



Approaches to vulnerabilities detection

First assignment from Software Dependability class, 18-19

De Simone Francesco
Di Luccio Luca
M. Tanveer Jan



Vulnerability

“A vulnerability is an instance of a mistake”
- Andy Ozment

The fundamental software engineering terms which I will rely are failure, fault, mistake, and error (...) I have written ‘a mistake.’ His use of ‘error’ is (...) “the amount by which the result is incorrect”

No confusion about whether or not different instances of the same mistake constitute different vulnerabilities



Software Vulnerability Analysis and Discovery

It focus on researching and discovering software vulnerability to make a categorization of those flaws

It examine:

- cause
- position
- implement features

Try to search unknown vulnerability

The idea is avoid problem in first place

Some technique are been developed



The main techniques

- Static Analysis
- Fuzzing
- Penetration Testing
- VDMs



Static Analysis

Evaluating a system or component analyzing;

- form
- structure
- content
- documentation



Method used in static analysis

- String search
 - grep
- tokenization
 - IST4 or RATs
- MARCO
 - Map the stack inspection mechanism on graph
 - Set the privileges
 - Propagate the privileges



Difficulty of static analysis

The main difficulty is that the actual tools are not fully automated

Some of the tools we can use:

- JDeodorant
- JTest
- Some IDE integrate tools for static analysis

We also must consider that static analysis is an undecidable problem (can be reduce in an halting problem)



False positive and False negative problems

A false positive is when something is recognized as a vulnerability, but it actually isn't

They are less alarming but they led to a waste of human resources

False negative occurs when a vulnerability is not reported

They are serious threat



Static analysis is done by:

- Compiler
 - unused variable
 - overflow
 - double free
- Automatic tools
 - Can do a deeper analysis
 - Can have more diagnostic rule

gcc -Wall -pedantic prova.c

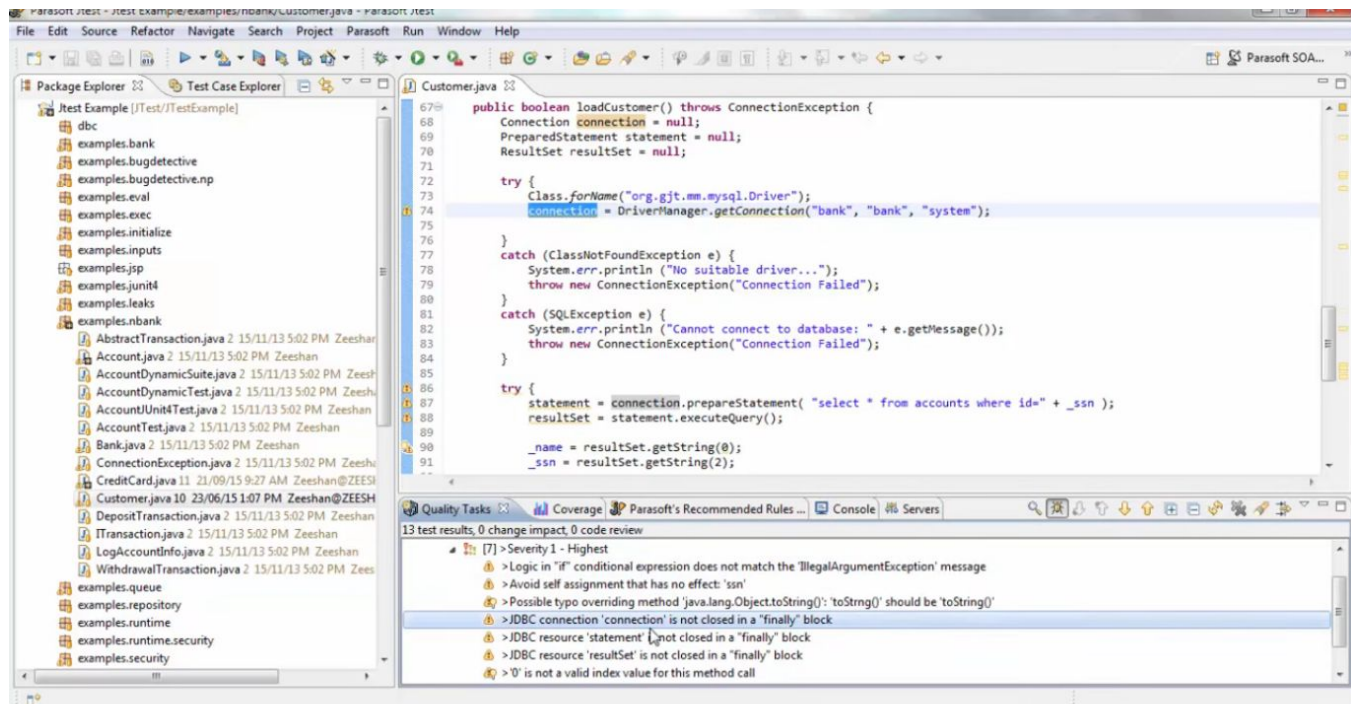
```
#include <stdio.h>
```

```
char *foo(){  
    char* x = malloc(1);  
    x[0] = 0;  
    if(!*x)  
        return "lol";  
    puts("ciao");  
}
```

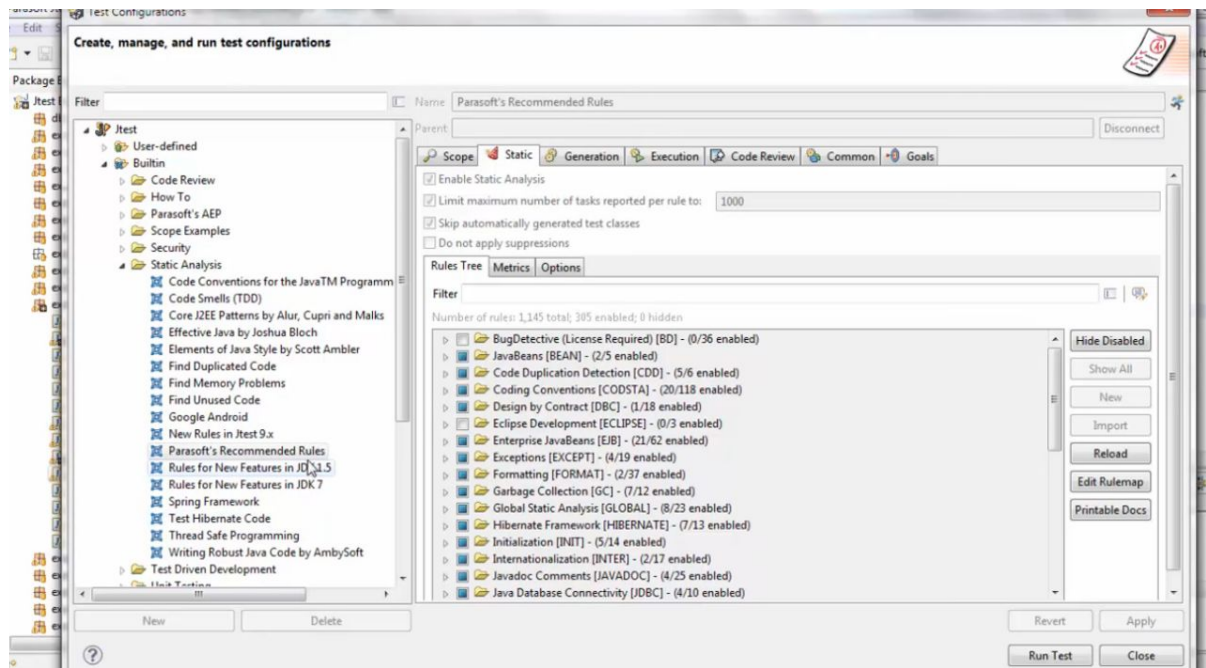
```
int main(){  
    int x;  
    puts(foo());  
}
```

```
prova.c: In function 'foo':  
prova.c:4:15: warning: implicit declaration of function 'malloc' [-Wimplicit-function-declaration]  
    char* x = malloc(1);  
               ^~~~~~  
prova.c:4:15: warning: incompatible implicit declaration of built-in function 'malloc'  
prova.c:4:15: note: include '<stdlib.h>' or provide a declaration of 'malloc'  
prova.c:2:1:  
+#include <stdlib.h>  
  
prova.c:4:15:  
    char* x = malloc(1);  
               ^~~~~~  
prova.c: In function 'main':  
prova.c:12:9: warning: unused variable 'x' [-Wunused-variable]  
    int x;  
      ^  
prova.c: In function 'foo':  
prova.c:9:1: warning: control reaches end of non-void function [-Wreturn-type]  
}  
^
```

Jtest



Jtest



In short:

**We analyze the source code
without compile or execute it
in order to discovery defects or
weaknesses**



Fuzzing

A type of testing where **automated** or **semi-automated** testing techniques are used to discover coding errors. The system is monitored for various exceptions, such as crashing down of the system or failing built-in code, etc.

Fuzz testing was originally developed by Barton Miller at the University of Wisconsin in 1989.

It is usually considered a black-box approach.



How to do Fuzz Testing

1. Identify the target system
2. Identify inputs
3. Generate Fuzzed data
4. Execute the test using fuzzy data
5. Monitor system behavior
6. Log defects



Kinds of Fuzzers

- **Mutation-Based** Fuzzers alter existing data samples to create new test data.
- **Generation-Based** Fuzzers define new data based on the input of the model. It starts generating input from the scratch based on the specification.
- **Protocol-Based** Fuzzers involves writing an array of the specification into the tool then by using model-based test generation technique go through the specification and add irregularity in the data contents, sequence, etc.



Vulnerabilities detected by Fuzz Testing

- Assertion failures and memory leaks
- Invalid input
- Correctness bugs

Fuzzing in a real world scenario



A faulty web-server

For example our web-server may receive handcrafted malicious requests from a client and we must deal with them without interrupting the service.

We send fuzzed data and we monitor the out to better understand where our system fails.

Standard HTTP GET

```
$ GET /index.html HTTP/1.1
```

Anomalous HTTP GET

```
$ AAAAAA...AAAA /index.html HTTP/1.1  
$ GET //////////index.html HTTP/1.1  
$ GET %n%n%n%n%n%.html HTTP/1.1  
$ GET /AAAAAAAAAAAAA.html HTTP/1.1  
$ GET /index.html HTTTTTTTTTTTTTTP/1.1  
$ GET /index.html HTTP/1.1.1.1.1.1.1.1
```

In short:

**We test how the system
accept input data and how it
manages corner cases**



Penetration testing

Evaluates the security of a system by simulating attacks by malicious users.

It is conducted by penetration testers employed by software suppliers usually before the last round of testing before the software is published.



Categories of Penetration Testing

Black-box penetration testing

- No prior knowledge of the system to be tested
- The testers must first determine the extent of the systems before commencing their analysis
- Simulates an attack from someone who is unfamiliar with the system

White-box penetration testing

- The tester has complete knowledge of the infrastructure to be tested (source code, ips, names etc.)
- Simulates what might happen during an "inside job", if the attacker has access to source code, network layouts, and possibly even some passwords

Gray-box penetration testing

- Variations between Black-box and White-box penetration based on what the client ask to do (or offer to share) with the pentester



Two processes of testing

Supplier planning test

- Build risk analysis document
- Make a test plan including:
 - test type (black box, white box etc.)
 - timetables
 - tools
 - requirements

Testers executing test

- Respect the Penetration Testing Execution Standard (PTES)
- The report to submit should contain:
 - reproduction steps
 - severity of the leaks
 - exploit code examples



Must consider

Common misconceptions are:

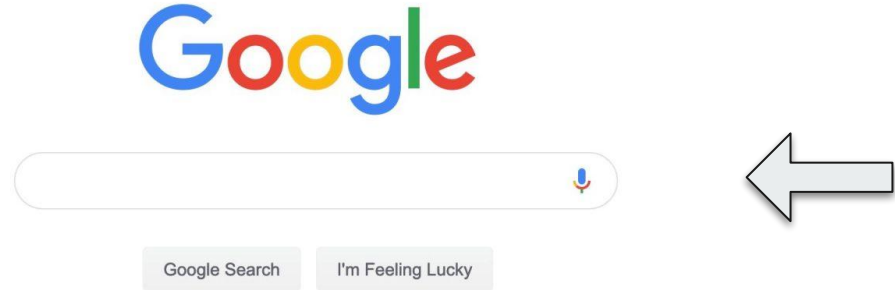
- Security through obscurity
- Thinking that our software is secure because it passed a pentest

Penetration Testing in a real world scenario






XSS on Google Search

The most used form on the web with billions of monthly users. How does a XSS slip away from all the testing made inside google's headquarters? How can we detect a vulnerability in such closed system?




← → ↻ [https://www.google.de/search?q=<noscript><p+title%3D"</noscript><img+src%3Dx+onerror%3Dalert\(1\)>">&cad=h](https://www.google.de/search?q=<noscript><p+title%3D)

 <noscript><p title="</noscript>">  

[All](#) [Maps](#) [Videos](#) [Images](#) [News](#) [More](#) [Settings](#) [Tools](#)

About 1.490 results (0,33 seconds)




A privacy reminder from Google

[REMIND ME LATER](#) [REVIEW](#)

[PayloadsAllTheThings/XSS injection at master · swisskyrepo ... - GitHub](#)
<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20injection> ▼
XSS in Markdown. [a](javascript:prompt(document.cookie)) [a](j a v a s c r i p t :
prompt(document.cookie)) [a](data:text/html;base64 ...

A simple non-compliant html markup, what will happen when we click on the page?

← → ↻ [https://www.google.de/search?q=<noscript><p+title%3D\"/noscript><img+src%3Dx+onerror%3Dalert\(1\)>\">&cad=h](https://www.google.de/search?q=<noscript><p+title%3D\)



`<noscript><p title=\"</noscript>\">`

xss without angle brackets
xss payloads 2018
advanced xss payloads
`\";!--\"<xss>=&{() }`
xss bypass html encoding
xss payloads github
xss cheat sheet github
href xss

[Report inappropriate predictions](#)
[Learn more](#)

[REMINDE ME LATER](#) [REVIEW](#)

[PayloadsAllTheThings/XSS injection at master · swisskyrepo ... - GitHub](#)
<https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20injection> ▼
XSS in Markdown. [a](javascript:prompt(document.cookie)) [a](j a v a s c r i p t :
prompt(document.cookie)) [a](data:text/html;base64 ...

www.google.de says

1

[OK](#)

A simple XSS attack! But how it works?

```

< " <noscript><p title="</noscript><img src=x onerror=alert(1)>"> "
> template.content.children[0]
< ▼<noscript>
    <p title="</noscript><img src=x onerror=alert(1)>"></p>
    </noscript>


> div = document.createElement("div")
< <div></div>

> div.innerHTML = template.innerHTML
< " <noscript><p title="</noscript><img src=x onerror=alert(1)>"></p></noscript> "
✖ ▶ GET http://liveoverflow.com/x 404 (Not Found)

> div
< ▼<div>
    <noscript><p title="</noscript>
    
    "">
    <p></p>
    </div>

```

They usually sanitize the inputs through a `<template>` tag, but there was an error in the parsing of the `<noscripts>` tags. Thus failing the actual sanitation




The image shows a Twitter profile page for Masato Kinugawa. The header features a black and white checkerboard pattern. The profile picture is a circular image with a swirling, optical illusion-like pattern. The bio mentions a vulnerability and a website. The tweet section shows a pinned tweet about an XSS challenge.

Masato Kinugawa
@kinugawamasato
脆弱性を探す時間が幸せ。
masatokinugawa.l0.cm
Iscrizione a gennaio 2010

Tweet 1.337 Following 174 Follower 9.045 Mi piace 1.000 [Segui](#)

Tweet Tweet e risposte Contenuti

 Tweet fissato

 **Masato Kinugawa** @kinugawamasato · 11 apr 2018
I made a small XSS challenge. This is another version of CNY challenge's third level. Can you solve it? :)
vulnerabledoma.in/cny2018/omake....

Discovered by Masato Kinugawa, he submitted it to the google's bounty hunt website. This was a complete black box approach to the problem

In short:

**We test the system with an
hands on approach, we find
vulnerabilities in all the areas**



Vulnerability Discovery Models

Vulnerability discovery model (VDM) is a derivation of a software reliability models apply on vulnerability discovery.

A vulnerability discovery model specifies the general form of vulnerability discovery process by using the principal factors that may affect the software.



VDMs are based on Software Reliability Models

Software reliability is the probability of the software causing a system failure over some specified operating time. Software reliability models are used to estimate software performance measures such as the reliability, number of errors remaining, and the time to next software failure of a partially debugged software package.

The models have two basic types:

- Prediction modeling
 - These models are derived from actual historical data from real software projects
- Estimation modeling
 - The models depend on the assumptions about the fault rate during testing



Two proposed model: AME and AML

AME is a **Effort-based model**

- Rely on the number of users of target systems.

AML is a **logistical model**

- Assumes that vulnerability discovery take place in three phases
 - learning
 - linear phase
 - saturation phase



Problems about VDMs now

- VDMs are based on SRMs model
 - Many assumption of SRMs are wrong in vulnerability discovery
- Improve vulnerability database
 - Develop novel VDMs and assess the existed ones

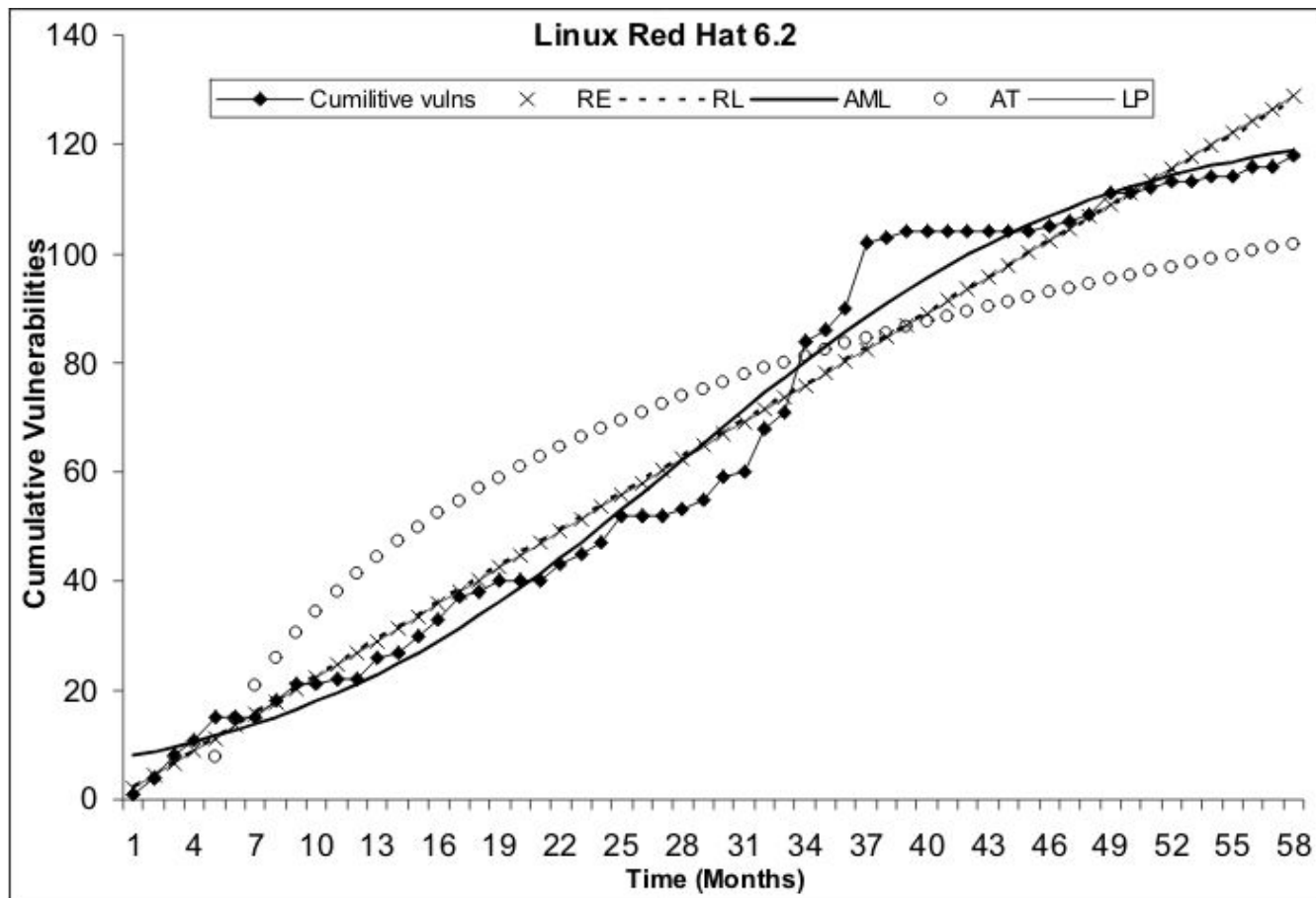
VDMs in a real world scenario



Predict vulnerabilities in Linux Red Hat 6.2

VDMs has been used to predict the discovery of the vulnerabilities in a real operating system. It was strangely accurate





As we see the vulnerabilities discovered followed the expected trend

In short:

**We use VDMs to test
vulnerability of a software
basing our assumption on past
experiences (and probability
that our systems shares
vulnerability with others)**

To recap

- Vulnerabilities must be found and addressed as soon as possible because are root causes of security problems.
- How people can quickly discover vulnerabilities existing in a certain software is one of the main focuses in the security field.
- There are advantages and disadvantages of each technique.



| Techniques | Static Analysis | Fuzzing | Pen. Testing | VDMs |
|---------------|---|---|---|---|
| Advantages | <ul style="list-style-type: none"> •No Need to execute software •Able to find implementation bugs | <ul style="list-style-type: none"> •Simple to implement •Simple to automate •No false positive | <ul style="list-style-type: none"> •No false positive •Simulate a real attack •Simulate social engineering factors •Can expose hard to detect vulnerabilities | <ul style="list-style-type: none"> •In theory, able to predict the rate of vulnerability discovery and the total amount of them •Promising method |
| Disadvantages | <ul style="list-style-type: none"> •Not able to describe software properties •High false positives •Can't find design errors | <ul style="list-style-type: none"> •High randomness •Low degree of generalization | <ul style="list-style-type: none"> •Based on testers knowledge, experience, and time | <ul style="list-style-type: none"> •Lack of general valid VDMs •Lack of VDMs databases |

Recap of all the advantages and disadvantages of the technique introduced here



Thank you for your attention