Empirical Studies to Discover Vulnerability Analysis Patterns

PI: MEHDI MIRAKHORLI
PRESENTER: JOANNA C. S. SANTOS
Software Vulnerabilities

- Knowledge of *how* vulnerabilities are *discovered* and *resolved* can be used to guide the development of more accurate software assurance tools.

- This knowledge can be used:
  - To develop techniques for *automated vulnerability detection*;
  - Combined with software metrics, data mining and statistical techniques to perform *vulnerability analysis*.
Goals

To empirically **compare** and **validate** the usage of **software metrics** and **automated vulnerability detection tools** to study the feasibility of using a hybrid approach for developing more accurate software assurance tools.
Proposed Work

**Phase 1**
- Establish a **benchmark dataset** to be used to compare different vulnerability detection tools

**Phase 2**
- Use the tools in SWAMP and create a **vulnerability profile** for the projects in our benchmark dataset
- Compute different **software metrics** related to these projects

**Phase 3**
- Utilize machine learning techniques to **learn vulnerability analysis patterns**
Phase 1

- Establish a benchmark dataset to be used to compare different vulnerability detection tools

Currently, vulnerability information is scattered across many sources

- One major source of information is the National Vulnerability Database (NVD)

Goal: augment the NVD to establish a benchmark dataset of vulnerabilities (CVEs)
Phase 1

- Establish a **benchmark dataset** to be used to compare different vulnerability detection tools

  - **Step 1:** Expand and *empirically* validate NIST’s Vulnerability Description Ontology (VDO), a framework for characterizing vulnerabilities

  **Step 2:** Develop **Natural Language Processing** (NLP) techniques to automatically *tag* and *collect* vulnerability attributes from a report as well as several other sources.

  **Step 3:** To Develop automated techniques to **search**, **collect** and **categorize** available **exploits** for each vulnerability.

  **Step 4:** Automate the process of **tracing vulnerabilities** to software releases, components and lines of code and identify **public patches** for that component.

  **Step 5:** Automatically establish **vulnerability life-cycle** for each vulnerability, which contains *time of introduction*, *time of report* and *time of fix*
Phase 1

• Step 1: Expanding and Empirically Validating NIST’s Standard Vocabulary Information

NISTIR 8138 - Vulnerability Description Ontology (VDO)
Phase 1

- Step 1: Expanding and Empirically Validating NIST's Standard Vocabulary Information

Refinements in the Ontology

→ Indicate the affected software component (e.g. file, module)

→ Augment the Impact Method types

![Diagram of affected component and vulnerability lifecycle]

→ Severity

![Diagram of severity level and vulnerability]

→ Exploit codes

![Diagram of exploit code and vulnerability]

→ Provenance per entity

![Diagram of provenance and vulnerability]

Common Attack Pattern Enumeration and Classification
Phase 1

*Step 1: Expanding and Empirically Validating NIST’s Standard Vocabulary Information*

**Validating the Ontology**

1. Select Open Source Projects

2. Combine vulnerability information from many data sources

3. Use the ontology to describe vulnerabilities

4. Verify whether the ontology accurately can describe all the characteristics of vulnerabilities
Phase 1

• Step 2: Develop Natural Language Processing (NLP) techniques to automatically **tag** and **collect** vulnerability **attributes** from several sources.

**Automatically Characterize Vulnerabilities**

1. Extraction of Information
2. Mapping of Information to the Ontology
3. Storage of Data
We will obtain information on exploits from the *Exploit Database, Mailing Lists, Security Focus* and *Issue Tracking Systems*.

**Sec Bug #69646** OS command injection vulnerability in escapeshellarg

From: ab@php.net  Assigned: ab  
Status: Closed  Package: Program Execution  
PHP Version: Irrelevant  OS: Windows  
Private report: No  
CVE-ID: 2015-4642  

[2015-05-15 23:12 UTC] ab@php.net  
Description:  
-------------  
In following is the report from Takayuki Uchiyama. This issue is an OS command injection vulnerability. Do you have a specific case that fails? I have attached the proof-of-concept code to reproduce this issue.

---

PoC Code  
------------------------------------------  
```php  
<?php  
$a = 'a\\';  
$b = 'b -c d\\';  
var_dump( $a, escapeshellarg($a) );  
var_dump( $b, escapeshellarg($b) );  
system( 'php arginfo.php ' . escapeshellarg($a) . ' ' . escapeshellarg($b) )  
?>  
[
```  
[arginfo.php]  
------------------------------------------  
```
<?php  
print( "--- ARG INFO ---\n" );  
var_dump( $argv );  
?>  
-------------------
```
• Identify the differences in syntax trees between the vulnerable version and the fixed version.

• Identify the potential variations of those differences

• Identify the corresponding changes needed to be made to the fixed version to introduce the vulnerability

• Compare changes from all known consecutive versions of the vulnerable changes

• Identify all the vulnerable versions

• **Step 4:** Automate the process of **tracing vulnerabilities** to software releases, components and lines of code and identify **public patches** for that component.
Using our technique that automatically traces vulnerable versions

Phase 1

- Step 5: Automatically establish **vulnerability life-cycle** for each CVE, which contains *time of introduction*, *time of report* and *time of fix*
Proposed Work

Phase 1
- Establish a benchmark dataset to be used to compare different vulnerability detection tools

Phase 2
- Use the tools in SWAMP and create a vulnerability profile for the projects in our benchmark dataset
- Compute different software metrics related to these projects

Phase 3
- Utilize machine learning techniques to learn vulnerability analysis patterns
Phase 2
• Compute metrics and detect vulnerabilities

Step 1
• Select projects from our benchmark dataset (they will contain labelled datasets of vulnerabilities - CVEs)

Step 2
• Run static analysis tools deployed in SWAMP on a large number of these selected open source projects

Step 3
• Identify metrics correlated with vulnerabilities
• Compute these metrics from these projects
Proposed Work

Phase 1
• Establish a benchmark dataset to be used to compare different vulnerability detection tools

Phase 2
• Use the tools in SWAMP and create a vulnerability profile for the projects in our benchmark dataset
• Compute different software metrics related to these projects

Phase 3
• Utilize machine learning techniques to learn vulnerability analysis patterns
Utilize machine learning techniques to learn vulnerability analysis patterns.

Represent the outcome of these tools in a uniform format.

Combine tools and the computed metrics to empirically examine the accuracy of various hybrid solutions (majority voting, metric-based and etc.).

Use machine learning to find analysis patterns in the output of the tools, in the software engineering metrics and in the occurrences of true positive and false positive vulnerabilities in these systems.
Final Considerations

- The long term goals are:
  1. Establish a **benchmark dataset** of vulnerabilities
  2. Empirical study of existing **vulnerability analysis tools** on a large number of open source systems.
  3. Identifying **vulnerability analysis patterns** resulting false positive and true positive in software vulnerability detection tools.
  4. Use software engineering **metrics** that are correlated with vulnerability analysis patterns.
  5. Combining tools and metrics to empirically **examine the accuracy of various hybrid solutions** (majority voting, metric based and etc.) in finding software vulnerabilities.

- Goal #1 is performed in collaboration with NIST
Empirical Studies to Discover Vulnerability Analysis Patterns

PI: MEHDI MIRAKHORLI
PRESENTER: JOANNA C. S. SANTOS