOI), please consult [NIST's Public Data Repository](#).

### APIs and Data Feeds

The following table contains

Users of the data feeds provided on this page must have an understanding of the XML, JSON, and CPE standards and XML or JSON related technologies as defined by [www.w3.org](#).

NVD website offers many mechanisms to recover its data. This “*CVE and CPE APIs*” seems interesting. Let’s navigate this link.

Type	Description
<a href="#">CVE and CPE APIs</a> 	An alternative to the traditional vulnerability data feed files. The APIs are far more flexible and offer a richer dataset in a single interface compared to the JSON Vulnerability Feeds and CPE Match Feed.
<a href="#">JSON Vulnerability Feeds</a>	Each vulnerability in the file includes a description and associated reference links from the CVE® dictionary feed, as well as CVSS base scores, vulnerable product configuration, and weakness categorization.
<a href="#">CPE Match Feed</a>	A feed that provides the product/platform applicability statement to CPE URI matching based on the CPEs in the official CPE dictionary.
<a href="#">RSS Vulnerability Feeds</a>	An eight day window of security related software flaws.
<a href="#">Vulnerability Translation Feeds</a>	Translations of vulnerability feeds.
<a href="#">Vulnerability Vendor Comments</a>	Comments provided by vendors regarding a particular flaw affecting within a product.
<a href="#">CPE Dictionary</a>	dictionary containing a list of products.
<a href="#">Common Configuration Enumeration (CCE) Reference Data</a>	Reference data for common configuration items.



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



## CVE and CPE APIs

The **CVE and CPE APIs** are the preferred method for staying up to date with the NVD. Users interested in learning where to begin with the API should visit the NVD developers pages.

Benefits of the APIs over the traditional data feeds include:

- The APIs are updated as frequently as our website (unlike the traditional feeds which have explicit update intervals)
- The APIs provide search capabilities based on the Advanced search feature of the website
- The APIs provide CVE and CPE based searching capabilities, including the ability to search for single CVE and CPE entries
- The ability to view only the information that has changed since a given date or time
- Simplified methods of identifying CPE matches to Applicability statements

CVE API Documentation	CPE API Documentation
Automation Support for CVE Retrieval	Automation Support for CPE Retrieval



Reliable data sources often come with **public web APIs**, allowing the retrieval of data—in JSON format—with simple HTTP requests. Whenever they exist, it's a good sign.



Let's go check the dedicated page on the NVD website: <https://nvd.nist.gov/vuln/data-feeds>

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



## CVE and CPE APIs

The **CVE and CPE APIs** are the preferred method for staying up to date with the NVD. Users interested in learning where to begin with the API should visit the NVD developers pages.

Benefits of the APIs over the traditional data feeds include:

- The APIs are updated as frequently as our website (unlike the traditional feeds which have explicit update intervals)
- The APIs provide search capabilities based on the Advanced search feature of the website
- The APIs provide CVE and CPE based searching capabilities, including the ability to search across multiple categories
- The ability to view only the information that has changed since a previous retrieval
- Simplified methods of identifying CPE matches to Applicability statements

We are interested in CVEs,  
not in CPEs.

<a href="#">CVE API Documentation</a>	<a href="#">CPE API Documentation</a>
Automation Support for CVE Retrieval	Automation Support for CPE Retrieval



Reliable data sources often come with **public web APIs**, allowing the retrieval of data—in JSON format—with simple HTTP requests. Whenever they exist, it's a good sign.



Let's go check the dedicated page on the NVD website: <https://nvd.nist.gov/vuln/data-feeds>

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

## Next on this lecture

### Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective

#### Vulnerabilities

This quickstart assumes that you already understand at least one common programming language and are generally familiar with JSON RESTful services. JSON specifies the format of the data returned by the REST service. REST refers to a style of services that allow computers to communicate via HTTP over the Internet.

#### Requests

All requests to the API use the HTTP GET method. The URL stem for making requests is different depending on whether the request is for one specific CVE, or a collection of CVEs. REST parameters allow you to control and customize which vulnerabilities are returned. The parameters are akin to those found on the NVD public vulnerability search page, <https://nvd.nist.gov/vuln/search>.

#### Retrieve a specific CVE

The URL stem for retrieving a single CVE is shown below. Please note how the required `{cveId}` appears in the URL path.

```
https://services.nvd.nist.gov/rest/json/cve/1.0/CVE-2021-41172?addOns=dictionaryCpes
```

#### + Parameters

#### Retrieve a collection of CVE

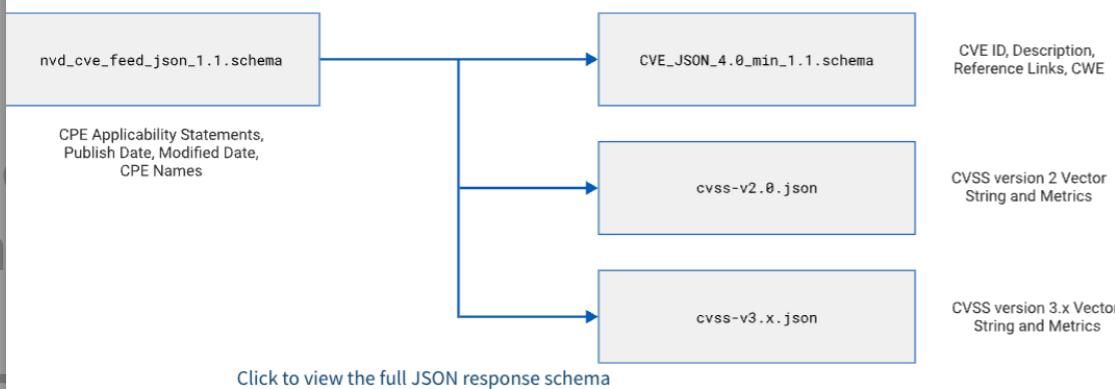
The parameters used to retrieve a collection are intended to limit or filter results. The parameters selected for the request are known as the search criteria, and all parameters should be included in the URL query. Please note how the only difference between the URL for requesting a single CVE and requesting a collection is a single "s".

```
https://services.nvd.nist.gov/rest/json/cves/1.0/
```

#### + Parameters

#### Response

This section describes the response returned by the vulnerability API. Each CVE has a text description and reference links. Vulnerabilities that have undergone NVD analysis include CVSS scores, product applicability statements, and more. The response is based on four JSON schema that were developed independently as part of three separate initiatives. Hence the stylistic differences in data element names. The following diagram shows where the main feed schema is dependent on the other three.



#### + Response Body

Here we discover that with a single HTTP GET request, we can retrieve the info of a given CVE.

itHub, etc. Depending on

ALL existing CVEs! The real  
allowing the retrieval of the

owing the retrieval of data—  
y exist, it's a good sign.

osite: <https://nvd.nist.gov/>

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

## Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

### Vulnerabilities

This quickstart assumes that you already understand at least one common programming language and are generally familiar with JSON RESTful services. JSON specifies the format of the data returned by the REST service. REST refers to a style of services that allow computers to communicate via HTTP over the Internet.

#### Requests

All requests to the API use the HTTP GET method. The URL stem for making requests is different depending on whether the request is for one specific CVE, or a collection of CVEs. REST parameters allow you to control and customize which vulnerabilities are returned. The parameters are akin to those found on the NVD public vulnerability search page, <https://nvd.nist.gov/vuln/search>.

#### Retrieve a specific CVE

The URL stem for retrieving a single CVE is shown below. Please note how the required `{cveId}` appears in the URL path.

`https://services.nvd.nist.gov/rest/json/cve/1.0/CVE-2021-41172?addOns=dictionaryCpes`

#### + Parameters

The example provided (<https://services.nvd.nist.gov/rest/json/cve/1.0/CVE-2021-41172>) returns a JSON with the data of the requested CVE.

```
{ "resultsPerPage":1,"startIndex":0,"totalResults":1,"result":  
+ { "CVE_data_type":"CVE","CVE_data_format":"MITRE","CVE_data_version":"4.0","CVE_data_timestamp":"2022-09-01T1  
0:12Z","CVE_Items":[{ "cve":{ "data_type":"CVE","data_format":"MITRE","data_version":"4.0","CVE_data_meta":  
+ { "ID":"CVE-2021-41172","ASSIGNER":"security-advisories@github.com"}, "problemtype":{ "problemtype_data":  
+ [ { "description":{ "lang":"en","value":"CWE-79" } } ] }, "references":{ "reference_data":{ "url":"https://  
github.com/AntSword-Store/AS_Redis/issues/1","name":"https://github.com/AntSword-Store/AS_Redis/issues/  
1","refsource":"MISC","tags":{ "Exploit","Issue Tracking","Third Party Advisory" } }, "url":"https://  
github.com/Medicean/AS_Redis/security/advisories/GHSA-j8j6-f829-w425","name":"https://github.com/Medicean/  
AS_Redis/security/advisories/GHSA-j8j6-f829-w425","refsource":"CONFIRM","tags":{ "Third Party Advisory" } },  
+ { "url":"https://mp.weixin.qq.com/s/yjuG6DLT_bSRnpqgPj21Xw","name":"https://mp.weixin.qq.com/s/  
yjuG6DLT_bSRnpqgPj21Xw","refsource":"MISC"  
[ ... ] }
```



However, we first need the full list of CVEs... otherwise, we won't know which CVE to request.

Here we discover that with a single HTTP GET request, we can retrieve the info of a given CVE.

itHub, etc. Depending on

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

## Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

### Vulnerabilities

This quickstart assumes that you already understand at least one common programming language and are generally familiar with JSON RESTful services. JSON specifies the format of the data returned by the REST service. REST refers to a style of services that allow computers to communicate via HTTP over the Internet.

### Requests

All requests to the API use the HTTP GET method. The URL stem for making requests is different depending on whether the request is for one specific CVE, or a collection of CVEs. REST parameters allow you to control and customize which vulnerabilities are returned. The parameters are akin to those found on the NVD public vulnerability search page, <https://nvd.nist.gov/vuln/search>.

#### Retrieve a specific CVE

The URL stem for retrieving a single CVE is shown below. Please note how the required `{cveId}` appears in the URL path.

```
https://services.nvd.nist.gov/rest/json/cve/1.0/CVE-2021-41172?addOns=dictionaryCpes
```

#### + Parameters

#### Retrieve a collection of CVE

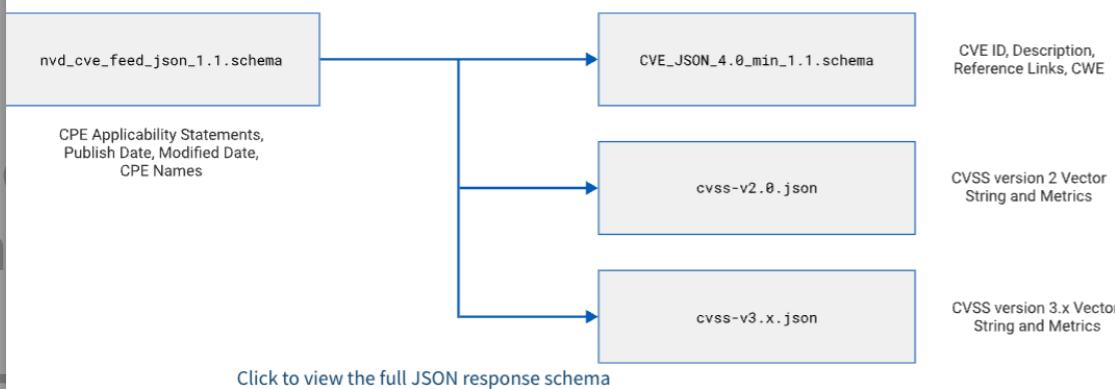
The parameters used to retrieve a collection are intended to limit or filter results. The parameters selected for the request are known as the search criteria, and all parameters should be included in the URL query. Please note how the only difference between the URL for requesting a single CVE and requesting a collection is a single "s".

```
https://services.nvd.nist.gov/rest/json/cves/1.0/
```

#### + Parameters

### Response

This section describes the response returned by the vulnerability API. Each CVE has a text description and reference links. Vulnerabilities that have undergone NVD analysis include CVSS scores, product applicability statements, and more. The response is based on four JSON schema that were developed independently as part of three separate initiatives. Hence the stylistic differences in data element names. The following diagram shows where the main feed schema is dependent on the other three.



#### + Response Body



itHub, etc. Depending on

Fortunately, we can collect  
multiple CVEs at once... right?

owing the retrieval of data—  
y exist, it's a good sign.

osite: <https://nvd.nist.gov/>

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

## Vulnerabilities

This quickstart assumes that you already understand at least one common programming language and are generally familiar with JSON RESTful services. JSON specifies the format of the data returned by the REST service. REST refers to a style of services that allow computers to communicate via HTTP over the Internet.

### Requests

All requests to the API use the HTTP GET method. The URL stem for making requests is different depending on whether the request is for one specific CVE, or a collection of CVEs. REST parameters allow you to control and customize which vulnerabilities are returned. The parameters are akin to those found on the NVD public vulnerability search page, <https://nvd.nist.gov/vuln/search>.

#### Retrieve a specific CVE

The URL stem for retrieving a single CVE is shown below. Please note how the required `{cveId}` appears in the URL path.

`https://services.nvd.nist.gov/rest/json/cve/1.0/CVE-2021-41172?addOns=dictionaryCpes`

+ Parameters

#### Retrieve a collection of CVE

The parameters used to retrieve a collection are intended to limit or filter results. The parameters selected for the request are known as the search criteria, and all parameters should be included in the URL query. Please note how the only difference between the URL for requesting a single CVE and requesting a collection is a single "s".

`https://services.nvd.nist.gov/rest/json/cves/1.0/`

+ Parameters

Let's try this one: <https://services.nvd.nist.gov/rest/json/cves/1.0/>

```
{"resultsPerPage":20,"startIndex":0,"totalResults":183571,"result": [{"CVE_data_type":"CVE","CVE_data_format":"MITRE","CVE_data_version":"4.0","CVE_data_timestamp":"2022-09-01T10:35Z","CVE_Items":[{"cve":{"data_type":"CVE","data_format":"MITRE","data_version":"4.0","CVE_data_meta":{"ID":"CVE-2022-3072","ASSIGNER":"security@huntr.dev","problemtype":{"problemtype_data":[{"description":[{"lang":"en","value":"CWE-79"}]}]},"references":{"reference_data":[{"url":"https://huntr.dev/bounties/9755ae6a-b08b-40a0-8089-c723b2d9ca52","name":"https://huntr.dev/bounties/9755ae6a-b08b-40a0-8089-c723b2d9ca52","refsource":"CONFIRM","tags":[]}]}}]}]}
```

This should be fine, but it seems we just obtained... 20 CVEs?! How is that possible? Many public APIs have two main limitations: **pagination**, which limits the amount of data per response, and **rate limits**, which prevent making too many requests in a short period, especially without an API token (to request).



itHub, etc. Depending on

Fortunately, we can collect  
**multiple CVEs** at once... right?

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: "cve api".

The screenshot shows a Google search results page with the query "cve api" in the search bar. Below the search bar, there are filters for "Tutti", "Notizie", "Immagini", "Video", "Shopping", "Altro", and "Strumenti". The search results indicate "Circa 10.100.000 risultati (0,43 secondi)". The first result is a link to the NVD API documentation: <https://nvd.nist.gov> > developers ▾ Traduci questa pagina. The title is "API Vulnerabilities - NVD". It notes that the URL stem for retrieving a single CVE is shown below, mentioning the required {cveld} placeholder in the URL path. The second result is a link to the cve-search public API documentation: <https://www.cve-search.org> > api ▾ Traduci questa pagina. The title is "cve-search public api.". It describes it as a public API operated by CIRCL, accessible without installing cve-search. The third result is a link to the API Documentation for CVE-Search: <https://cve.circl.lu> > api ▾ Traduci questa pagina. The title is "API Documentation - CVE-Search". It mentions that some API calls require authentication, which is done in one of two ways. The fourth result is a link to the cve-search Common Vulnerabilities and Exposures (CVE) documentation: <https://www.circl.lu> > services > c... ▾ Traduci questa pagina. The title is "cve-search Common Vulnerabilities and Exposures (CVE)". It describes it as a Public Web API of cve-search, accessible via a web interface and an HTTP API.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: "cve api".

cve api

Tutti Notizie Immagini Video Shopping Altro Strumenti

Circa 10.100.000 risultati (0,43 secondi)

<https://nvd.nist.gov> › developers ▾ Traduci questa pagina

**API Vulnerabilities - NVD**

The URL stem for retrieving a single **CVE** is shown below. Please note how the required {cveld} appears in the URL path. <https://services.nvd.nist.gov/rest/json...>

<https://www.cve-search.org> › api ▾ Traduci questa pagina

**cve-search public api!**

A public API of cve-search is operated by CIRCL and can be accessed without installing cve-search. The same API is accessible if cve-search is installed ...

<https://cve.circl.lu> › api ▾ Traduci questa pagina

**API Documentation - CVE-Search**

Some API calls require authentication. These are colored yellow in the API Query list. Authentication is done in one of two ways:

<https://www.circl.lu> › services › c... ▾ Traduci questa pagina

**cve-search Common Vulnerabilities and Exposures (CVE)**

Public Web API of cve-search — cve-search is accessible via a web interface and an HTTP API. cve-search is an interface to search publicly known ...



We discover **CVE Search**, a tool that automatically imports CVEs into *MongoDB*. CVE Search also exposes public web API operated by *Computer Incident Response Center Luxembourg (CIRCL)* initiative. Let's take a look.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: “cve api”.

## API

A public API of cve-search is operated by **CIRCL** and can be accessed without installing cve-search. The same API is accessible if cve-search is installed internally.

### Public Web API of cve-search

The HTTP API outputs JSON.

Browse vendor and product

To get a JSON with all the vendors:

```
curl https://cve.circl.lu/api/browse
```

To get a JSON with all the products associated to a vendor:

```
curl https://cve.circl.lu/api/browse/microsoft
```

Browse CVEs per vendor/product

To get a JSON with all the vulnerabilities per vendor and a specific product:

```
curl https://cve.circl.lu/api/search/microsoft/office
```

Get CVE per CVE-ID

To get a JSON of a specific CVE ID:

```
curl https://cve.circl.lu/api/cve/CVE-2010-3333
```

Get the last updated CVEs

To get a JSON of the last 30 CVEs including CAPEC, CWE and CPE expansions:

```
curl https://cve.circl.lu/api/last
```

Get more information about the current CVE database

To get more information about the current databases in use and when it was updated:

```
curl https://cve.circl.lu/api/dbInfo
```



We can get the list of all the vendors...

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: "cve api".

## API

A public API of cve-search is operated by **CIRCL** and can be accessed without installing cve-search. The same API is accessible if cve-search is installed internally.

### Public Web API of cve-search

The HTTP API outputs JSON.

Browse vendor and product

To get a JSON with all the vendors:

```
curl https://cve.circl.lu/api/browse
```

To get a JSON with all the products associated to a vendor:

```
curl https://cve.circl.lu/api/browse/microsoft
```

Browse CVEs per vendor/product

To get a JSON with all the vulnerabilities per vendor and a specific product:

```
curl https://cve.circl.lu/api/search/microsoft/office
```

Get CVE per CVE-ID

To get a JSON of a specific CVE ID:

```
curl https://cve.circl.lu/api/cve/CVE-2010-3333
```

Get the last updated CVEs

To get a JSON of the last 30 CVEs including CAPEC, CWE and CPE expansions:

```
curl https://cve.circl.lu/api/last
```

Get more information about the current CVE database

To get more information about the current databases in use and when it was updated:

```
curl https://cve.circl.lu/api/dbInfo
```



We can get the list of all the vendors...

...the list of their products...

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

**Next on this lecture**

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: “cve api”.

## API

A public API of cve-search is operated by **CIRCL** and can be accessed without installing cve-search. The same API is accessible if cve-search is installed internally.

### Public Web API of cve-search

The HTTP API outputs JSON.

Browse vendor and product

To get a JSON with all the vendors:

```
curl https://cve.circl.lu/api/browse
```

To get a JSON with all the products associated to a vendor:

```
curl https://cve.circl.lu/api/browse/microsoft
```

Browse CVEs per vendor/product

To get a JSON with all the vulnerabilities per vendor and a specific product:

```
curl https://cve.circl.lu/api/search/microsoft/office
```

Get CVE per CVE-ID

To get a JSON of a specific CVE ID:

```
curl https://cve.circl.lu/api/cve/CVE-2010-3333
```

Get the last updated CVEs

To get a JSON of the last 30 CVEs including CAPEC, CWE and CPE expansions:

```
curl https://cve.circl.lu/api/last
```

Get more information about the current CVE database

To get more information about the current databases in use and when it was updated:

```
curl https://cve.circl.lu/api/dbInfo
```



We can get the list of all the vendors...

...the list of their products...

...and the CVEs of each product.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

If these limitations are too tight, we must look for something different. Let's make a quick Google search: "cve api".

## API

A public API of cve-search is operated by **CIRCL** and can be accessed without installing cve-search. The same API is accessible if cve-search is installed internally.

### Public Web API of cve-search

The HTTP API outputs JSON.

Browse vendor and product

To get a JSON with all the vendors:

```
curl https://cve.circl.lu/api/browse
```

To get a JSON with all the products associated to a vendor:

```
curl https://cve.circl.lu/api/browse/microsoft
```

Browse CVEs per vendor/product

To get a JSON with all the vulnerabilities per vendor and a specific product:

```
curl https://cve.circl.lu/api/search/microsoft/office
```

Get CVE per CVE-ID

To get a JSON of a specific CVE ID:

```
curl https://cve.circl.lu/api/cve/CVE-2010-3333
```

Get the last updated CVEs

To get a JSON of the last 30 CVEs including CAPEC, CWE and CPE expansions:

```
curl https://cve.circl.lu/api/last
```

Get more information about the current CVE database

To get more information about the current databases in use and when it was updated:

```
curl https://cve.circl.lu/api/dbInfo
```



We can get the list of all the vendors...

...the list of their products...

...and the CVEs of each product.

But even in this case, we might need several requests for the complete list of CVEs. Apparently, there are no particular limitations...

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

**Next on this lecture**

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Querying an API is not that difficult. We can opt for:

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Querying an API is not that difficult. We can opt for:

**Command-line tools.** For example, wget.

```
 wget https://cve.circl.lu/api/cve/cve-2020-1234
```

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Querying an API is not that difficult. We can opt for:

**Command-line tools.** For example, wget.

```
 wget https://cve.circl.lu/api/cve/cve-2020-1234
```

**Dedicated libraries.** For example, Python's requests.

```
def call_api(url, try_num=1, max_try=None):
    try:
        headers = {'Accept': 'application/json'}
        return requests.get(url, headers=headers).json()
    except:
        time.sleep(2**try_num + random() * 0.01)
        if max_try and try_num < max_try:
            return call_api(url, try_num=try_num+1)
        else:
            return None
```

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To recap, when does public web API worth it?

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To recap, when does public web API worth it?



**When there are no limits!** This is undoubtedly the best-case scenario, but it is not quite common as you would expect... or there might be a catch, e.g., poor-quality data are returned.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

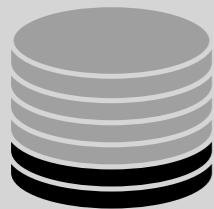
Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To recap, when does public web API worth it?



**When there are no limits!** This is undoubtedly the best-case scenario, but it is not quite common as you would expect... or there might be a catch, e.g., poor-quality data are returned.



**When we just need a sample of the full content.** For instance, we are only interested in collecting the CVEs published since 2019 or those with CVSS Base Score = 10. Not suitable for large-scale mining.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

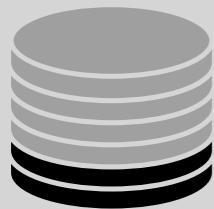
Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

To recap, when does public web API worth it?



**When there are no limits!** This is undoubtedly the best-case scenario, but it is not quite common as you would expect... or there might be a catch, e.g., poor-quality data are returned.



**When we just need a sample of the full content.** For instance, we are only interested in collecting the CVEs published since 2019 or those with CVSS Base Score = 10. Not suitable for large-scale mining.



**When we just need to enrich a set of known data.** For instance, we already own some CVEs and want to add more data using the “single CVE” endpoint.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.



JSON format



JSON format

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.



JSON format

Extended dump  
(daily, per year)



JSON format

Extended dump  
(daily, full)

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.



JSON format

Extended dump  
(daily, per year)

“Recent” dump  
(2-hourly)



JSON format

Extended dump  
(daily, full)

**Partial dumps**  
**(daily, CVEs and references)**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

APIs are cool, but they are not the only entry point to data sources...

If we closely inspect both the NVD and CVE Search websites, we discover the existence of **dumps** (called *data feeds*), i.e., JSON files containing a subset of the full content of the databases.



JSON format

Extended dump  
(daily, per year)

“Recent” dump  
(2-hourly)



Constantly monitor their websites as things can change through time. For example, by the end of 2023, NVD will only offer APIs.



JSON format

Extended dump  
(daily, full)

**Partial dumps**  
**(daily, CVEs and references)**

I tend to use this one, as it's “just” less than 200MB. I integrate the missing data with API calls or other methods (e.g., scraping).

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.



Cell “Data Collection”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```



Cell “Data Collection”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

We read the file and get the JSON of all CVEs, which becomes a dictionary in Python.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

We loop through CVEs and query additional data using CVE Search API.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

We update the in-memory data structure.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

We periodically save the processed data into our storage.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

We periodically save the processed data into our storage.



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

**JSON**

Human-readable, handles nested data, nice diff, memory inefficient (can be compacted).

**CSV**

Straightforward and memory efficient, good diff, can't handle nested data.

**SQL DB**

Perfect with deeply nested data, less straightforward to set up, no diff.



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

Choose whichever you think is the best, depending on your needs.  
The important is to...

2

**Honour thy storage**



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

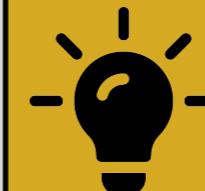
How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

Choose whichever you think is the best, depending on your needs.  
The important is to...

2

**Honour thy storage**



Store in different files, each with a **chunk of M data elements** (CVEs), to reduce the writing time.



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an

```
def store_chunk(items, filepath, num_chunk, lower, upper=None):
    chunk = {k: v for (k, v) in items[lower:upper]}
    num_chunk_str = "0" + str(num_chunk) if num_chunk < 10 else str(num_chunk)
    dest_file = join(dirname(filepath), num_chunk_str + "_" + basename(filepath))
    with open(dest_file, "w") as json_file:
        json.dump(chunk, json_file, indent=2)

def store_cves(data, filepath, chunk_size=5000, rewrite_past_chunks=False):
    items = [(k, v) for k, v in data.items()]
    selected = 0
    num_chunk = 0
    while len(items) - selected > chunk_size:
        upper = selected + chunk_size
        if rewrite_past_chunks:
            store_chunk(items, filepath, num_chunk, selected, upper)
        selected = upper
        num_chunk += 1
    if len(items) - selected > 0:
        store_chunk(items, filepath, num_chunk, selected)

    store_cves(cves, cves_file)
```

storage

different files, chunk of M  
data elements (CVEs), to  
reduce the writing time.



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an

```
def store_chunk(items, filepath, num_chunk, lower, upper=None):
    chunk = {k: v for (k, v) in items[lower:upper]}
    num_chunk_str = "0" + str(num_chunk)
    dest_file = join(dirname(filepath), num_chunk_str + ".json")
    with open(dest_file, "w") as json_file:
        json.dump(chunk, json_file, indent=4)

def store_cves(data, filepath, chunk_size=5000, rewrite_past_chunks=False):
    items = [(k, v) for k, v in data.items()]
    selected = 0
    num_chunk = 0
    while len(items) - selected > chunk_size:
        upper = selected + chunk_size
        if rewrite_past_chunks:
            store_chunk(items, filepath, num_chunk, selected, upper)
        selected = upper
        num_chunk += 1
    if len(items) - selected > 0:
        store_chunk(items, filepath, num_chunk, selected)

store_cves(cves, cves_file)
```

Basically, each 5000 CVEs, a new file is stored and the writings happen on that file, until reaching 5000 CVEs, and so on...

storage

different files, chunk of M data elements (CVEs), to reduce the writing time.



In this respect, the way we organize our storage is critical. We should decide the **format** and the **mechanism** to store the processed data efficiently.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

The mining heavily relies on this loop. We should put in place some other good practices.

3

**Thou shalt not take hostage your machine**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

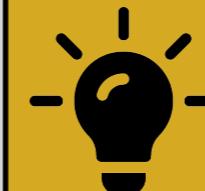
How to collect data from dumps and APIs? The first step is to load the dump into an **iterable data structure** (using any programming language), then loop through each element. During the loop, we can query missing data using the web API.

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Additional mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

The mining heavily relies on this loop. We should put in place some other good practices.

3

**Thou shalt not take hostage your machine**



Mining can take several days/weeks. Put in place several actions to improve your mining scripts.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in raw_cves:
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

**[Tip #1] Monitor the loop progress.**  
Don't make a guess, but monitor the loop status using a **progress bar**. Other than using the terminal, you can also print the progress onto a file—good when running the script on a server.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in tqdm(raw_cves):
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

**[Tip #1] Monitor the loop progress.**  
Don't make a guess, but monitor the loop status using a **progress bar**. Other than using the terminal, you can also print the progress onto a file—good when running the script on a server.

I recommend the TQDM library, which implements a good-looking progress bar by just wrapping the iterable structure in a `function`.

80% |  | 160/200 [02:00<08:00, 1.3it/s]

If each element takes about the same time, we have a good estimate of the duration of the loop.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for cve in tqdm(raw_cves):
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
store_cves(cves, cves_file)
```

**[Tip #2] Save intermediate results.** Storing all the processed data at the end of the loop is not a smart move— things can go wrong (power outage, accidental SIGKILL), and we have to start over. However, doing it at every iteration can be costly. Hence, storing data every K iterations is a good **balance between speed and safety**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for idx, cve in enumerate(tqdm(raw_cves)):
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
    if idx + 1 == len(raw_cves) or \
       len(cves) % 100 == 0:
        store_cves(cves, cves_file)
```

We need the iteration index.

We write into storage at the last iteration or once every 100 CVEs successfully processed.

**[Tip #2] Save intermediate results.** Storing all the processed data at the end of the loop is not a smart move— things can go wrong (power outage, accidental SIGKILL), and we have to start over. However, doing it at every iteration can be costly. Hence, storing data every K iterations is a good **balance between speed and safety**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = {}
for idx, cve in enumerate(tqdm(raw_cves)):
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
    if idx + 1 == len(raw_cves) or \
       len(cves) % 100 == 0:
        store_cves(cves, cves_file)
```

**[Tip #3] Restart from the intermediate results.** The intermediate results are not only meant for back-ups but can be used to **find the point where to start again after an interruption**. Basically, we read the file of processed CVEs and avoid re-processing them.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

## Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open('cves.json') as in_file:
    dump_content.update(json.load(in_file))
    as in_file:
        .read())
raw_cves = dump_content['cves']
cves_file = "path/to/cves.json"
cves = read_cves(cves_file)
for idx, cve in enumerate(tqdm(raw_cves)):
    if cve in cves:
        continue
    resp = call_api(endpoint, cve)
    # Additional processing
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
    if idx + 1 == len(raw_cves) or \
       len(cves) % 100 == 0:
        store_cves(cves, cves_file)
```

We initialize by reading any file containing processed CVEs.

If a CVE has already been processed, skip it.

**[Tip #3] Restart from the intermediate results.** The intermediate results are not only meant for back-ups but can be used to **find the point where to start again after an interruption**. Basically, we read the file of processed CVEs and avoid re-processing them.

```
def read_cves(filepath):
    dn = dirname(filepath)
    bn = basename(filepath)
    data = {}
    for f in sorted(listdir(dn)):
        path = join(dn, f)
        if exists(path) and getsize(path):
            with open(path, "r") as in_file:
                data.update(json.load(in_file))
return data
```

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content["cves"]
cves_file = "path/to/cves.json"
cves = read_cves(cves_file)
for idx, cve in enumerate(tqdm(raw_cves)):
    if cve in cves:
        continue
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
    if idx + 1 == len(raw_cves) or \
       len(cves) % 100 == 0:
        store_cves(cves, cves_file)
```

**[Tip #4] Enable graceful interruption.** Sometimes we have to interrupt the script manually. However, interrupting during a file writing has the risk of corrupting its content: we lose our progress. Hence, we can intercept the CTRL+C (SIGINT), set a flag to true, and stop the loop in a safe state (e.g., between iterations) to avoid damage.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

```
endpoint = "https://cve.circl.lu/api/cve"
dump_content = {}
with open("path/to/dump.json", "r") as in_file:
    dump_content = json.loads(in_file.read())
raw_cves = dump_content
cves_file = "path/to/cves.json"
cves = read_cves(cves_file)
stop_signal = False
signal.signal(signal.SIGINT, signal_handler)
for idx, cve in enumerate(tqdm(raw_cves)):
    if stop_signal:
        break
    if cve in cves:
        continue
    resp = call_api(join(endpoint, cve))
    # Addition mining goes here
    cves[cve] = {
        "cwe": resp["cwe"],
        "cvss": resp["cvss"]
    }
    if idx + 1 == len(raw_cves) or \
       len(cves) % 100 == 0:
        store_cves(cves, cves_file)
```

We can use a custom handler triggered when a SIGINT is received.

**[Tip #4] Enable graceful interruption.** Sometimes we have to interrupt the script manually. However, interrupting during a file writing has the risk of corrupting its content: we lose our progress. Hence, we can intercept the CTRL+C (SIGINT), set a flag to true, and stop the loop in a safe state (e.g., between iterations) to avoid damage.

The handler just sets a global variable to true, queried at the start of each iteration.

```
def signal_handler(sig, frame):
    global stop_signal
    print('Going to gracefully stop')
    stop_signal = True
```

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

It is known that **things never go as expected**. It happens continuously: we run our fantastic mining script before ending the work day. We go back home, arrange something with our friends, eat or drink something, and then go to sleep.

The next day at the office we discovered something terrible: the script crashed 10 minutes after we closed our office's door!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

It is time to introduce another commandment:

4

**Thou shalt not ignore  
corner cases**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

It is time to introduce another commandment:

4

**Thou shalt not ignore  
corner cases**



Make your script **resistant to unforeseen events**. We could wrap all the loop logic inside a great *try-catch* block that captures any unexpected exception.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

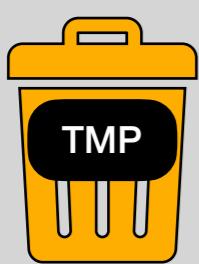
It is time to introduce another commandment:

4

**Thou shalt not ignore corner cases**



Make your script **resistant to unforeseen events**. We could wrap all the loop logic inside a great *try-catch* block that captures any unexpected exception.



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

It is time to introduce another commandment:

4

**Thou shalt not ignore corner cases**



Make your script **resistant to unforeseen events**. We could wrap all the loop logic inside a great *try-catch* block that captures any unexpected exception.



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.

Later, we can inspect this file, try to figure out a way to fix the issue, and re-run the loop to include the discarded CVEs.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.

Later, we can inspect this file, try to figure out a way to fix the issue, and re-run the loop to include the discarded CVEs.

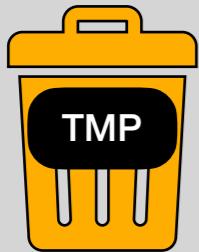


Yet, there might be cases when we can't fix the CVE, e.g., some critical data are entirely missing or too malformed to be fixed.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.

Later, we can inspect this file, try to figure out a way to fix the issue, and re-run the loop to include the discarded CVEs.



Yet, there might be cases when we can't fix the CVE, e.g., some critical data are entirely missing or too malformed to be fixed.

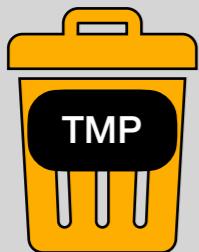


The idea is to solve as many issues as possible. When not possible, we **permanently store** the problematic CVEs in another dedicated file, with the associated exception message as well.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.

Later, we can inspect this file, try to figure out a way to fix the issue, and re-run the loop to include the discarded CVEs.



Yet, there might be cases when we can't fix the CVE, e.g., some critical data are entirely missing or too malformed to be fixed.



The idea is to solve as many issues as possible. When not possible, we **permanently store** the problematic CVEs in another dedicated file, with the associated exception message as well.

**REJECTED**

The CVE number had been allocated but was not approved for various reasons (duplicate, not a real vulnerability, etc.). **Example:** CVE-2012-2701

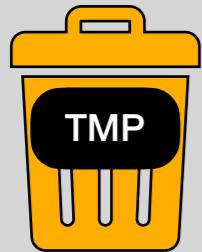
**NON-EXISTENT**

The CVE “number” appears in a dump but does not point to a really-existing CVE due to the invalid format. **Examples:** CVE20163325, CVE-2012-087

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.



The idea is to solve as many issues as possible. When not possible, we **permanently store** the problematic CVEs in another dedicated file, with the associated exception message as well.

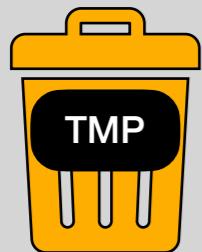
5

**Thou shalt not keep  
your secrets**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



The idea is to catch the exception and skip that iteration. At the same time, we **temporarily store** the problematic CVEs in a dedicated file, with the associated exception message as well.



The idea is to solve as many issues as possible. When not possible, we **permanently store** the problematic CVEs in another dedicated file, with the associated exception message as well.

5

**Thou shalt not keep  
your secrets**



Keep track of everything, especially the data that you discard. Do it for **transparency**—and so, the study's credibility—and for **your future** self!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.



Cell “Data Preparation”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

6

**Thou shalt apply all  
pre-processing at once**



Cell “Data Preparation”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

6

**Thou shalt apply all  
pre-processing at once**



Data preparation is not a transaction. We make an initial preparation to arrange the data in an **exportable format**. Then, we further prepare the data to extract the dataset for **training and testing**.



Cell “Data Preparation”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid



Cell “Data Preparation”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

7

**Thou shalt not put your  
faith in the collected data**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

It's essential to **profile** our data with sufficient effort to understand their nature and decide how to handle them. Investing no time in doing this will cost you a lot. Here is another commandment:

7

**Thou shalt not put your  
faith in the collected data**



The data collected are not exempt from errors—e.g., the CVSS Base Score could be 100 due to an extra zero typed. Ensure the data have the values you expect.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

Standardized formats are easier to inspect, so we can **quickly identify data outside our scope**, e.g., CVEs published after 2021-01-01.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

Whenever possible, we should identify possible errors in data and try to fix them. The most common case is **missing data**. There are cases in which null can be safely intended as 0. Other times, null really means “missing information”. If that information is too-critical, we might think of discarding those CVEs.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Explore Data

Standardize Formats

Drop Out-of-scope

Fix/Impute Data

Drop Invalid

After all these steps, we might discover CVEs having “weird data”. As seen in the example before, there might be CVEs with a CVSS Base Score equal to 100. We cannot safely say that the intended number was 10, so it is safer to drop that CVE. **It's better to have less noise than to have lots of data.**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Yet, even after cleaning, we could still have forgotten something, i.e., letting **erroneous data circumvent the filters** or **discarding valid data**. Are we really sure we have implemented everything correctly? We are software engineers, so we should test at least our final results.

8

**Thou shalt not put your  
faith in the processed data**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*



Our set of collected CVEs may need **some further refinement**. Previously, we just checked whether a CVE was “sufficiently valid” to be involved in the study, but there are some other **quality checks** that we should put in place. We initiate the **data preparation** phase.

Yet, even after cleaning, we could still have forgotten something, i.e., letting **erroneous data circumvent the filters** or **discarding valid data**. Are we really sure we have implemented everything correctly? We are software engineers, so we should test at least our final results.

8

**Thou shalt not put your  
faith in the processed data**



We should do **post-condition verification**: read the `-files` with the prepared data and ensure all the steps had the intended effect.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.



Cell “Dataset Setup”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10



Cell “Dataset Setup”

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

The first step is to decide what goes into the rows! We have to combine all the various files we obtained in the previous phase and express any data at the targeted level. For example, if the target are *commits*, then each row should represent a commit!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Select the data to be directly used as features or run algorithms to compute additional metrics that could not be mined directly.

9

Thou shalt not be shy  
on the feature set

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Select the data to be directly used as features or run algorithms to compute additional metrics that could not be mined directly.

9

Thou shalt not be shy  
on the feature set



Try to involve as many features as possible.  
Consider reasonable metrics only, i.e., those  
that (might) have some correlation with the  
label. Avoid **shortcut features**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Select the data and/or run algorithms to assign labels. If this cannot be done, rely on (semi-)manual approaches.

10

Remember the ground truth, to keep it reliable

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

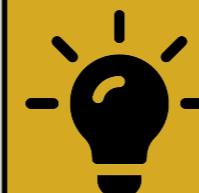
ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Select the data and/or run algorithms to assign labels. If this cannot be done, rely on (semi-)manual approaches.

10

Remember the ground truth, to keep it reliable



Never underestimate the importance of a good, sound, and reliable ground truth. The models' performance is **highly influenced by this choice**.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Now, we can go for the second part of the data preparation: preparing them in a suitable format for the ML models.

ML models expect data in a **tabular format** (e.g., a pandas DataFrame). Each row represents the **observation** (CVE, commit, etc.), depending on the granularity of our task. The columns are dedicated to (1) the ground truth labels and (2) the features.

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

When running empirical studies, we should consider using **validation schemes**, e.g., random or time-aware cross-validations. Since we are preparing the data for the learners, we can already prepare all the  $N$  pairs of training and test sets in this phase.

- ⚠️ Depending on the **degree of realism** of our validation, we might need to re-assign the labels and/or extract some features. Within a validation round, **we are not supposed to look at the data of other rounds!**

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

## Round #1

### Training

Indices
5
83
120
253
1245
2210

### Test

Indices
15
110
2145

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

## Round #1

### Training

Indices
5
83
120
253
1245
2210

Let's suppose Feature #2 is computed depending on the value of Feature #1 **of ALL observations**.

$$f_2(x) = \frac{f_1(x)}{\sum_x f_1(x)}$$

The pre-computed Feature #2 column is invalid! We have to re-compute it again on the training set!

### Test

Indices
15
110
2145

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

## Round #1

Training

Indices	Feature #1	Feature #2
5	1	0.67
83	2	0.13
120	3	0.20
253	4	0.27
1245	3	0.20
2210	2	0.13

$$\sum_x f_1(x) = 15$$

$$f_2(x) = \frac{f_1(x)}{\sum_x f_1(x)}$$

The pre-computed Feature #2 column is invalid! We have to re-compute it again on the training set!

Test

Indices
15
110
2145

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

## Round #1

### Training

Indices	Feature #1	Feature #2
5	1	0.67
83	2	0.13
120	3	0.20
253	4	0.27
1245	3	0.20
2210	2	0.13

$$\sum_x f_1(x) = 15$$

Let's suppose Feature #2 is computed depending on the value of Feature #1 **of ALL observations**.

$$f_2(x) = \frac{f_1(x)}{\sum_x f_1(x)}$$

The pre-computed Feature #2 column is invalid! We have to re-compute it again on the training set!

### Test

Indices
15
110
2145



We store this sum to compute each test instance's Feature #2 value. The test instances **NEVER LOOK** at other test instances!

# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Next on this lecture

*Fundamentals of Mining Software Repositories for Vulnerability Prediction: The Practical Perspective*

Indices	Label	Feature #1	Feature #2	...	Feature #k
0	✗	100	0.1		1
1	★	200	0.05		2
...	...	...	...		...
N	✗	50	0.025		10

Let's suppose we use a traditional random 10-fold cross-validation. We need to create 10 training sets and 10 test sets.

## Round #1

### Training

Indices	Feature #1	Feature #2
5	1	0.67
83	2	0.13
120	3	0.20
253	4	0.27
1245	3	0.20
2210	2	0.13

$$\sum_x f_1(x) = 15$$

Let's suppose Feature #2 is computed depending on the value of Feature #1 **of ALL observations**.

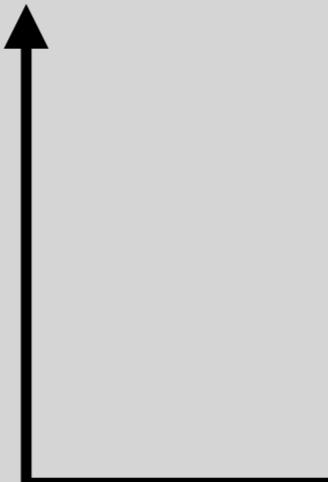
$$f_2(x) = \frac{f_1(x)}{\sum_x f_1(x)}$$

The pre-computed Feature #2 column is invalid! We have to re-compute it again on the training set!

We store this sum to compute each test instance's Feature #2 value. The test instances **NEVER LOOK** at other test instances!

### Test

Indices	Feature #1	Feature #2
15	1	0.67
110	1	0.67
2145	2	0.13





# Mining Software Repositories for Vulnerability Prediction: Lessons Learned, Challenges, and Recommendations

Emanuele Iannone  
Ph.D. Student  
Software Engineering (SeSa) Lab  
University of Salerno

[eiannone@unisa.it](mailto:eiannone@unisa.it)  
<https://EmanueleIannone3>  
<https://emaiannone.github.io>

