Finding Bugs in Source Code Using Commonly Available Development Metadata

Devin Cook
Yung Ryn Choe
Drew Hamilton
Auburn University
Sandia National Laboratories
CSET 2015

Sandia National Laboratories
Exceptional service in the national interest

AUBURN

General Problem Statement

Anatomy of an Exploit

Goal:
Find bugs before they are able to be exploited.

So you've found an 0day... what next?

Central Hypothesis
We can use development metadata from a specific project (namely bug reports and changeset information) to find new bugs in that project and others that are difficult to find using traditional static analysis techniques alone.

Background → Literature Review

Literature Review
Finding Bugs
Using Development Metadata
Matching ASTs

Proposal → The Idea

The Idea
Proposed Architecture
Proposed Methodology

Experiment → How it Works

How it Works
Final Architecture

Conclusions → Results

Results
Test Sets
Roundup

Prezi

Devin Cook
Yung Ryn Choe
Drew Hamilton
Auburn University
Sandia National Laboratories
CSET 2015

Finding Bugs in Source Code Using Commonly Available Development Metadata

General Problem Statement

Anatomy of an Exploit

Goal:
Find bugs before they are able to be exploited.

Everyone wins!*

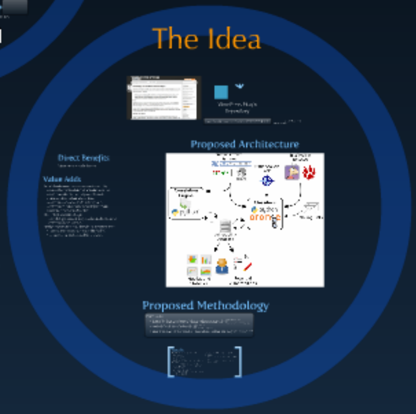
So you've found an 0day... what next?

- Sell it.
- Disclose it.
- Use it for good.
- Use it for evil.

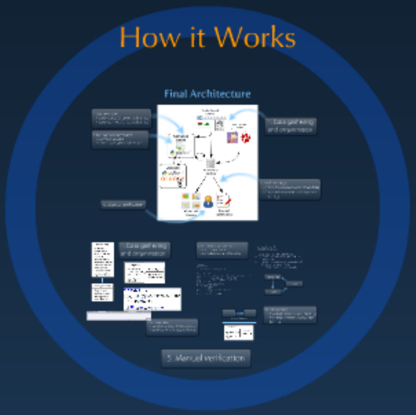
Central Hypothesis
We can use development metadata from a specific project (namely bug reports and changeset information) to find new bugs in that project and others that are difficult to find using traditional static analysis techniques alone.

Background → Literature Review

Proposal → The Idea



Experiment → How it Works



Conclusions → Results

Prezi

General Problem Statement

Anatomy of an Exploit

Goal:
Find bugs before they are able to be exploited.

Everyone wins!*

*except the bad guys

So you've found an 0day... what next?

- Sell it.
- Disclose it.
- Use it for good.
- Use it for evil.

Central Hypothesis

We can use development metadata from a specific project (namely bug reports and changeset information) to find new bugs in that project and others that are difficult to find using traditional static analysis techniques alone.

Anatomy of an Exploit

Prezi

Centra

Goal:

Find bugs before they are able to be exploited.

Everyone wins!*

*except the bad guys

Prezi

So you've found an 0day... what next?

Sell it.

Use it for good.

Disclose it.

Use it for evil.

Everyone wins!*

*except the bad guys

Central Hypothesis

We can use development metadata from a specific project (namely bug reports and changeset information) to find new bugs in that project and others that are difficult to find using traditional static analysis techniques alone.

Finding Bugs in Source Code Using Commonly Available Development Metadata

Devin Cook
Yung Ryn Choe
Drew Hamilton
Auburn University
Sandia National Laboratories
CSET 2015

Sandia National Laboratories
Exceptional service in the national interest

AUBURN

General Problem Statement

Goal:
Find bugs before they are able to be exploited.

Background

Literature Review

Finding Bugs

Proposal

The Idea

Proposed Architecture

Proposed Methodology

Experiment

How it Works

Final Architecture

Conclusions

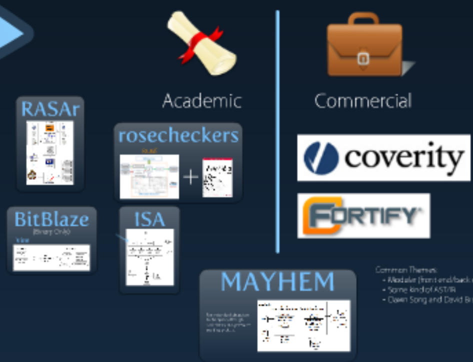
Results

Test Sets

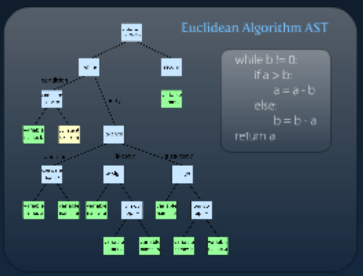
Prezi

Literature Review

Finding Bugs

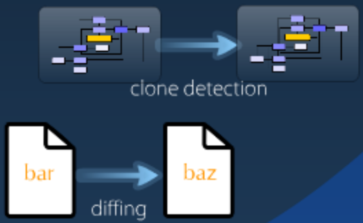


Common Themes:
- Modular (front-end/back-end)
- Some kind of AST/IR
- Dawn Song and David Brumley



Using Development Metadata

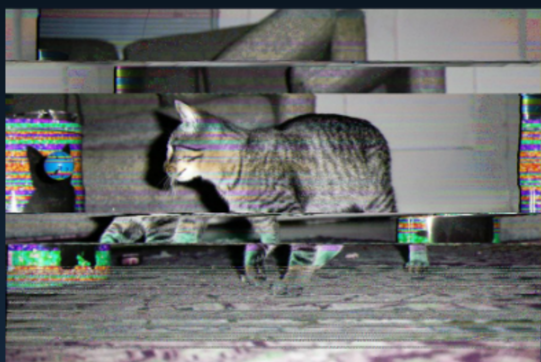
Matching ASTs



BugBench

Prezi

Finding Bu

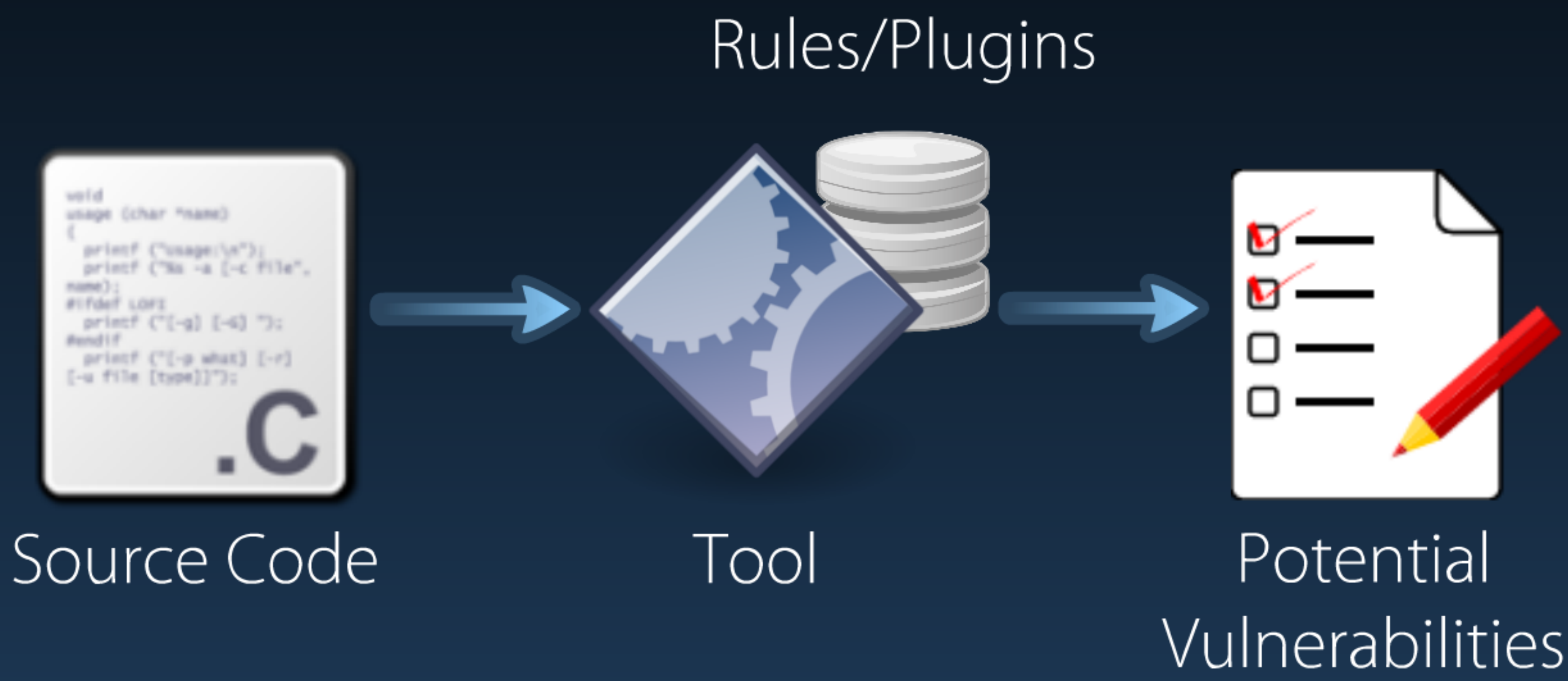


Fuzzing

Dynamic Analysis

Static Analysis

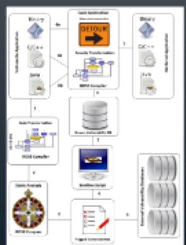
Typical Workflow



Academic

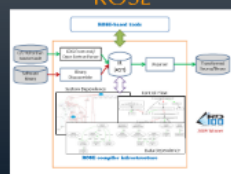
Commercial

RASAr



rosecheckers

ROSE

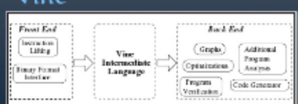


+

BitBlaze

(Binary Only)

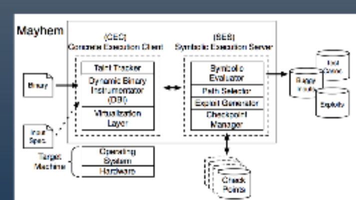
Vine



ISA

MAYHEM

Fairly standard detection techniques (although optimized), but generates working exploits.



coverity

FORTIFY®

Common Themes:

- Modular (front end/back end)
- Some kind of AST/IR
- Dawn Song and David Brumley

Prezi

Literature Review

Finding Bugs

Fuzzing | Dynamic Analysis | Static Analysis

Typical Workflow

Academic

RASAr

rosecheckers

BitBlaze

ISA

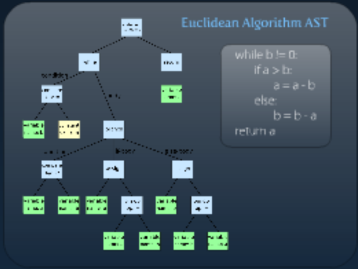
MAYHEM

Common Themes:
- Modular (front-end/back end)
- Some kind of AST/IR
- Dawn Song and David Brumley

Commercial

coverity

FORTIFY



Using Development Metadata

mercurial

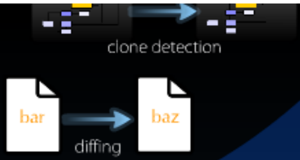
git

trac

Matching ASTs

BugBench

Prezi



The Idea

WordPress Plugin Repository

Direct Benefits

More secure code. Easier.

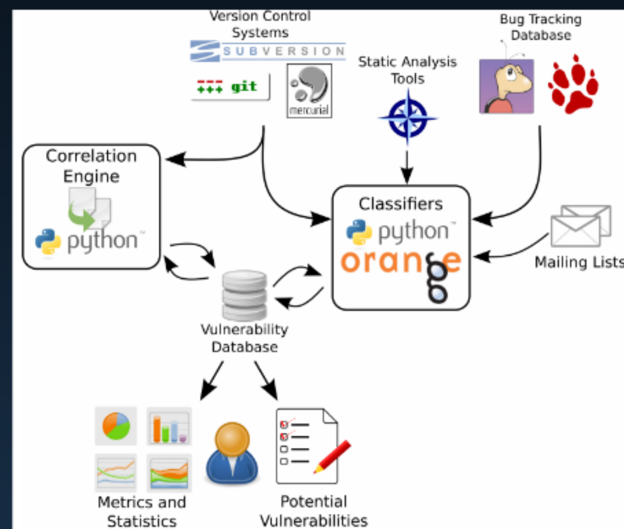
Value Adds

Better development metrics related to security
- committers with a habit of introducing vulns
- common vulns found in a project's code
- number of vulns found over time
- committers good at finding/fixing vulns
- common mistakes certain developers make
- average time to find/fix vulns

Not limited to security bugs
    Everything discussed may also be applied to other common classes of bugs

Interfacing with existing vulnerability detection tools
- incorporation into larger tools (like RASA?)
- using existing tools as additional signals

Proposed Architecture



Proposed Methodology

Core tasks:
- Identify that a piece of data relates to a vuln }
- correlate all related data }
- use that new information to detect other vulns }

Details

SPARE CLOCK CYCLES

HACKING IS FREEDOM.

Subscribe via RSS

HOME | ABOUT ME (AND THIS BLOG) | CODE AND SUCH

18
SEP/11

14

Explo(itlr)ing the WordPress Extension Repos

Today's post is kind of long, so I thought I should warn you in advance by adding an additional paragraph for you to read. I also wanted to provide download links for those who'd rather just read the code. It isn't the cleanest code in the world, so I apologize in advance. I discuss what all of these are for and how they work later on in the post, so if you're confused and/or curious, read on. Downloads:

- Copies of the WordPress theme and plugin repositories can be grabbed via torrent (Please note that the plugin repo has a few directories incomplete/missing; this can be fixed by running my checkout code)
- A new WordPress plugin fingerprinting tool, wpfinger (download). This tool can infer detailed version information on just about every plugin in the WordPress repository. This package also contains some useful libraries for checking out the repositories and scraping plugin rankings, as this is used in the fingerprinting tool.

Intro

After finding an arbitrary file upload vulnerability in 1 Flash Gallery, I became curious as to how many other WordPress plugins made basic security mistakes. The 1 Flash Gallery plugin issue, it seems, is that they CTRL-C-V'd code from a project called Uploadify, which has been known to be vulnerable for quite awhile.

After realizing this, I became curious as to how many plugins make easy-to-spot security mistakes, such as reusing vulnerable libraries or doing such things as include($_REQUEST['lulz']). However, my curiosity was initially somewhat hampered by the fact that downloading and auditing every WordPress plugin one at a time is not only a mind numbing task, but a herculean one as well. And, well, I'm incredibly lazy.

Getting the Repos

So what to do? Well, it turns out that WordPress is nice enough to have public repositories (http://plugins.svn.wordpress.org and http://themes.svn.wordpress.org) containing all plugins that have ever been submitted, as well as every theme. This, of course, was exciting: I could just check this out, whip out some grep-fu, and have my answers.

Recent Posts

- Stack Necromancy: Defeating Debuggers By Raising the Dead
- Exploiting an IP Camera Control Protocol: Redux
- Explo(it|r)ing the WordPress Extension Repos
- 1 Flash Gallery: Arbitrary File Upload
- Sergio Proxy v0.2 Released

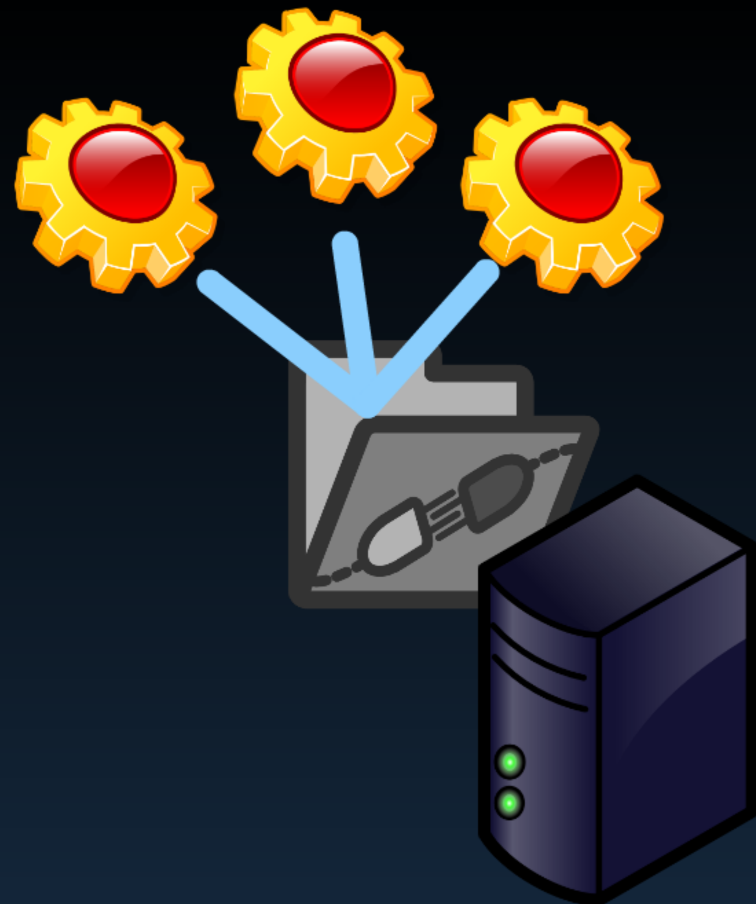
Beer Fund!

Find something particularly useful or interesting? Buy me a beer!

Donate

Blogroll

- Cujef.com
- ha.ckers.org
- Mark Adam's Blog
- Metasploit Blog
- Schneier on Security

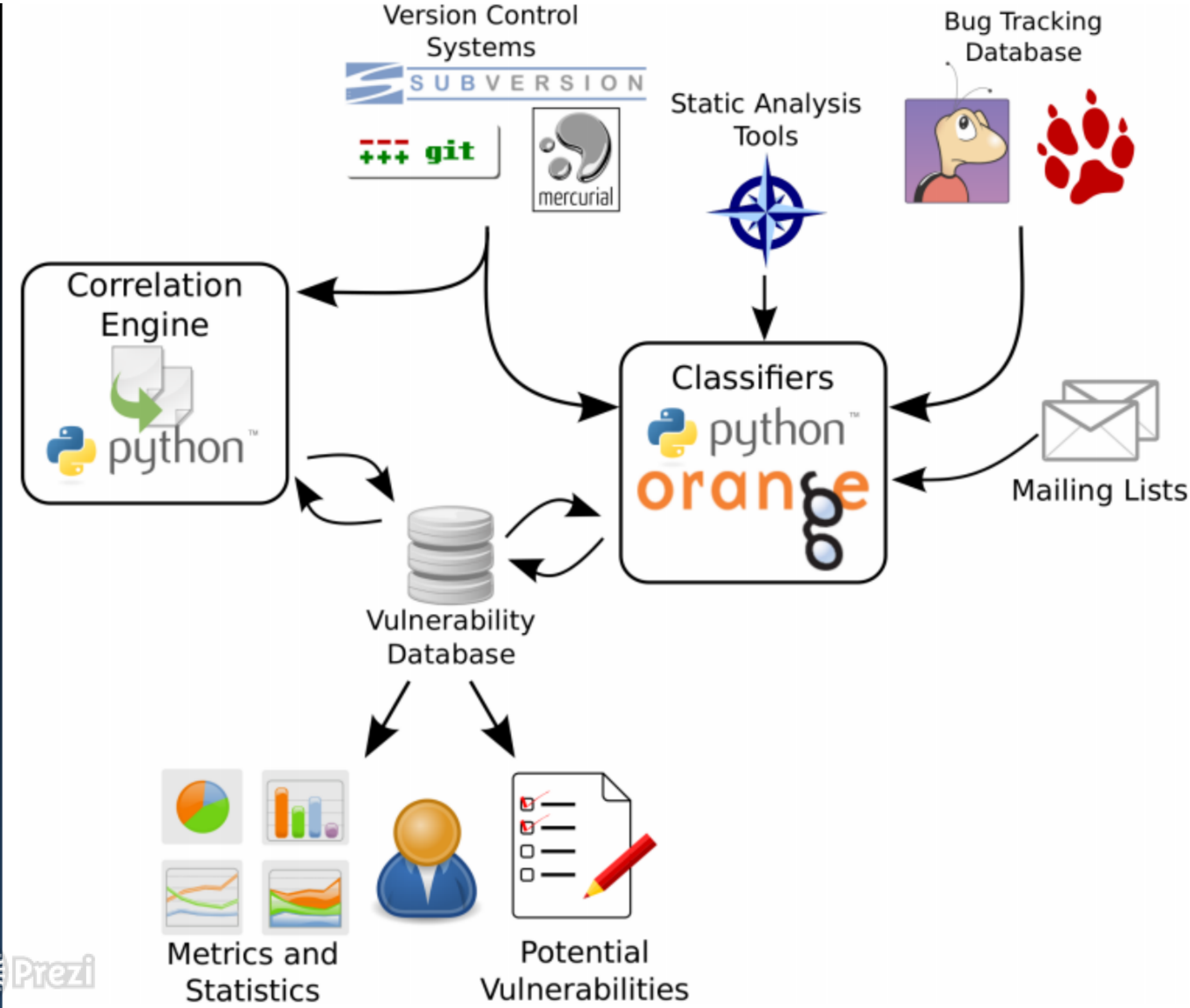


WordPress Plugin Repository

WordPress Plugin Repository

```
$ egrep -i '(include|require)(_once)?(\(|\s+)[^[;)]*\$_(REQUEST|GET|POST|COOKIE)'
```

= 11 unauthenticated RFI vulnerabilities
...little to no effort


Version Control Systems
SUBVERSION
git
mercurial
Static Analysis Tools
Bug Tracking Database
Correlation Engine
python
Classifiers
python
orange
Mailing Lists
Vulnerability Database
Metrics and Statistics
Potential Vulnerabilities
Prezi

Proposed Methodology

Core tasks:

- Identify that a piece of data relates to a vuln } Explicit metadata tags / AI classifiers / relation to other data already identified
- correlate all related data } Dereference links / Recurse / Build DB of indicators/signals
- use that new information to detect other vulns } assign "security score" to commits / examine existing code

Details

Use additional data as input - more "signals"
- bugtrackers
- commit messages
- mailing lists
- public vuln DBs (known-vuln code/packages)

Find patterns
- certain committers that make certain mistakes
- certain files or areas that often contain bugs
- vuln code reuse
- more vulns late at night?
- devs with short names introduce more vulns? - could be anything!

Generate a DB of vulnerable patterns/code

Look for bugs
- Given a new commit, compute the "security score"
- Given a new repo (or just some source), go find some 0days!

Statistics   Vulnerabiliti

Proposed Methodolo

Core tasks:

- Identify that a piece of data relates to a vuln } Explicit metadata tags / AI classifiers / relation to other data already identified
- correlate all related data } Dereference links / Recurse / Build DB of indicators/signals
- use that new information to detect other vulns } assign "security score" to commits / examine existing code

Details

Use additional data as input - more "signals"

- bugtrackers
- commit messages
- mailing lists
- public vuln DBs (known-vuln code/packages)

Find patterns

- certain committers that make certain mistakes

Generate a DB of vulnerable patterns/code

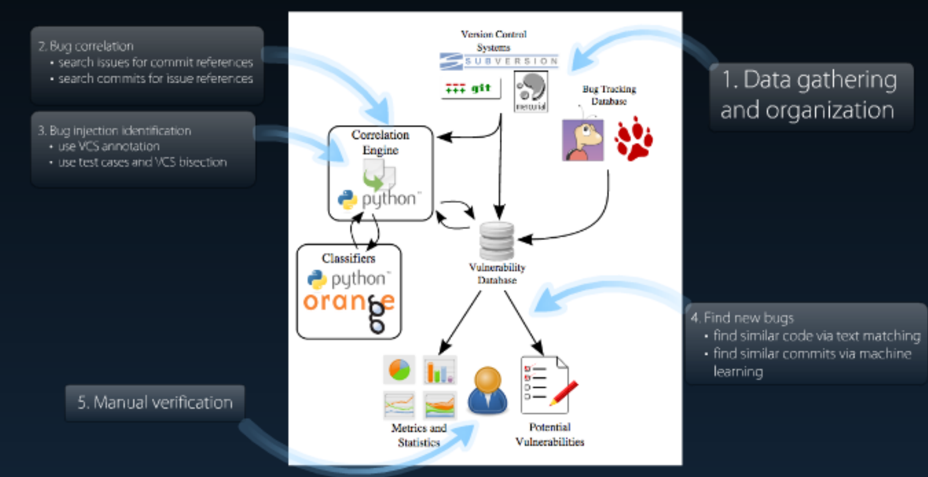
Look for bugs

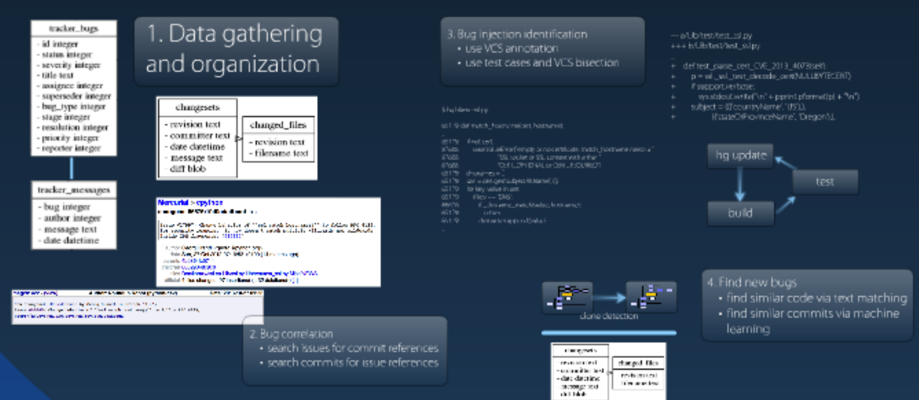
- Given a new commit, compute the "security score"
- Given a new repo (or just some source), go find some 0days!

Prezi

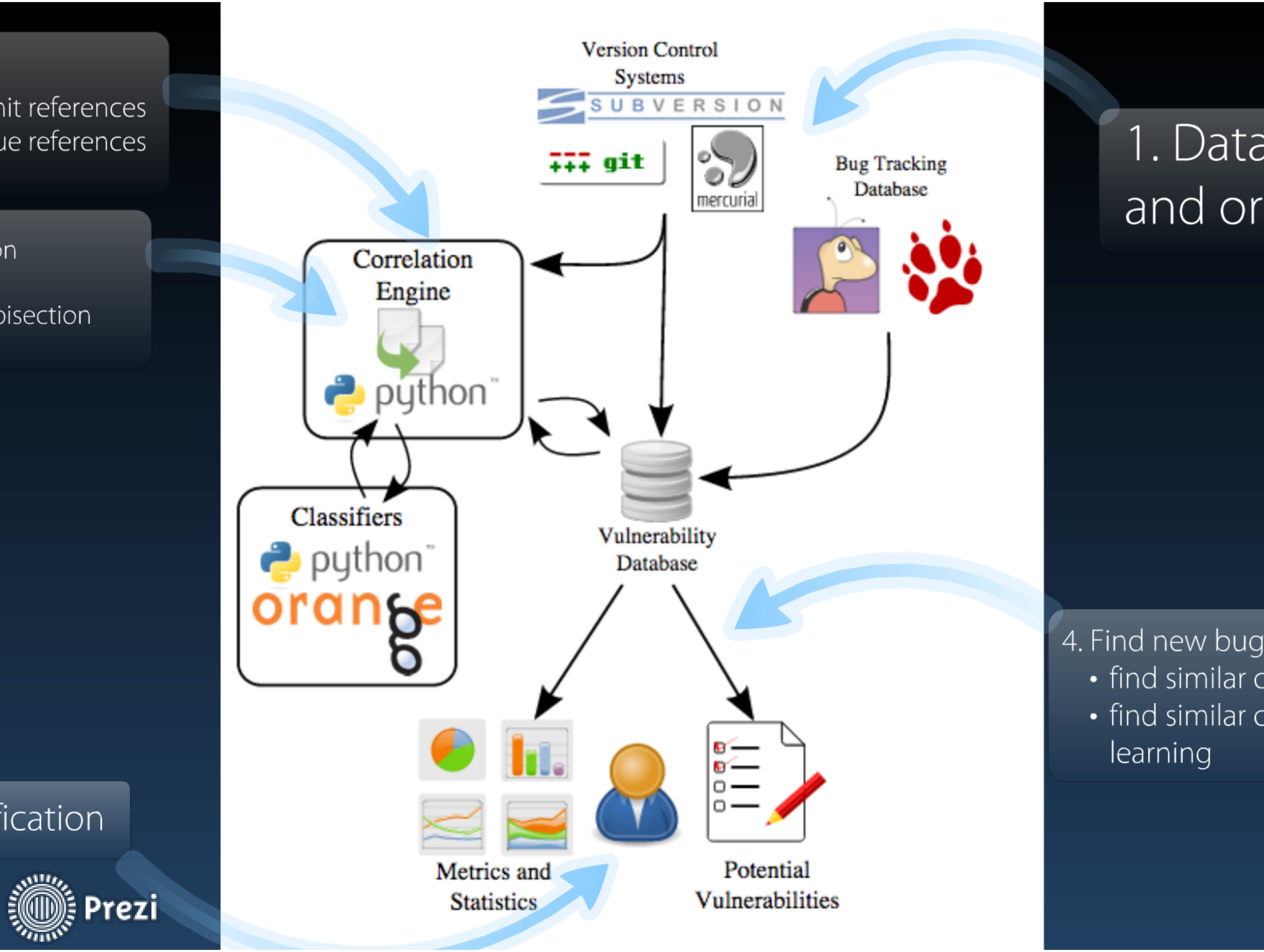
How it Works

Final Architecture





5. Manual verification



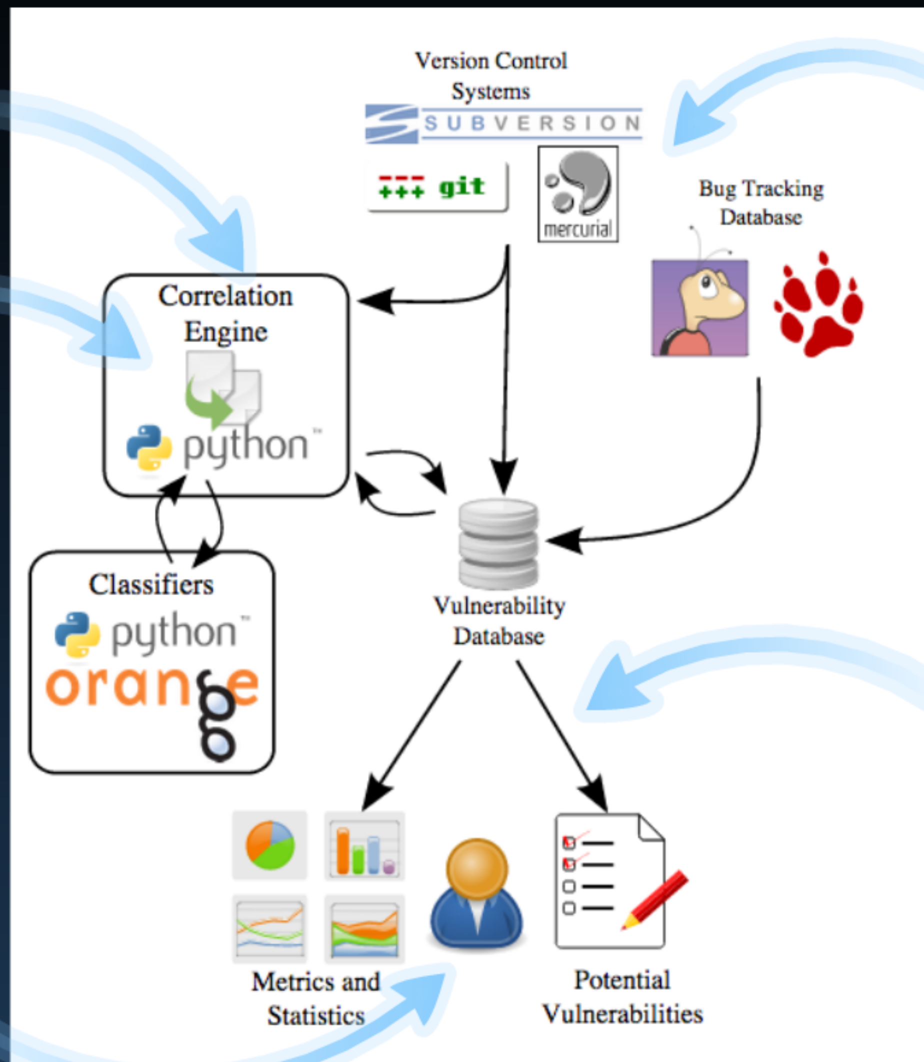
nit references
ue references
on
bisection
fication
Prezi
Version Control Systems
SUBVERSION
git
mercurial
Bug Tracking Database
Correlation Engine
python
Classifiers
python
orange
Vulnerability Database
Metrics and Statistics
Potential Vulnerabilities
1. Data
and or
4. Find new bug
• find similar c
• find similar c
learning

Final Architecture

1. Data gathering and organization

2. Bug correlation
   - search issues for commit references
   - search commits for issue references

3. Bug injection identification
   - use VCS annotation
   - use test cases and VCS bisection



4. Find new bugs
   - find similar code via text matching
   - find similar commits via machine learning

5. Manual verification

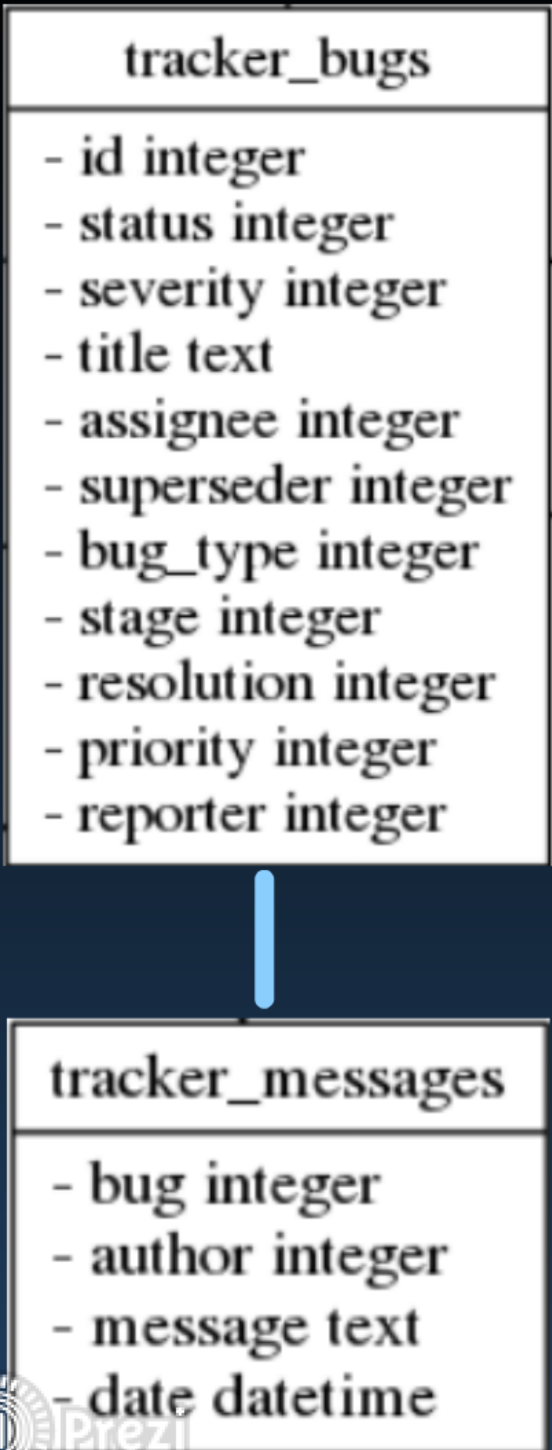
Prezi

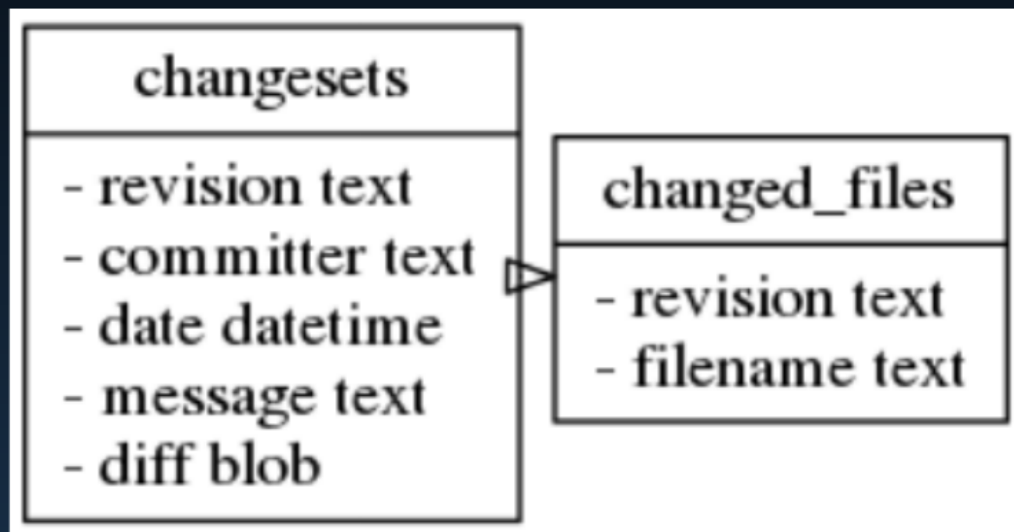
1. Data gathering

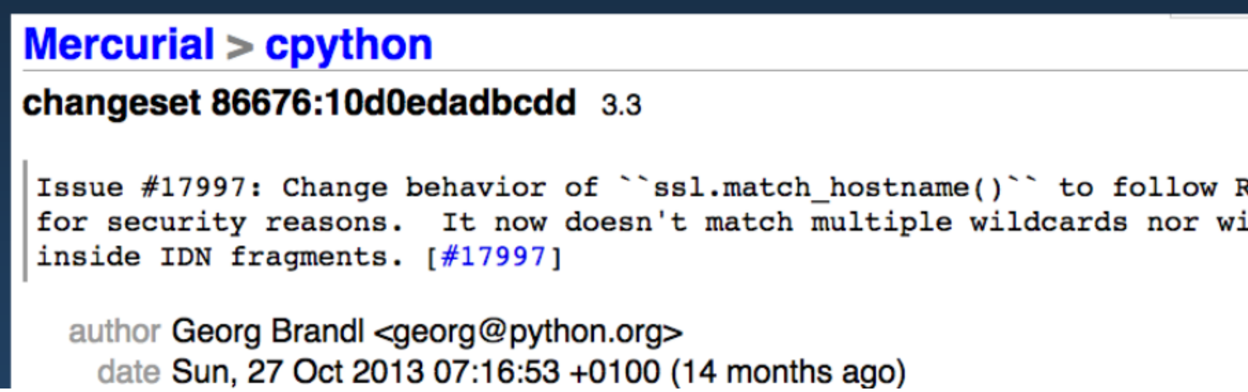
3. Bug injection identification
   - use VCS annotation

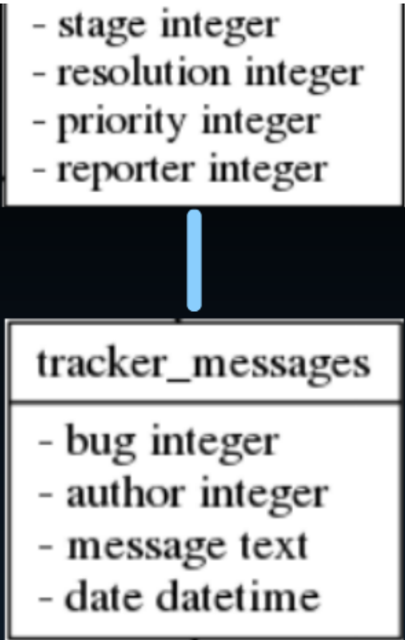
--- a/Lib/test/test_ssl.py
+++ b/Lib/test/test_ssl.py

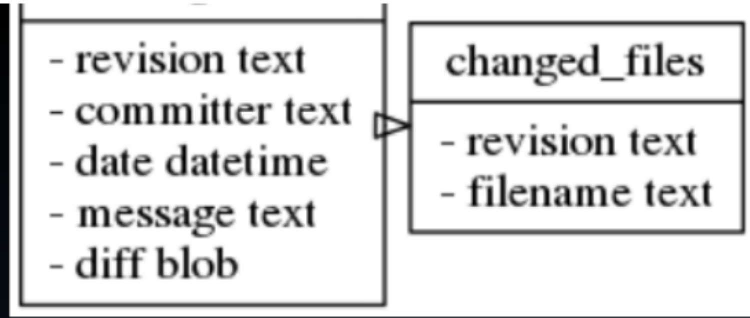
1. Data gathering and organization











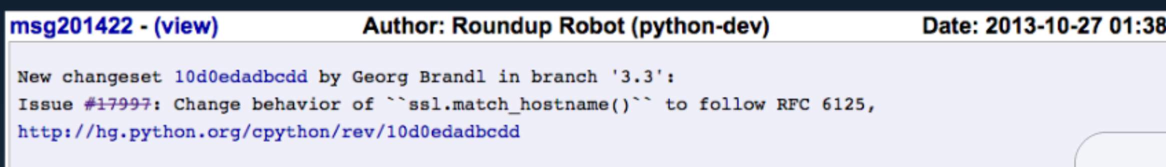
...
65179: c
...
65179:
87688:
87688:
87688:
65179:
65179:
65179:
65179:
86676:
65179:
65179:
...

Mercurial > cpython

changeset 86676:10d0edadbcdd 3.3

```
Issue #17997: Change behavior of ``ssl.match_hostname()`` to follow RFC 6125,
for security reasons.  It now doesn't match multiple wildcards nor wildcards
inside IDN fragments. [#17997]
```

author Georg Brandl <georg@python.org>
date Sun, 27 Oct 2013 07:16:53 +0100 (14 months ago)
parents 4b0364fc5711
children 68029048c9c6
files Doc/library/ssl.rst Lib/ssl.py Lib/test/test_ssl.py Misc/NEWS
diffstat 4 files changed, 97 insertions(+), 32 deletions(-) [+]



msg201422 - (view) **Author: Roundup Robot (python-dev)** **Date: 2013-10-27 01:38**

```
New changeset 10d0edadbcdd by Georg Brandl in branch '3.3':
Issue #17997: Change behavior of ``ssl.match_hostname()`` to follow RFC 6125,
http://hg.python.org/cpython/rev/10d0edadbcdd
```

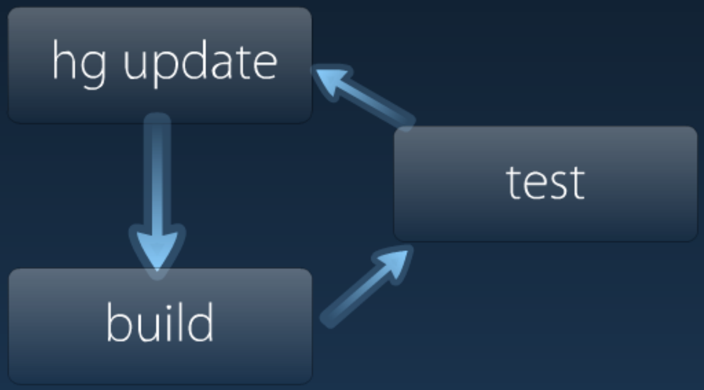
2. Bug correlation

- search issues for commit references
- search commits for issue references

3. Bug injection identification
- use VCS annotation
- use test cases and VCS bisection

```
--- a/Lib/test/test_ssl.py
+++ b/Lib/test/test_ssl.py
...
+   def test_parse_cert_CVE_2013_4073(self):
+       p = ssl._ssl._test_decode_cert(NULLBYTECERT)
+       if support.verbose:
+           sys.stdout.write("\n" + pprint.pformat(p) + "\n")
+       subject = ((('countryName', 'US'),),
+                  (('stateOrProvinceName', 'Oregon'),),
```

```
$ hg blame ssl.py
...
65179: def match_hostname(cert, hostname):
...
65179:     if not cert:
87688:         raise ValueError("empty or no certificate, match_hostname needs a "
87688:                          "SSL socket or SSL context with either "
87688:                          "CERT_OPTIONAL or CERT_REQUIRED")
65179:     dnsnames = []
65179:     san = cert.get('subjectAltName', ())
65179:     for key, value in san:
65179:         if key == 'DNS':
86676:             if _dnsname_match(value, hostname):
65179:                 return
65179:             dnsnames.append(value)
...
```



Prezi

4. Find new bugs
- find similar code via text match

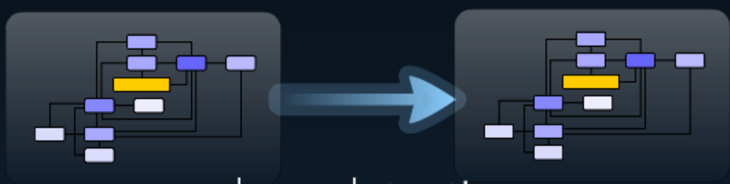
th(value, hostname):

(value)

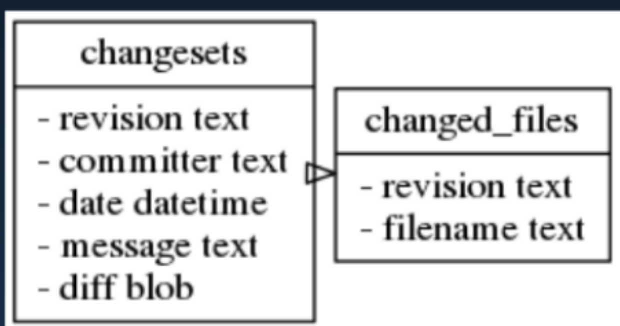
build

4. Find new bugs

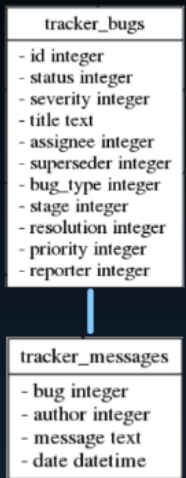
- find similar code via text matching
- find similar commits via machine learning



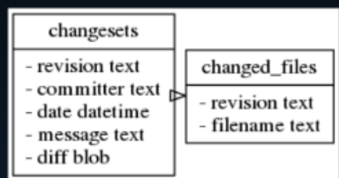
clone detection

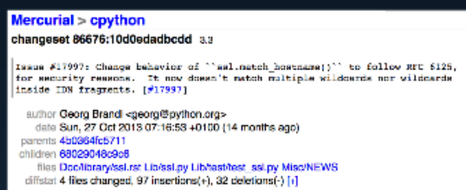


fication



1. Data gathering and organization

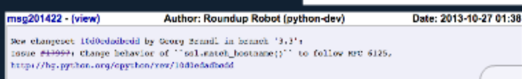



Mercurial > cpython

changeset 86676:10d0edadbcdd 3.3

Issue #17997: Change behavior of ``ssl.match_hostname()`` to follow RFC 6125, for security reasons. It now doesn't match multiple wildcards nor wildcards inside IDN fragments. [#17997]

author Georg Brandl <georg@python.org>
date Sun, 27 Oct 2013 07:16:53 +0100 (14 months ago)
parents 4b0364fc5711
children 68029048c9c6
files Doc/library/ssl.rst Lib/ssl.py Lib/test/test_ssl.py Misc/NEWS
diffstat 4 files changed, 97 insertions(+), 32 deletions(-) [+]


msg201422 - (view) Author: Roundup Robot (python-dev) Date: 2013-10-27 01:38

New changeset 10d0edadbcdd by Georg Brandl in branch '3.3':
Issue #17997: Change behavior of ``ssl.match_hostname()`` to follow RFC 6125,
http://hg.python.org/cpython/rev/10d0edadbcdd

2. Bug correlation

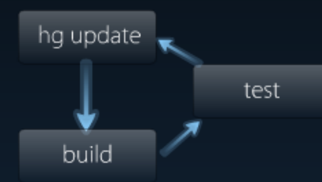
- search issues for commit references
- search commits for issue references

3. Bug injection identification

- use VCS annotation
- use test cases and VCS bisection

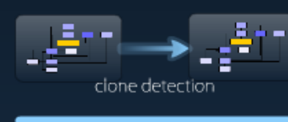
```
$ hg blame ssl.py
...
65179: def match_hostname(cert, hostname):
...
65179:     if not cert:
87688:         raise ValueError("empty or no certificate, match_hostname needs a "
87688:                          "SSL socket or SSL context with either "
87688:                          "CERT_OPTIONAL or CERT_REQUIRED")
65179:     dnsnames = []
65179:     san = cert.get('subjectAltName', ())
65179:     for key, value in san:
65179:         if key == 'DNS':
86676:             if _dnsname_match(value, hostname):
65179:                 return
65179:             dnsnames.append(value)
...
```

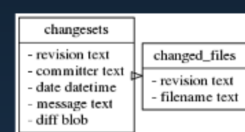
```
--- a/Lib/test/test_ssl.py
+++ b/Lib/test/test_ssl.py
...
+    def test_parse_cert_CVE_2013_4073(self):
+        p = ssl._ssl._test_decode_cert(NULLBYTECERT)
+        if support.verbose:
+            sys.stdout.write("\n" + pprint.pformat(p) + "\n")
+        subject = ((('countryName', 'US'),),
+                   (('stateOrProvinceName', 'Oregon'),),
```



4. Find new bugs

- find similar code via text matching
- find similar commits via machine learning





5. Manual verification

Results

Test Sets

python™

- Large codebase
- Mostly C and Python
- Routine Coverity scans

93,000+ commits
192 committers
~1M lines of code

34,000 issues
162 security issues

Roundup

- Order of magnitude smaller
- Mostly Python

<5,000 commits
~100K lines of code

22 security issues

Python:
- DoS in reference WSGI implementation
- Information disclosure via pydoc server

Roundup:
- 2x Unsafe Python functions
- 2 others were mitigated directory traversal vulnerabilities

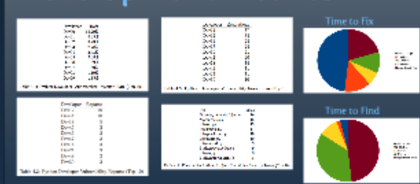
```
[*] issue10714{closed|security|msg:4|related:3}
[ ] self.raw_requestline = self.rfile.readline()
cpython/Lib/wsgiref/simple_server.py: self.raw_requestline = self.rfile.readline()

[*] issue672656{closed|security|msg:5|related:2}
[ ] self.address = ('', port)
cpython/Lib/pydoc.py: self.address = ('', port)
```

Validation

- Can we find 0days?
- Is the false positive rate reasonable?
  - Coverity developers say <30%
- Is the false negative rate reasonable?

Project	LOC (Approx.)	Total Issues	Code Signatures	Candidates	True Positives
Python	[illegible]	162	[illegible]	21	2
Roundup	[illegible]	22	5	4	2

Development Metrics



Conclusions

- Novel approach for identifying vulnerability injection
- Good at finding new bugs that are similar to bugs that have already been fixed (and can sometimes reuse old fixes)
- Can still find bugs in projects with insufficient development metadata
- Almost entirely automated

Prezi

Results

Test Sets

python™

- Large codebase
- Mostly C and Python
- Routine Coverity scans

93,000+ commits
192 committers
~1M lines of code

34,000 issues
162 security issues

Roundup

- Order of magnitude smaller
- Mostly Python

<5,000 commits
~100K lines of code

22 security issues

Python:
...S in reference WSGI implementation
...formation disclosure via pydoc server


```
[*] issue10714{closed|security|msg:4|related:3}
[ ] self.raw_requestline = self.rfile.readline()
cpython/Lib/wsgiref/simple_server.py: self.raw_requestline = self.rf

[*] issue672656{closed|security|msg:5|related:2}
```

python™

- Large codebase
- Mostly C and Python
- Routine Coverity scans

93,000+ commits
192 committers
~1M lines of code

34,000 issues
162 security issues

Roundup

- Order of magnitude smaller
- Mostly Python

<5,000 commits
~100K lines of code

22 security issues

Python:

- DoS in reference WSGI implementation
- Information disclosure via pydoc server

Roundup:

- 2x Unsafe Python functions
- 2 others were mitigated directory traversal vulnerabilities

Validation

```
[*] issue10714{closed|security|msg:4|related:3}
[ ] self.raw_requestline = self.rfile.readline()
cpython/Lib/wsgiref/simple_server.py: self.raw_requestline = self.rfile.readline()

[*] issue672656{closed|security|msg:5|related:2}
[ ] self.address = ('', port)
cpython/Lib/pydoc.py: self.address = ('', port)
```

elopment Metrics

Validation

- Can we find 0days?
- Is the false positive rate reasonable?
  - Coverity developers say <30%
- Is the false negative rate reasonable?

Project	LOC (Approx.)	Total Issues	Code Signatures	Candidates	True Positives
Python	976,413	162	38	21	2
Roundup	88,578	22	6	4	2

Table 4.5: Vulnerability Detection Rates

Development Metrics

Developer	Rate
Dev00	14.29%
Dev01	4.35%
Dev02	3.23%
Dev03	2.78%
Dev04	2.53%
Dev05	2.08%
Dev06	1.59%
Dev07	1.59%
Dev08	1.19%
Dev09	1.15%

Table 4.1: Python Developer Vulnerability Injection Rate (Top 10)

Developer	Reports
Dev12	18
Dev44	10
Dev18	5
Dev45	3
Dev46	3
Dev47	3
Dev14	3
Dev24	3
Dev48	2
Dev30	2

Table 4.2: Python Developer Vulnerability Reports (Top 10)

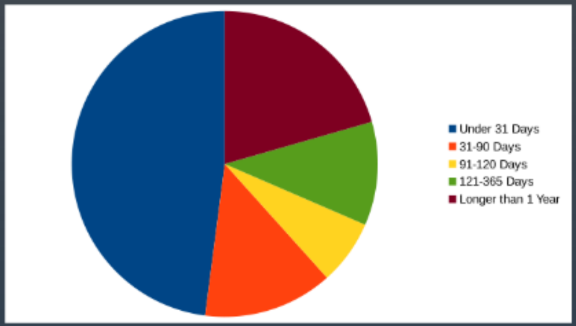
Developer	Resolutions
Dev14	37
Dev12	34
Dev31	24
Dev37	24
Dev36	21
Dev42	20
Dev18	14
Dev43	11
Dev24	11
Dev30	10

Table 4.3: Python Developer Vulnerability Resolutions (Top 10)

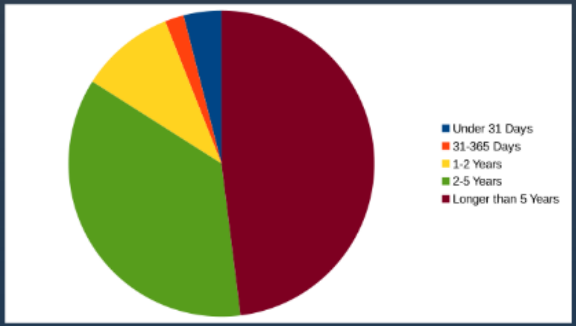
File	Issues
Objects/unicodeobject.c	16
Modules/_ssl.c	15
Lib/ssl.py	13
Lib/zipfile.py	11
Lib/platform.py	10
Lib/netrc.py	10
Lib/smtpd.py	9
Python/sysmodule.c	9
Lib/os.py	8
Python/mysnprintf.c	8

Table 4.4: Files in the Python Project Containing Security Issues (Top 10)

Time to Fix



Time to Find



Developer	Rate
Dev00	14.29%
Dev01	4.35%
Dev02	3.23%
Dev03	2.78%
Dev04	2.53%
Dev05	2.08%
Dev06	1.59%
Dev07	1.59%
Dev08	1.19%
Dev09	1.15%

Table 4.1: Python Developer Vulnerability Injection Rate (Top 10)

Developer Reports

Table 4.1: Python Developer Vulnerability Injection Rate (Top 10)

Developer	Reports
Dev12	18
Dev44	10
Dev18	5
Dev45	3
Dev46	3
Dev47	3
Dev14	3
Dev24	3
Dev48	2
Dev30	2

Table 4.2: Python Developer Vulnerability Reports (Top 10)

Table 4.3: Python Developer Vulnerability Resolutions (Top 10)

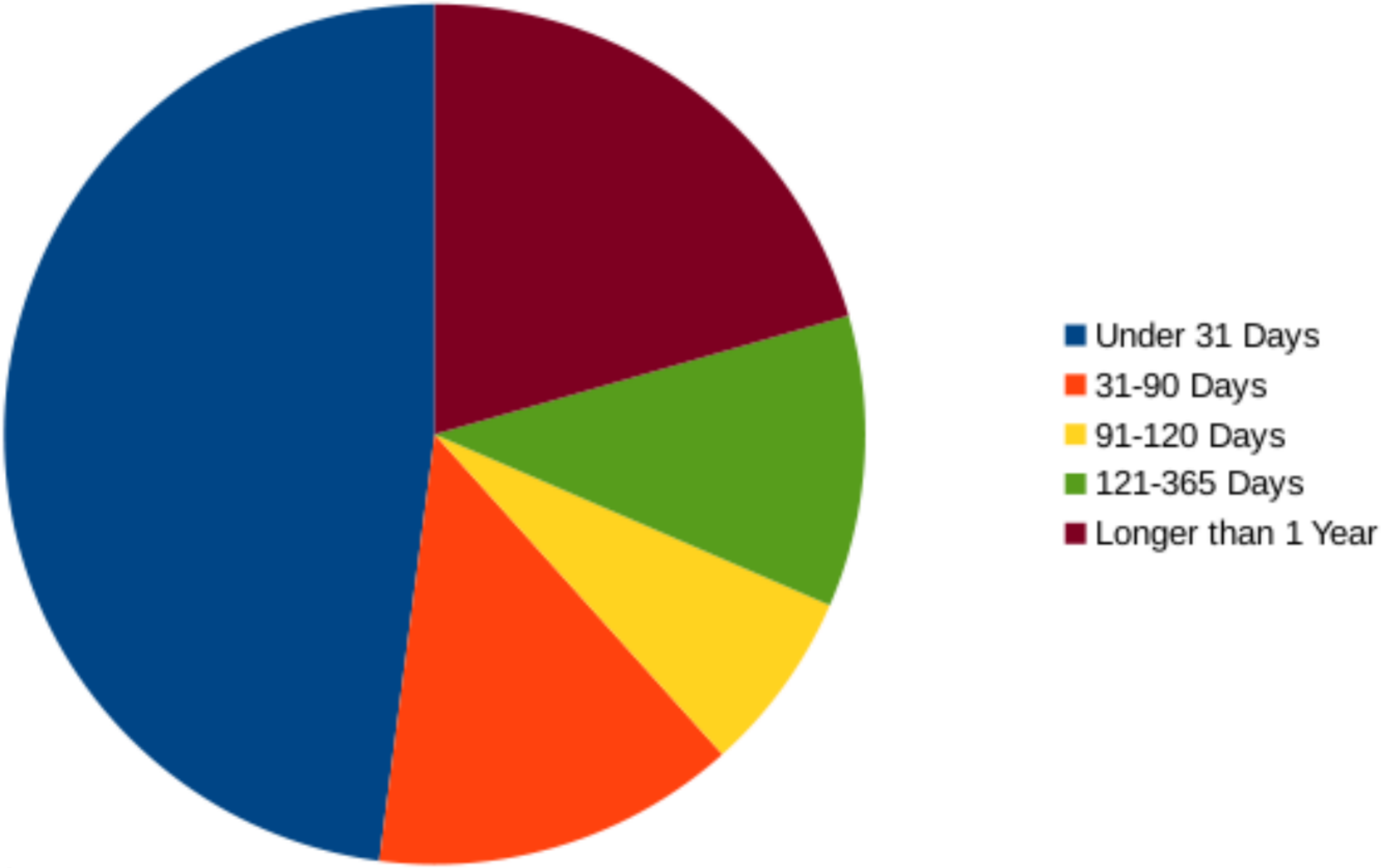
File	Issues
Objects/unicodeobject.c	16
Modules/_ssl.c	15
Lib/ssl.py	13
Lib/zipfile.py	11
Lib/platform.py	10
Lib/netrc.py	10
Lib/smtpd.py	9
Python/sysmodule.c	9
Lib/os.py	8
Python/mysnprintf.c	8

Table 4.4: Files in the Python Project Containing Security Issues (Top 10)

Developer	Resolutions
Dev14	37
Dev12	34
Dev31	24
Dev37	24
Dev36	21
Dev42	20
Dev18	14
Dev43	11
Dev24	11
Dev30	10

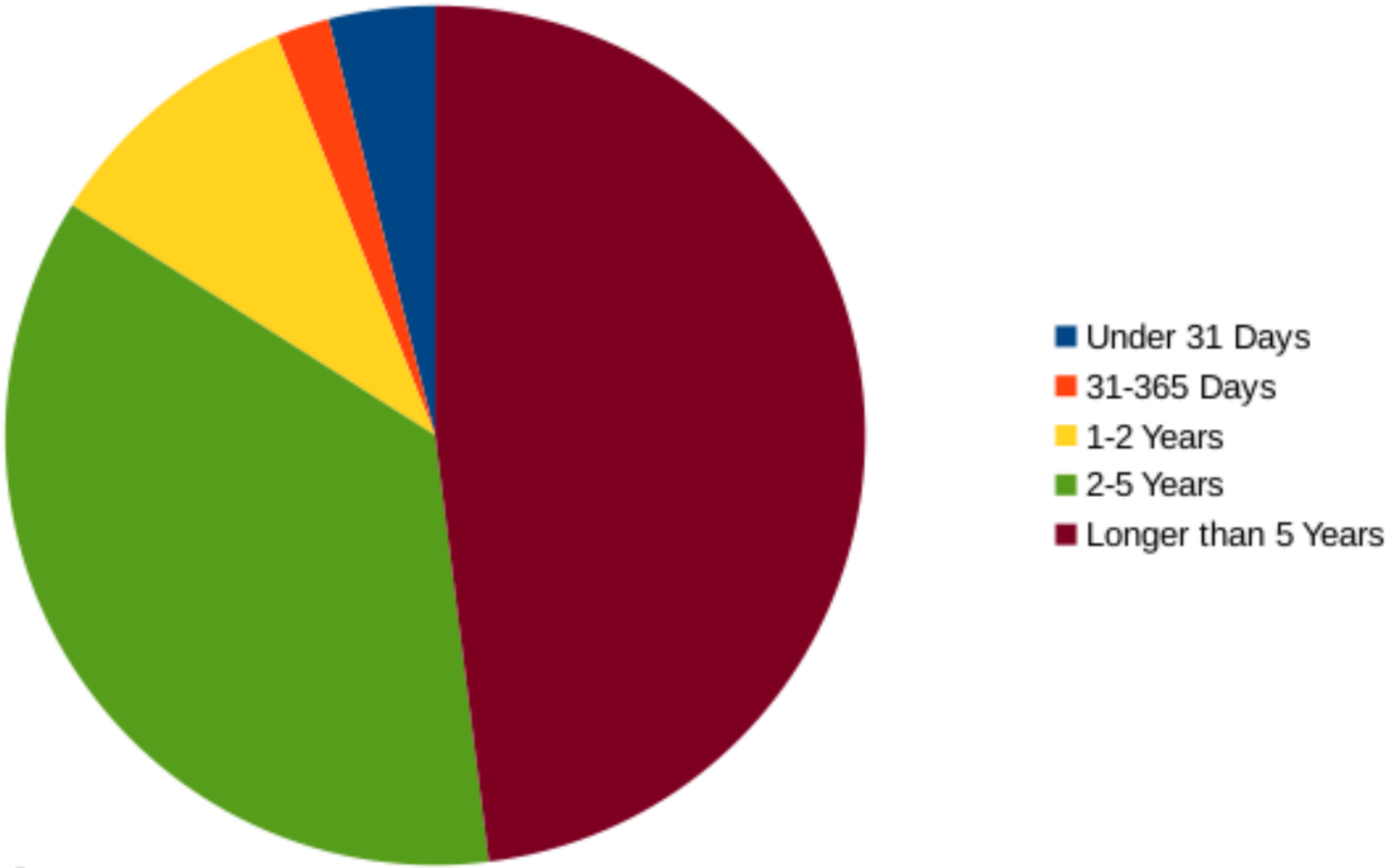
Table 4.3: Python Developer Vulnerability Resolutions (Top 10)

Time to Fix



Under 31 Days
31-90 Days
91-120 Days
121-365 Days
Longer than 1 Year

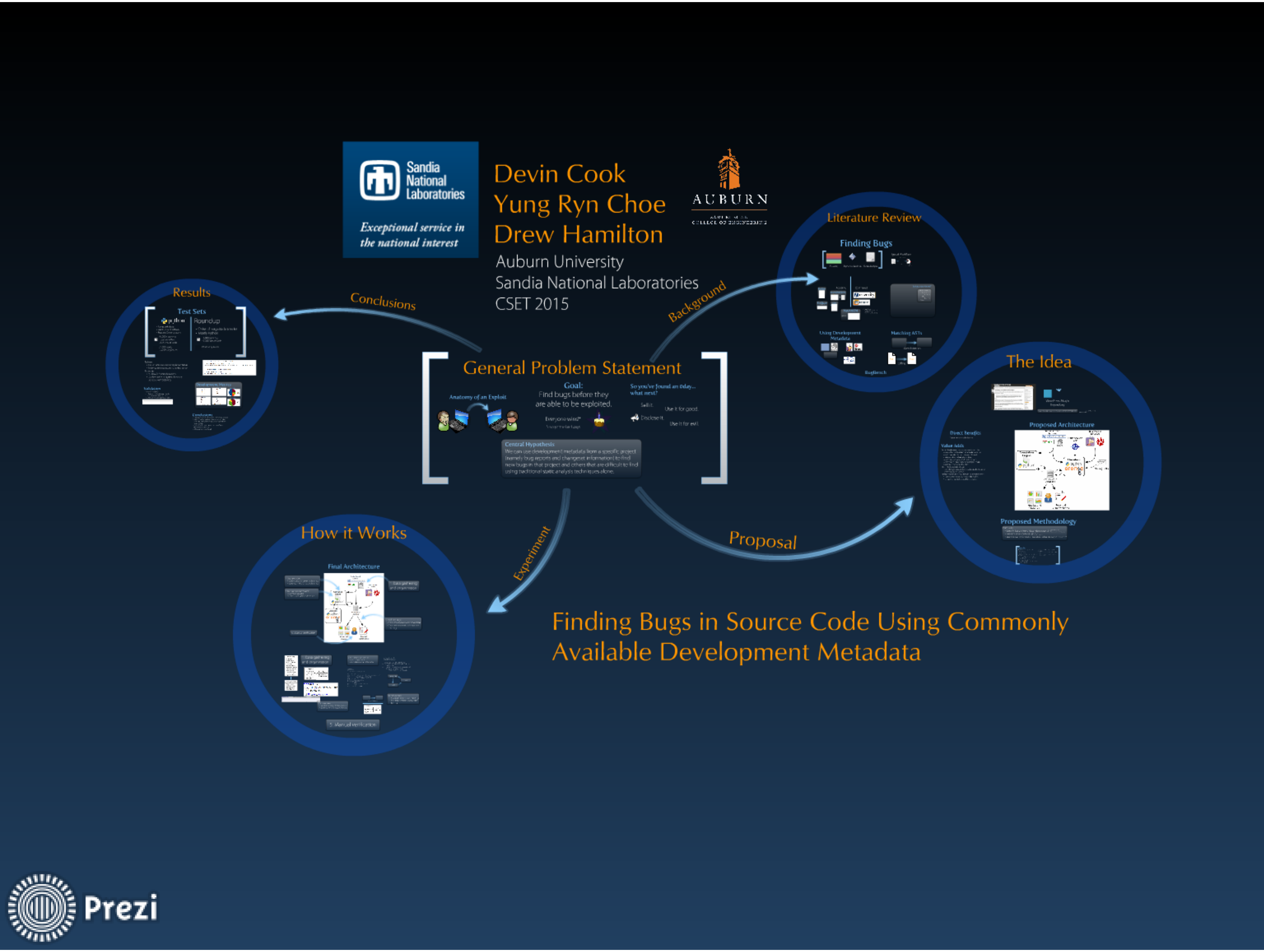
Time to Find



Prezi

Conclusions

- Novel approach for identifying vulnerability injection
- Good at finding new bugs that are similar to bugs that have already been fixed (and can sometimes reuse old fixes)
- Can still find bugs in projects with insufficient development metadata
- Almost entirely automated


Finding Bugs in Source Code Using Commonly Available Development Metadata
Devin Cook
Yung Ryn Choe
Drew Hamilton
Auburn University
Sandia National Laboratories
CSET 2015
Sandia National Laboratories
Exceptional service in the national interest
AUBURN
General Problem Statement
Goal:
Find bugs before they are able to be exploited.
Anatomy of an Exploit
So you've found an 0day... what next?
Sell it.
Disclose it.
Use it for good.
Use it for evil.
Central Hypothesis
We can use development metadata from a specific project (namely bug reports and changeset information) to find new bugs in that project and others that are difficult to find using traditional static analysis techniques alone.
Background
Literature Review
Finding Bugs
Using Development Metadata
Matching ASTs
Proposal
The Idea
Proposed Architecture
Proposed Methodology
Direct Benefits
Value Adds
Experiment
How it Works
Final Architecture
5. Manual verification
Conclusions
Results
Test Sets
Roundup
Prezi