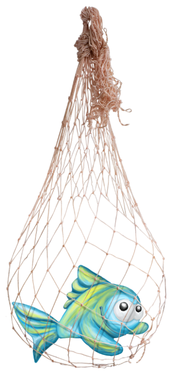
Detecting the 1%: Growing the Science of Vulnerability Discovery

Laurie Williams
laurie_williams@ncsu.edu

Real people – Real Projects – Real Impact

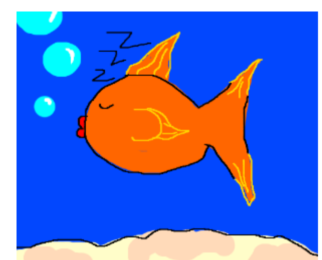
Meet the “fishy” vulnerability characters



Adam the Attack-prone

Edwin the Exploitable

David the Detected



Larry the Latent

The goal is to aid software practitioners in efficiently detecting exploitable vulnerabilities through empirical study of the characteristics of vulnerabilities and through the development of vulnerability prediction models.

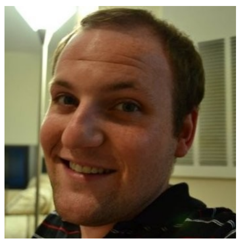
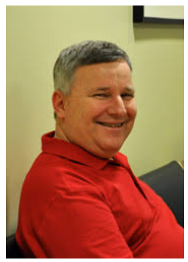
The goal is to aid software practitioners in efficiently detecting exploitable vulnerabilities through empirical study of the characteristics of vulnerabilities and through the development of vulnerability prediction models.

The goal is to aid software practitioners in efficiently detecting exploitable vulnerabilities through empirical study of the characteristics of vulnerabilities and through the development of vulnerability prediction models.

The goal is to aid software practitioners in efficiently detecting exploitable vulnerabilities through empirical study of the characteristics of vulnerabilities and through the development of vulnerability prediction models.

Collaborators



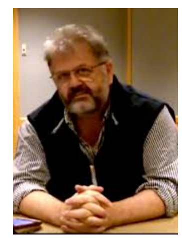











Funded by: NSF

In cooperation: Microsoft Research

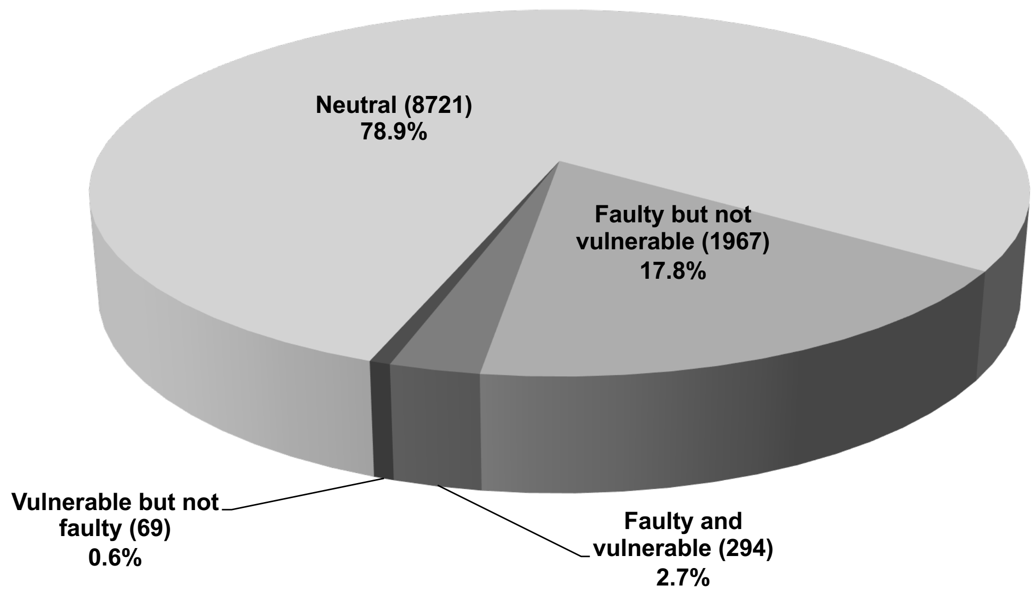
Where are we going?

- Setting the stage
- Complications in vulnerability research
- The real questions …
  - Where shall we look?
  - How shall we look?
  - Which vulnerabilities are likely to be exploited?
- Future directions

Design flaws and implementation bugs

Stage | Complications | Where | How | Exploited | Future

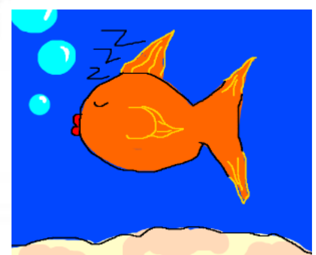
Vulnerabilities are rare events (Firefox 2.0)



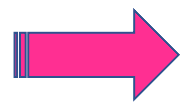
Getting, creating, and cleaning the data 😳

Stage | Complications | Where | How | Exploited | Future

Where shall we look?



Larry the Latent



David the Detected

Stage | Complications | **Where** | How | Exploited | Future

Unfiltered Static Analysis Alerts as Predictor

If a developer has such poor coding practices that he/she causes lots of (unfiltered) static analysis alerts, you should look carefully in that area for <u>other implementation bugs</u> and larger <u>design flaws</u>.

Stage | Complications | Where | How | Exploited | Future

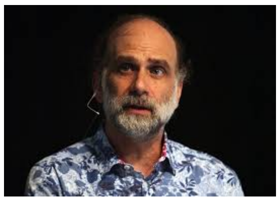
Correlations between static analysis alerts and vulnerability count

(all statistically significant)

Metric	Case study 1 (component-level)	Case study 2 (file-level)	Case study 3 (component-level)
All SA alerts	0.2	0.2	0.2
Security SA alerts	0.2	0.2	0.2

Stage | Complications | Where | How | Exploited | Future

Complexity as Predictor



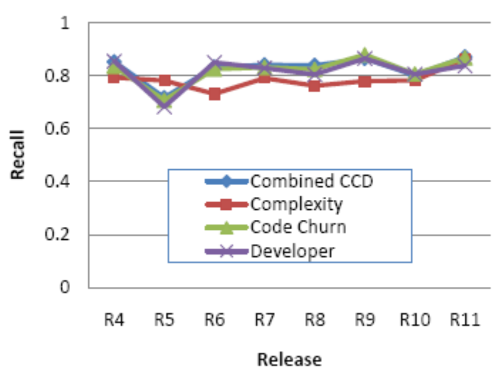
Security experts say:

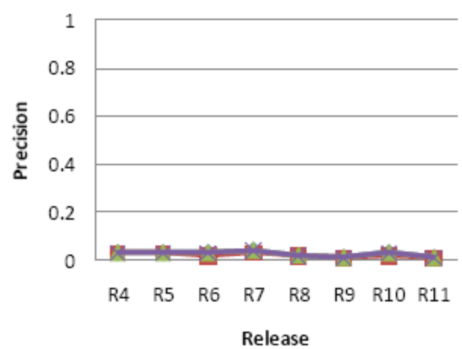
- Bruce Schneier
  - “Complexity is the worst enemy of security.”
- Dan Geer
  - “Complexity provides both opportunity and hiding places for attackers.”
- Gary McGraw
  - “A … trend impacting software security is unbridled growth in … complexity …”

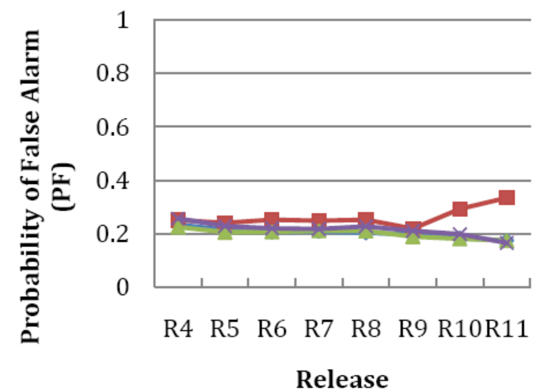
Complexity and Other Metrics

- 14 code complexity metrics
  - Lines of code, cyclomatic complexity, fan-in/fan-out, coupling, comment density and others
- 3 code churn metrics
  - Frequency of file changes, lines of code changed, and new lines of code
- 11 developer metrics
  - Number of developers and other network analysis-inspired metrics (e.g. betweenness, closeness)

Results: Predictability (11 releases Firefox)

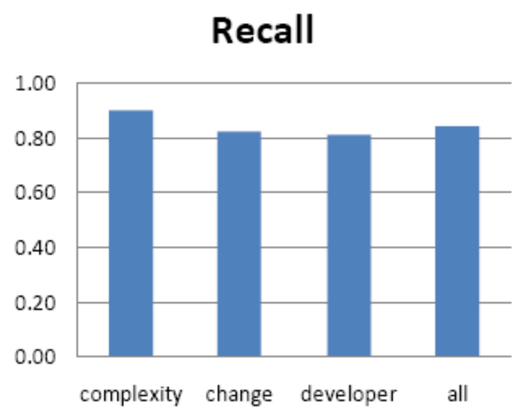


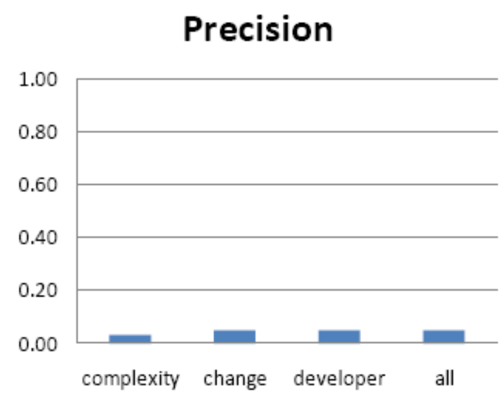


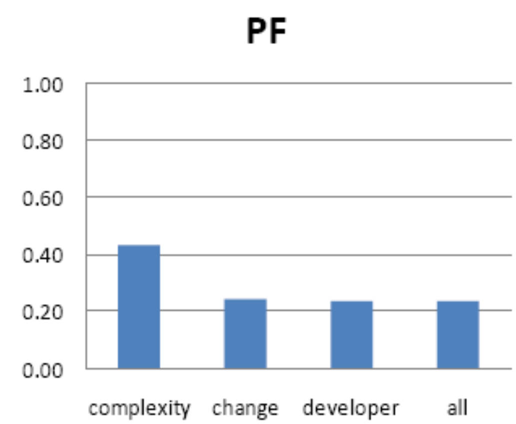


Stage | Complications | Where | How | Exploited | Future

Results: Predictability (RHEL)

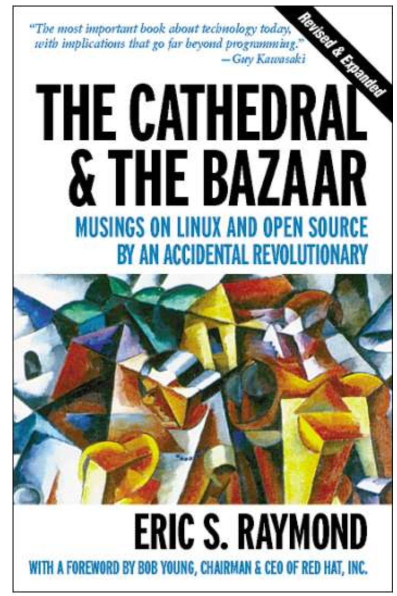






Developer Metrics as Predictor

> "*Given a large enough beta-tester and* ***co-developer*** *base, almost every problem will be characterized quickly and the fix* ***obvious to someone.*** *[…]*
> *Many eyes make all bugs shallow.*"
>
> *-Linus' Law*
> *Eric Raymond*



How Many Developers?

- Metric: NumDevs
  The number of distinct developers who changed a given source code file

In all three case studies…

Vulnerable files had more developers than neutral files ($p<0.001$)

*Files changed by **6 or more developers** were **4 times more likely** to have a vulnerability, ($p<0.001$)*

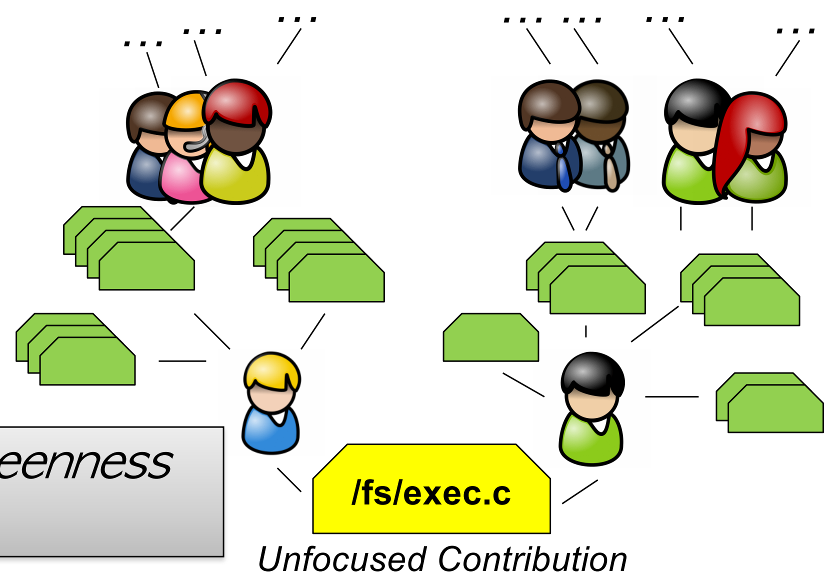
(…not quite what Linus' Law says…)

Unfocused Contributions

Examined files changed by **many** developers who were **working on many other files** at the time (an "*unfocused contribution*")

Used contribution network centrality (**CNBetweenness**)

Vulnerable files had a higher CNBetweenness ($p<0.001$) than neutral files.



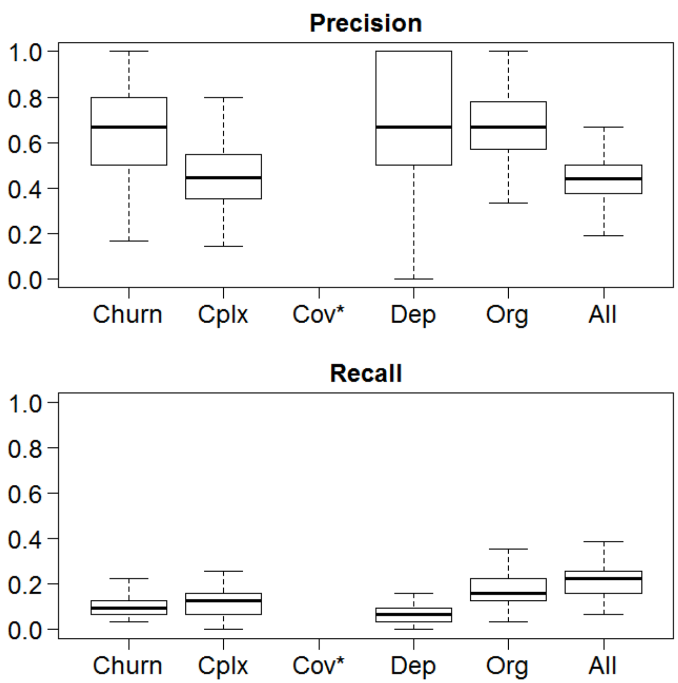
Stage | Complications | Where | How | Exploited | Future

Traditional Code Metrics as Predictor

Metric	rho
Edit Frequency (EF)	0.292
Total Lines of Code	0.281
Frequency	0.279
Total Complexity	0.276
Repeat Frequency	0.273
Number of Ex-Engineers (NOEE)	0.270
TotalFanIn	0.263
TotalFanOut	0.262
Number of Engineers (NOE)	0.261
Total Global Variables	0.255
Total Churn	0.254
Max FanIn	0.224
Max Complexity	0.207
Max FanOut	0.196
Max Lines of Code	0.194
Outgoing direct	0.168
Total ClassMethods	0.167
Max ClassMethods	0.164
Total InheritanceDepth	0.161
Total BlockCoverage	0.157
Incoming direct	0.156
Tota ClassCoupling	0.154
Total ArcCoverage	0.152
Incoming closure	0.148
Total SubClasses	0.141
Max InheritanceDepth	0.137
Max ClassCoupling	0.137
Max SubClasses	0.124
Level of Org. Code Ownership (OCO)	0.123
Depth of Master Ownership (DMO):	0.101

All correlations values are significant at p<0.0001.

Windows Vista



What you look at will likely be a vulnerability …

… But many vulnerabilities will be missing.

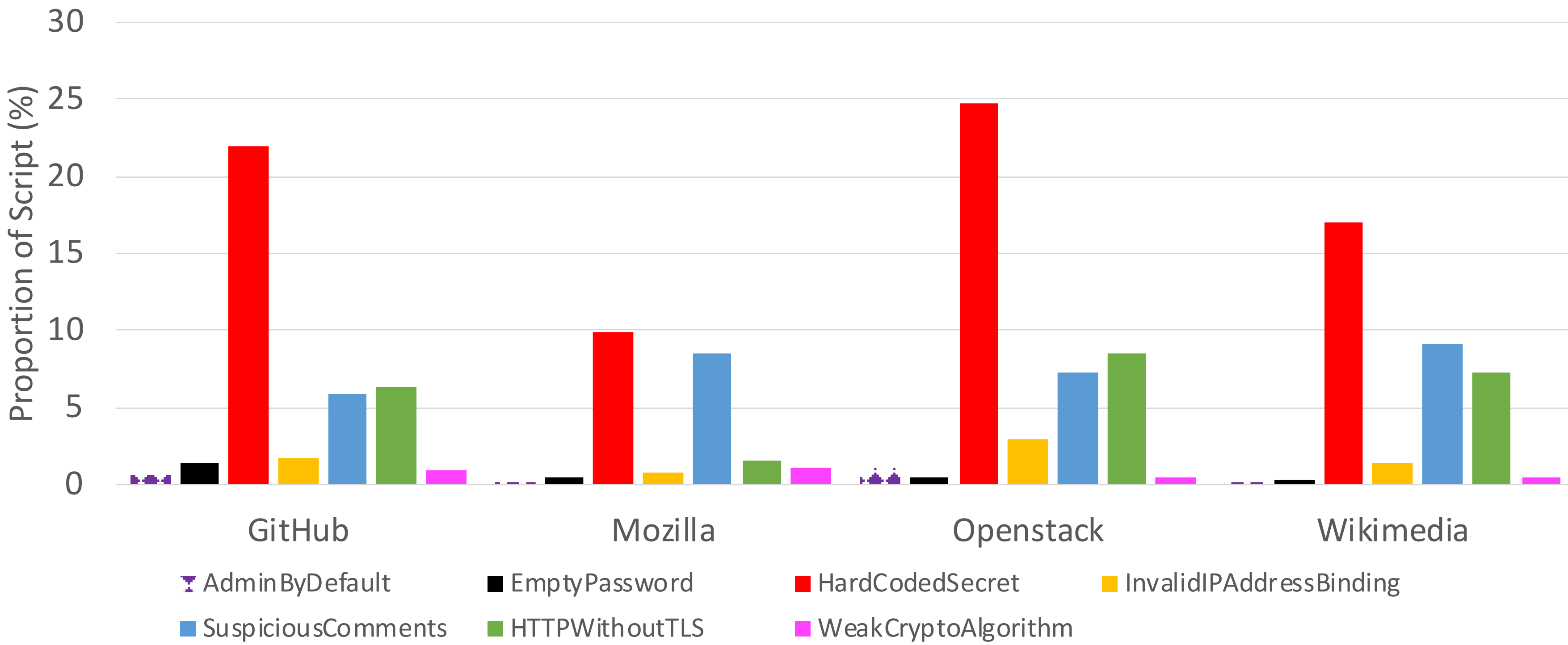
Vulnerability prediction modeling by others

- Without much better results when tested with similar vulnerability scarcity:
  - Dependency structure
  - Text mining
  - Design churn
  - More code metrics
  - Neural networks and deep learners

Infrastructure as Code Security Smells

`$power_username='admin'`	Admin by default
`password=>''`	Empty password
`$power_password='admin'`	Hard-coded secret
`$bind_host='0.0.0.0'`	Invalid IP address binding
`#FIXME(bogdando) remove these hacks after switched to systemd service.units`	Suspicious comment
`$quantum_auth_url = 'http://127.0.0.1:35357/v2.0'`	Use of HTTP without TLS
`password => ht_md5($power_password)`	Use of weak cryptography algorithm

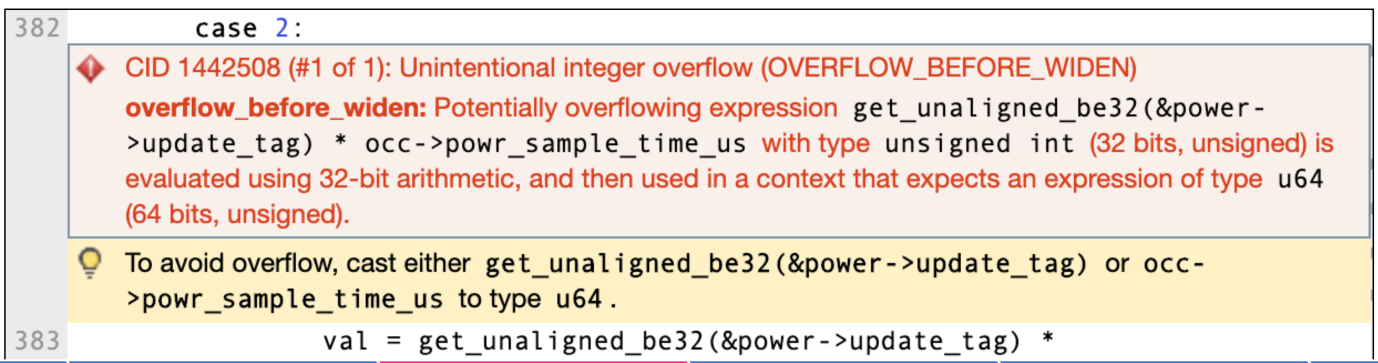
Frequency of Security Smells



Actionable and/or Predictive Heuristics

- Static Analysis Alerts
  - Predictive: Static analysis alerts are indicative of all security vulnerabilities.
  - No pre-processing to determine true positive necessary.

- Code complexity
  - Actionable and predictive: Complex code is less secure



Actionable and/or Predictive Heuristics - 2

- Developer activity metrics
  - Actionable and predictive
    - Don't allow too many people to change same (critical) file
    - Watch for the "hummingbirds" that change many files.

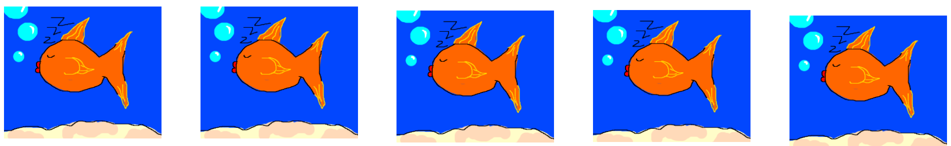
- Traditional code metrics
  - Predictive: Traditional code metrics can be used to find vulnerabilities
  - Support that vulnerabilities have the same characteristics as faults

- Infrastructure as code smells
  - Actionable: Identify and mitigate code smells

Takeaway

Vulnerability prediction models are not yet practical ... but patterns of what to watch for have been identified.

How shall we look?



NEED FOR SPEED™

Comparison of Vulnerability Discovery Techniques

Discovery Technique	Vulnerabilities Per Hour		
	Tolven eCHR	OpenEMR	PatientOS
Exploratory Manual Penetration Testing	0.00	0.40	.07
Systematic Manual Penetration Testing	0.94	0.55	0.55
Automated Penetration Testing	22.00	71.00	N/A
Static Analysis	2.78	32.40	11.15

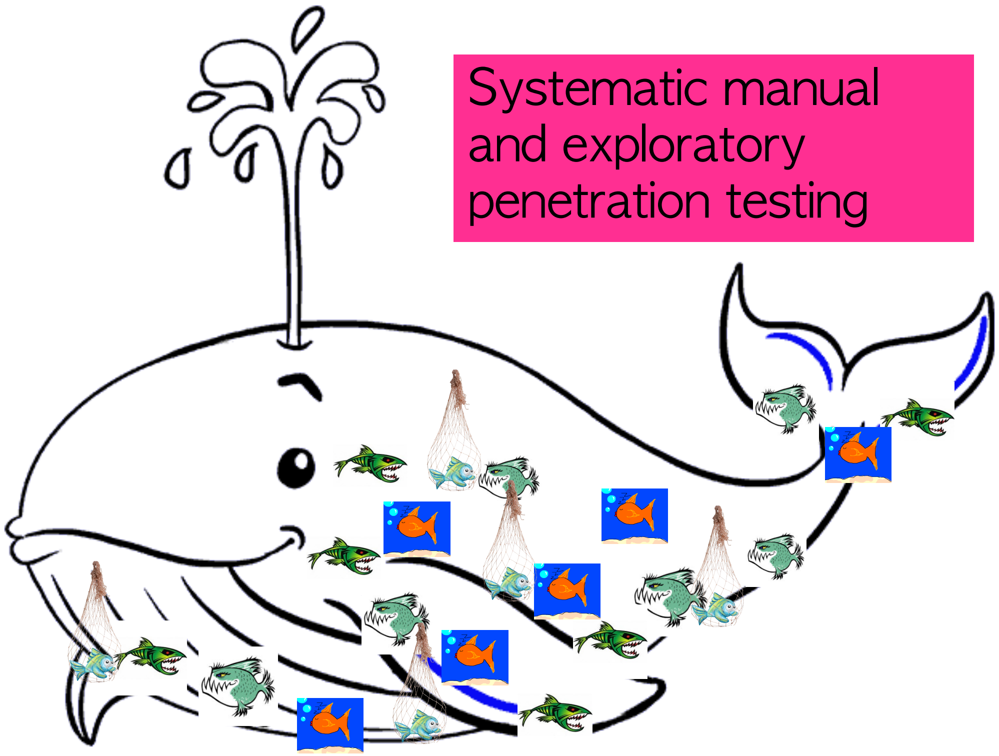
Other observations

No single technique discovered every type of vulnerability.

Very few individual vulnerabilities discovered with multiple discovery techniques.

Which technique?

Systematic manual and exploratory penetration testing



Design flaw

Automated penetration testing and static analysis

Implementation bug

Stage | Complications | Where | How | Exploited | Future

Takeaway

One technique is not enough.

What will be <u>exploited</u>?

Edwin the
Exploitable

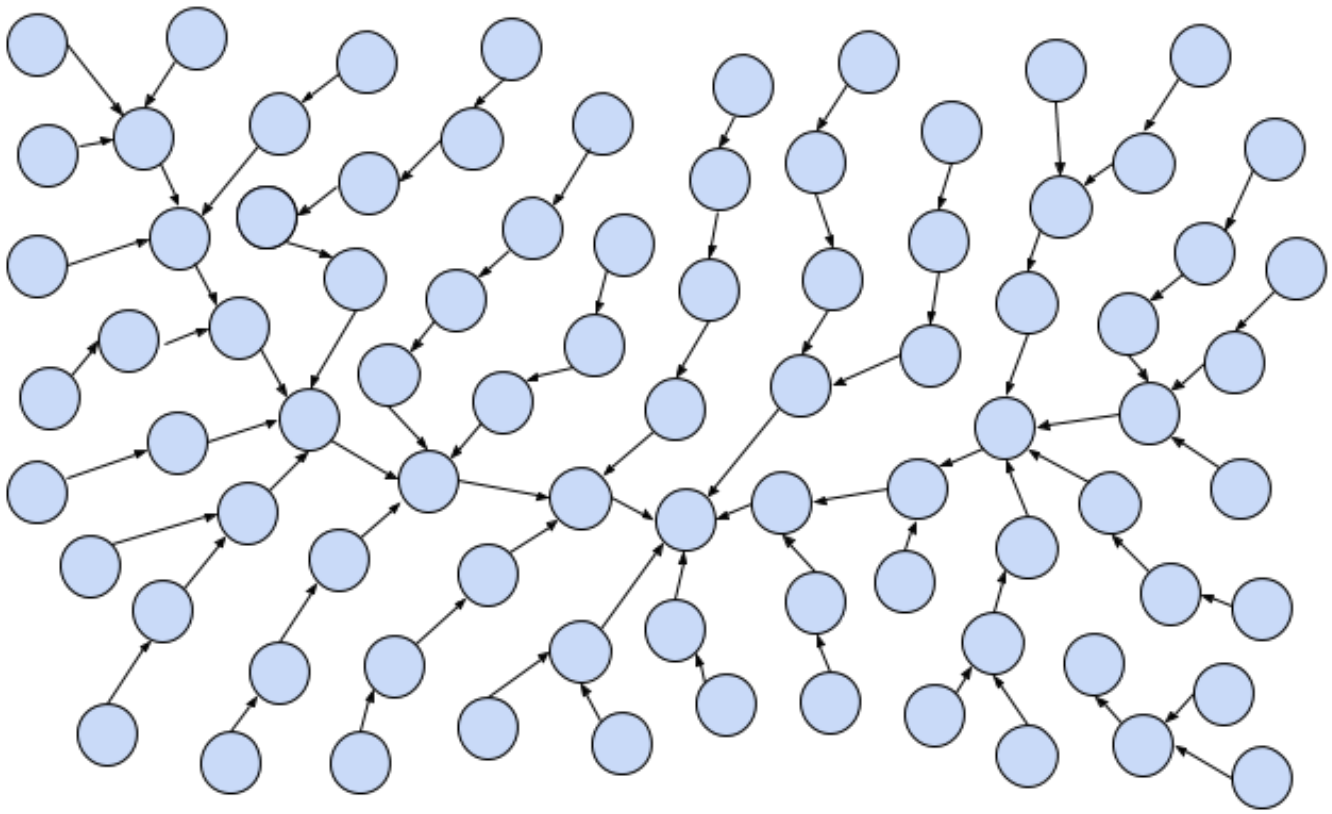
Adam the
Attack-prone

Stage | Complications | Where | How | Exploited | Future

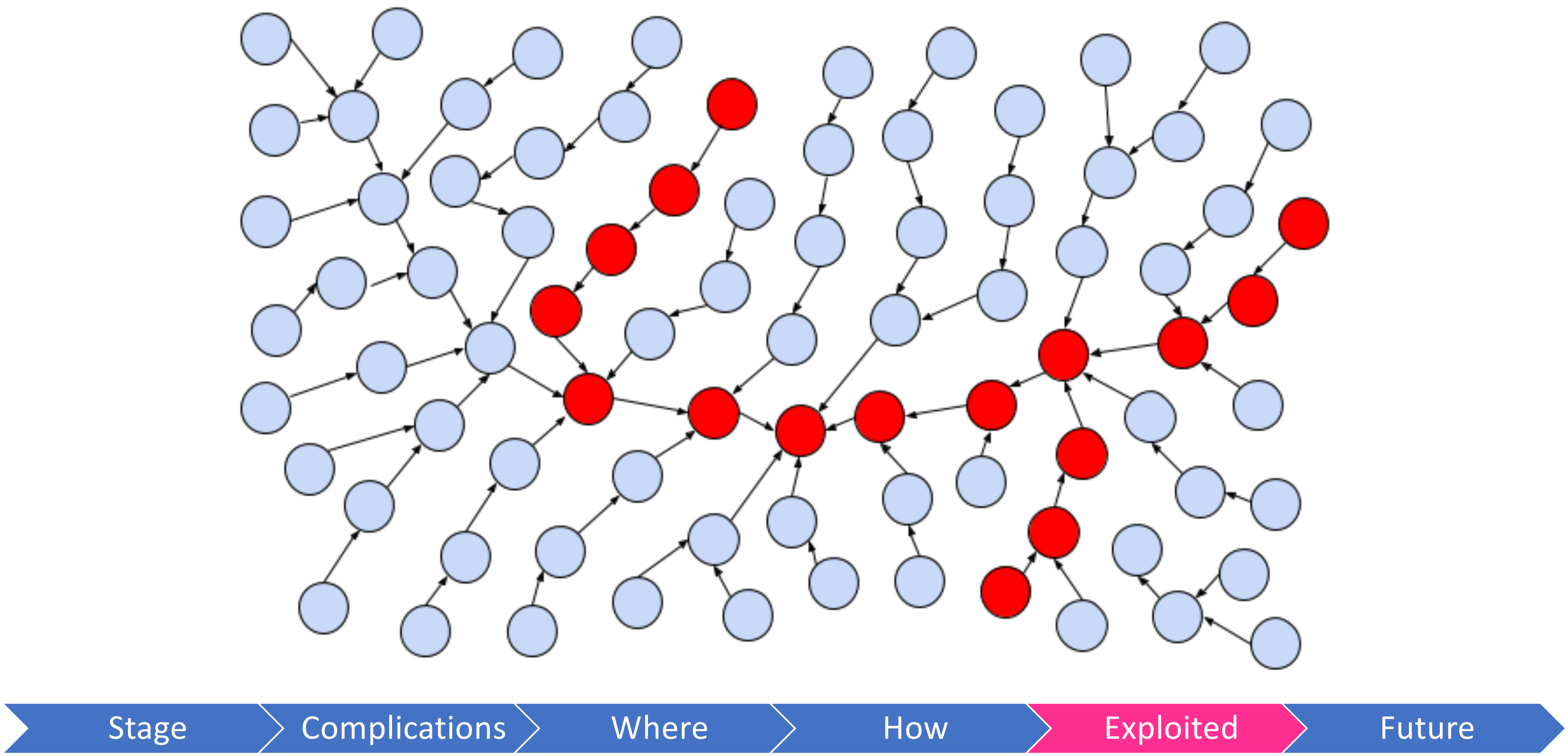
Risk-based Attack Surface Approximation

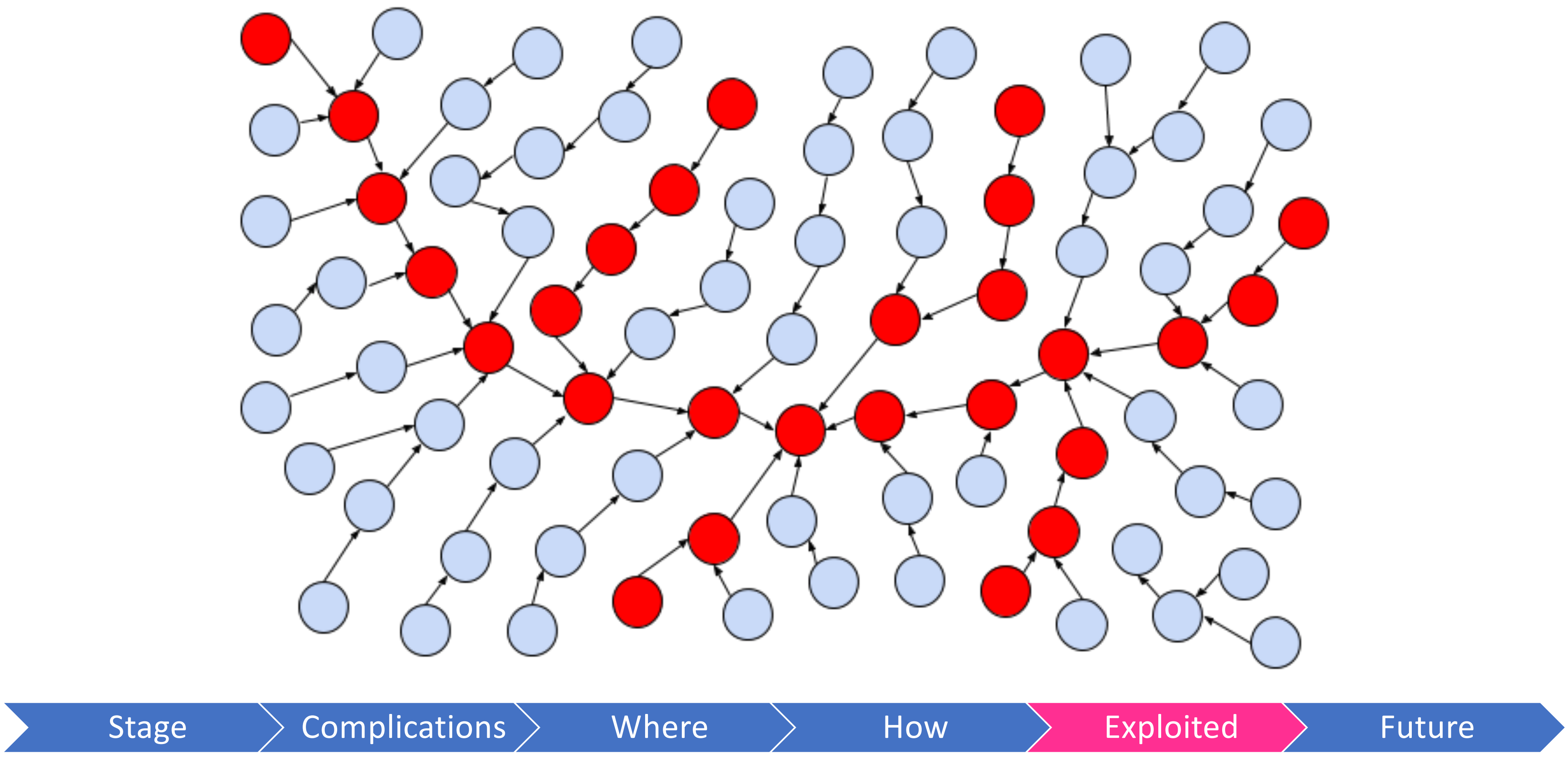
Code artifacts that appear in crash dump stack traces from a software system are more likely to have exploitable vulnerabilities than code artifacts that do not appear in crash dump stack traces.

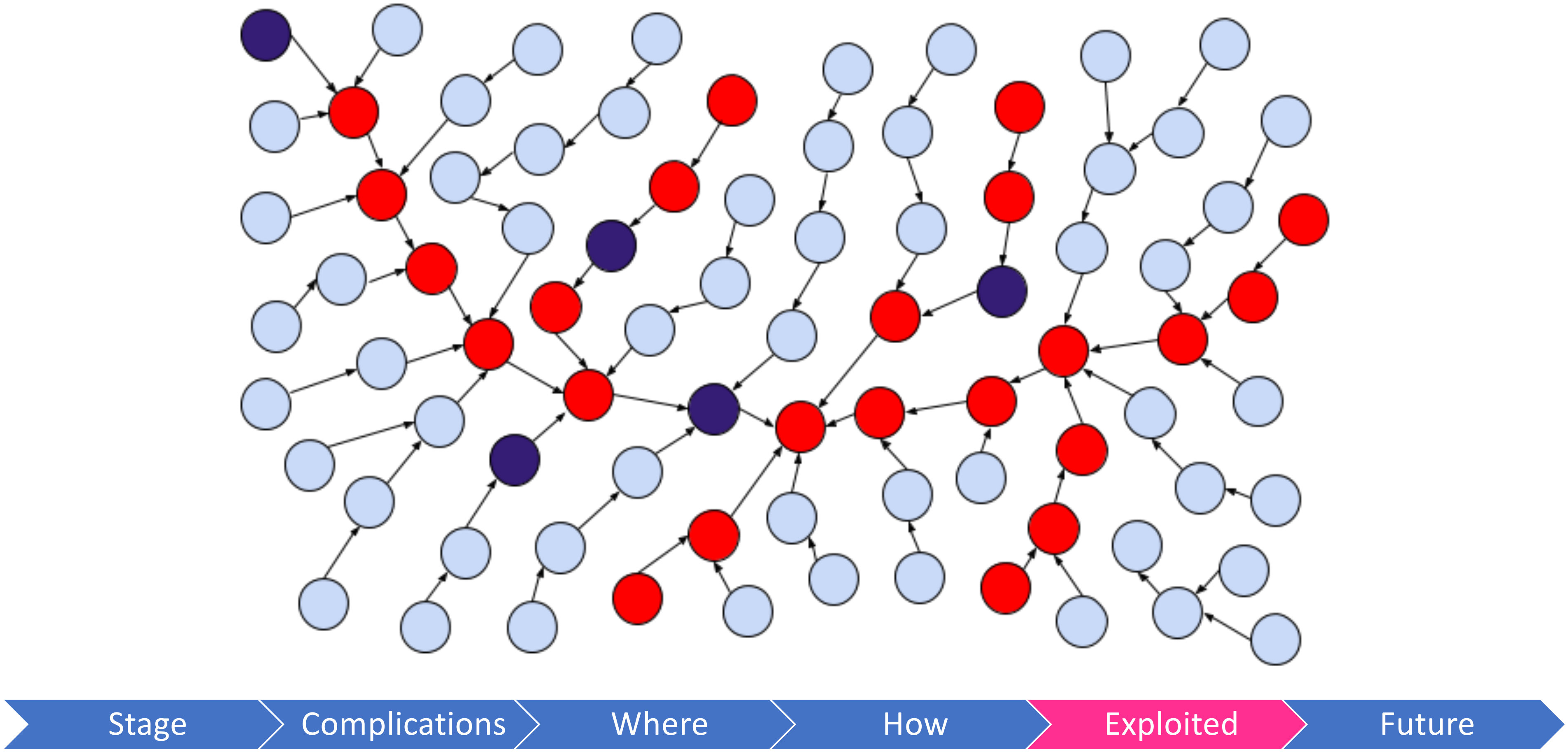
Stage | Complications | Where | How | **Exploited** | Future



Stage | Complications | Where | How | Exploited | Future


Stage
Complications
Where
How
Exploited
Future


Stage
Complications
Where
How
Exploited
Future


Stage
Complications
Where
How
Exploited
Future

Where the Exploitable Vulnerabilities Lie

	Code Coverage	Vulnerability Coverage
Windows (Binaries)	48.4%	94.8%
Firefox (Source Code Files)	14.8%	85.6%
Fedora (Packages)	8.9%	63.3%

Clustering on the Boundary?

Boundary Code (BC): percentage of code that appears on the boundary of a software system

Boundary Vulnerabilities (BV): percentage of vulnerabilities on Boundary Code (BC)

		BC	BV	Ratio
Windows 8	2014	4.5%	17.2%	3.8
	2015	4.6%	18.6%	4.0
Windows 8.1	2014	4.6%	16.5%	3.6
	2015	6.9%	23.7%	3.4
Windows 10	2014	3.4%	10.5%	3.1
	2015	3.9%	25.1%	6.4

Takeaway

Vulnerabilities found on the attack surface are exploitable. More work need to characterize exploitable and attack-prone vulnerabilities.

The goal is to aid software practitioners in efficiently detecting exploitable vulnerabilities through empirical study of the characteristics of vulnerabilities and through the development of vulnerability prediction models.

Building Vulnerability Datasets

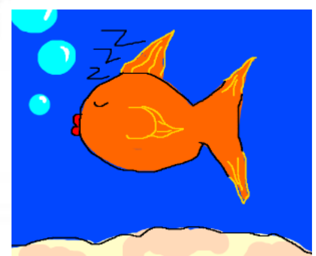
Stage → Complications → Where → How → Exploited → Future

Understanding the 1%

- Vulnerabilities versus non-security defects?
  - What technique was used to detect?
  - What was the role of the detector?
  - What is the complexity of the patch?
  - How much time elapsed from injection until detection?
  - How much time elapsed from the detection until the patch?
  - What patterns exist in the longitudinal arrival rate?
  - Can fault prediction models be used for vulnerabilities?

Stage | Complications | Where | How | Exploited | Future

<u>Where</u> shall we look?



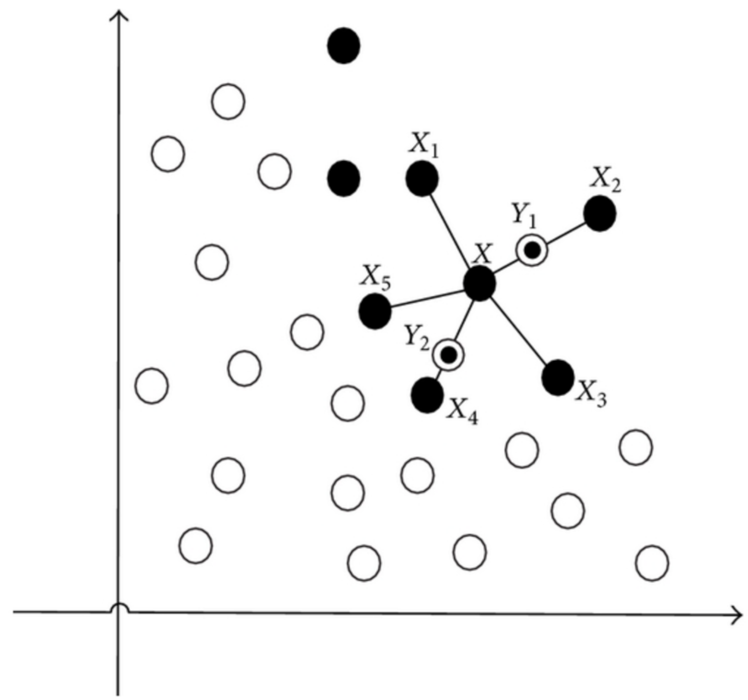
Larry the Latent

David the Discovered

Stage › Complications › **Where** › How › Exploited › Future

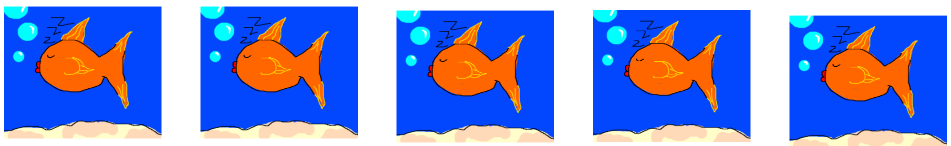
Training learners to recognize rare target

- SMOTE (Synthetic Minority Over-sampling)
- Fiddle the training data (but not the test data)
- Ignore the non-vulnerable files
- Synthesize more examples of the vulnerable files



Stage | Complications | Where | How | Exploited | Future

How shall we look?



NEED FOR SPEED

Stage › Complications › Where › How › Exploited › Future

Comparison of Vulnerability Discovery Techniques

Discovery Technique	Vulnerabilities Per Hour		
	OpenMRS	??	??
Exploratory Manual Penetration Testing			
Systematic Manual Penetration Testing			
Automated Penetration Testing			
Static Analysis			

What will be exploited?

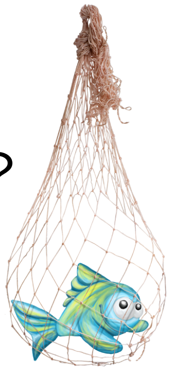
Edwin the Exploitable

Adam the Attack-prone

Stage > Complications > Where > How > Exploited > Future

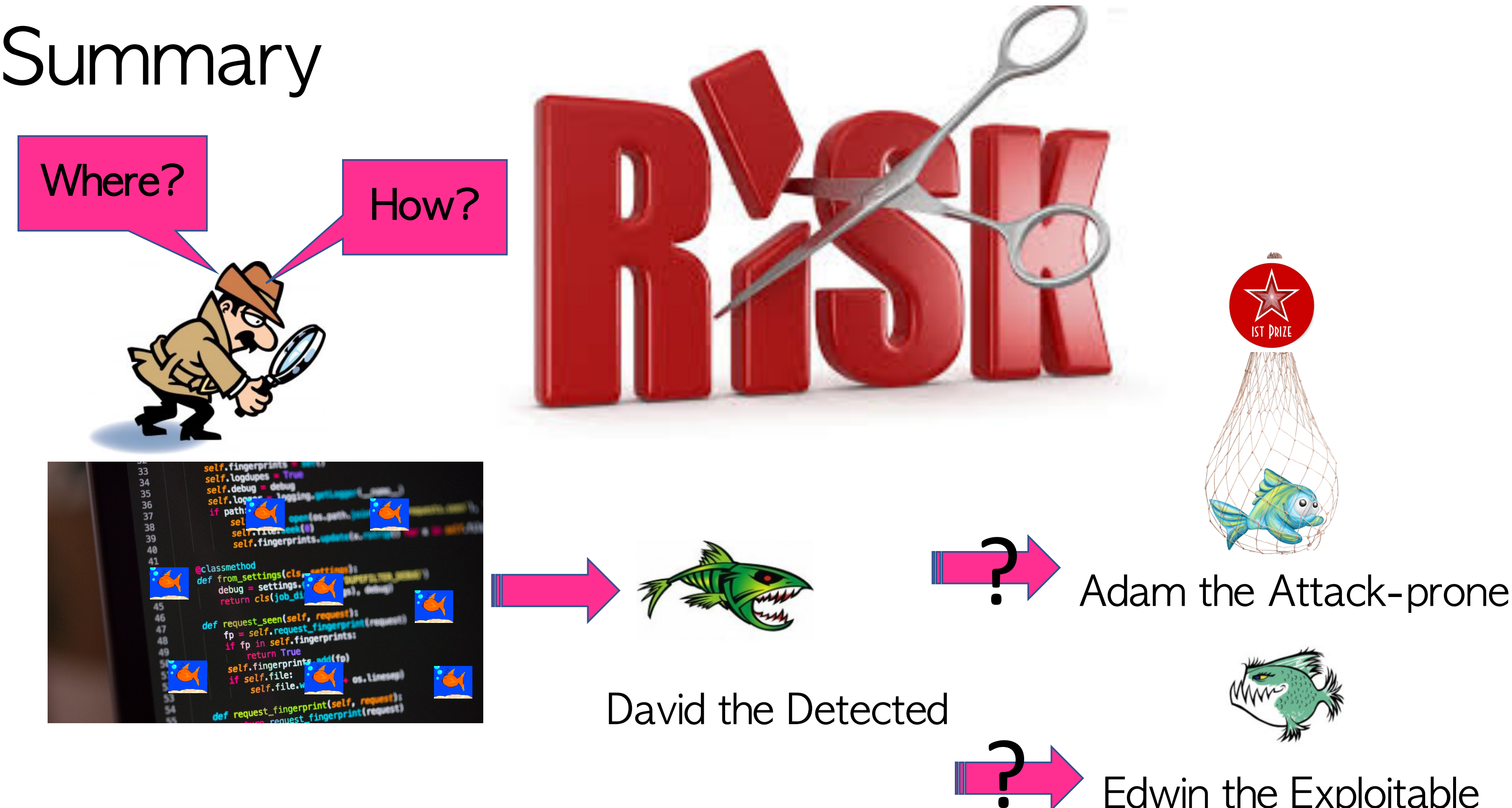
Characteristics of Exploitable Vulnerabilities

- Detected versus Exploitable versus Attack-prone
  - What vulnerability type (CWE)?
  - What severity (CVSS) per CWE type(in the NVD)?
  - Time to discover?
  - Distance from the attack surface edge?
  - Detectable in how many ways?
  - Who detected? Who exploited? What assets involved?



Stage → Complications → Where → How → Exploited → Future

Summary


Where?
How?
RISK
1ST PRIZE
David the Detected
?
Adam the Attack-prone
?
Edwin the Exploitable

Graduate studies at NCSU

Degrees

- PhD
- Master of Science
- Master of Computer Science
  - Track in Data Science
  - Track in Security
  - Track in Software Engineering
- Master of Computer Science (Distance Education)
- Master of Science in Computer Networking
- Master of Science in Computer Networking (Distance Education)

Certificate

- Computer Science
- Data Science Foundations

Images

- https://dementiacarebooks.com/how-to-become-a-dementia-behavior-detective/
- https://pixabay.com/vectors/fish-hook-fishing-hook-recreation-2027781/
- https://prosportstickers.219signs.com/index.php?route=product/product&product_id=37152
- http://www.brianbarber.com/illustration/
- https://prosportstickers.219signs.com/index.php?route=product/product&product_id=37152
- https://drawception.com/game/HM8CfM7pHD/sleepy-fish/
- Vectorstock.com/9961574
- https://requestreduce.org/categories/fish-trap-clipart.html#overlayGallery9_post_17509_fish-trap-clipart-17.png
- http://www.e2studysolution.com/news/How-can-I-become-a-Cybersecurity-Expert
- https://www.zazzle.com/red_star_1st_prize_round_sticker_red-217743138139492519
- https://www.datanami.com/2016/09/23/past-present-future-finance/
- https://easydrawingguides.com/how-to-draw-a-whale/
- https://achievingbeautifuldreams.files.wordpress.com/2015/09/50-50.jpg
- https://www.merchantmaverick.com/best-high-risk-merchant-account-providers/
- https://digest.bps.org.uk/2018/03/21/is-the-future-ahead-not-for-those-born-blind/

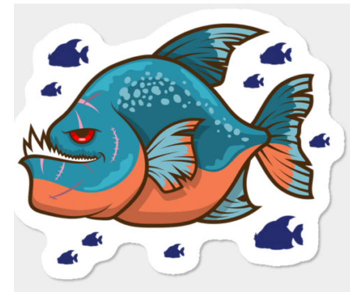
Images

- https://www.monitis.com/blog/why-your-small-business-needs-penetration-testing/
- https://www.foolishbricks.com/day-276-the-needle-in-the-haystack/
- https://betanews.com/2016/06/30/solve-shortage-data-scientists/
- https://www.playstation.com/en-gb/games/need-for-speed-ps4/
- https://www.bizcatalyst360.com/casting-a-wide-net-while-innovating/
- https://simpleprogrammer.com/get-programming-job-no-experience/
- https://towardsdatascience.com/organizing-your-first-text-analytics-project-ce350dea3a4a
- https://www.mnn.com/green-tech/research-innovations/quiz/can-you-pass-governments-10-simple-science-question-quiz
- https://marketeer.kapost.com/programming-for-marketers/
- http://www.devsanon.com/page/4/

Possible fish

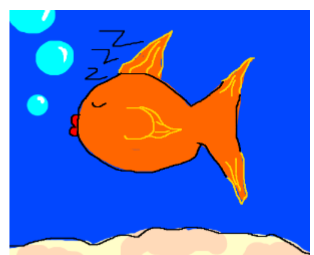
https://prosportstickers.219signs.com/index.php?route=product/product&product_id=37152

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQFnTWQGJI6jLxeHmzDNqJCl2Rrgm2Fp5hiwZFBv3XBKOhG1PC6

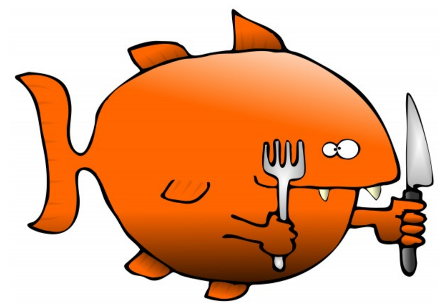


https://www.designbyhumans.com/shop/sticker/mean-fish/660022/

http://www.brianbarber.com/illustration/



https://drawception.com/game/HM8CfM7pHD/sleepy-fish/



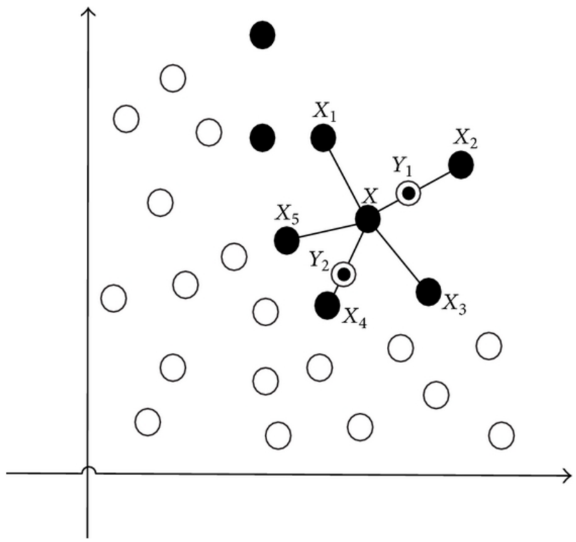
https://suzyssitcom.com/2013/08/can-you-do-the-heimlich-on-a-fish.html

Q: How to synthesize examples of vulnerable software?
A: SMOTE (**Synthetic Minority Over-sampling)**

```
function SMOTE()
    while Majority > m do  delete any Majority item
    while Minority < m do  add something_like(any Minority item )

function something_like( X0 )
    { X1, X2, … }  = k nearest neighbors of X0
    Z =  any of X0
    Y =  interpolate( X0, Z)
    return Y

function minkowski_distance(a, b, r)
   return   ( ∑ abs(a.i - b.i)^r  ) ^ (1/r)
```



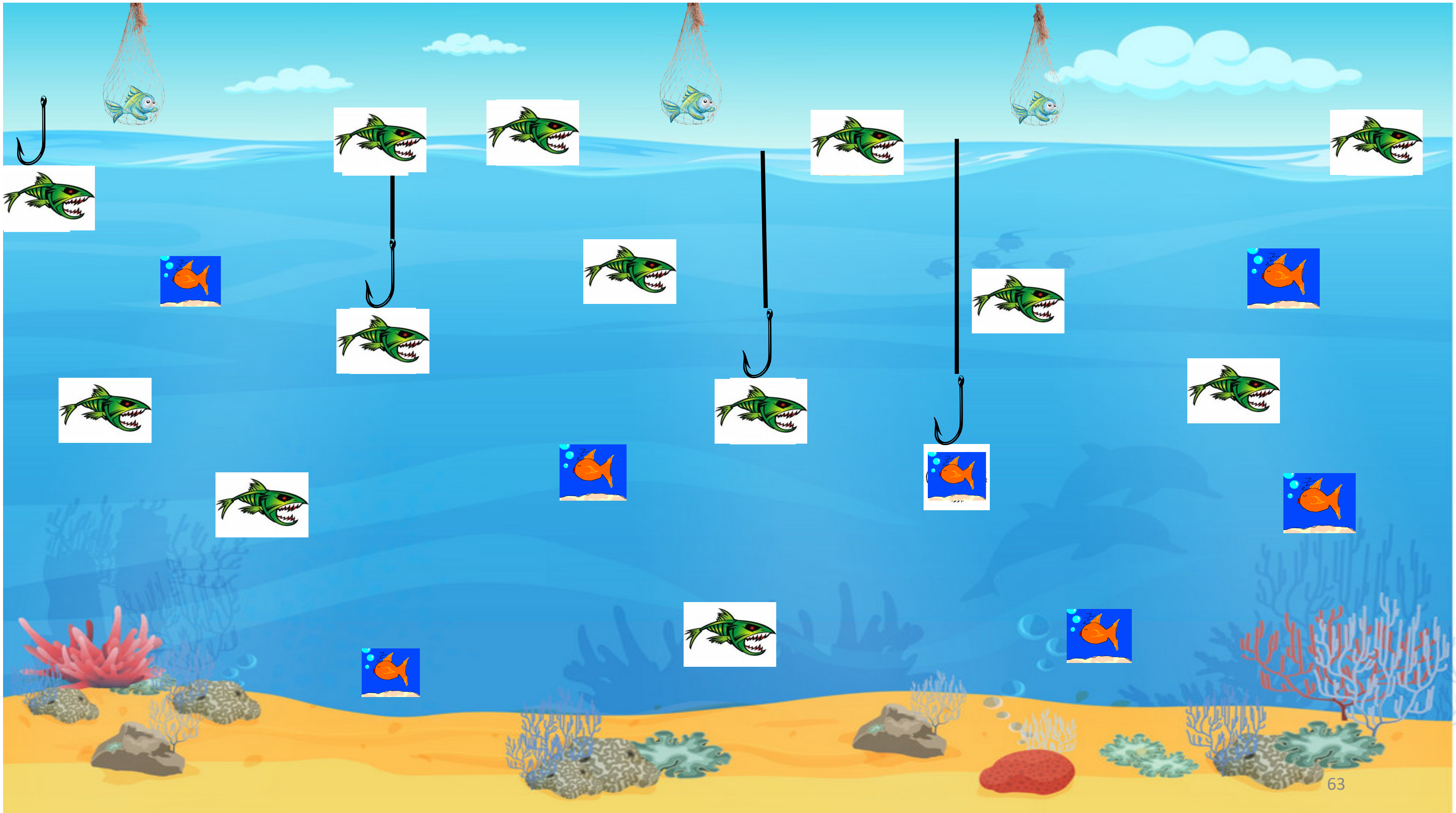
Q: How to do this better?
A1: Tune the magic parameters of SMOTE <m,k,r>

Case Studies

Three empirical case studies

- RHEL4 Linux kernel, PHP, and Wireshark
- Pre-release version control logs
- Post-release security vulnerabilities
- Viewed files as **vulnerable** (>0 vulnerabilities) or **neutral** (none found yet)

	RHEL4 kernel	PHP	Wireshark
Number of committers	557	84	19
Source code files	14,454	1,039	2,688
% files vulnerable	3%	6%	3%
Pre-release version control log data	16 months	2 years	2 years
Years of security data	5 years	3 years, 5 months	3 years, 5 months

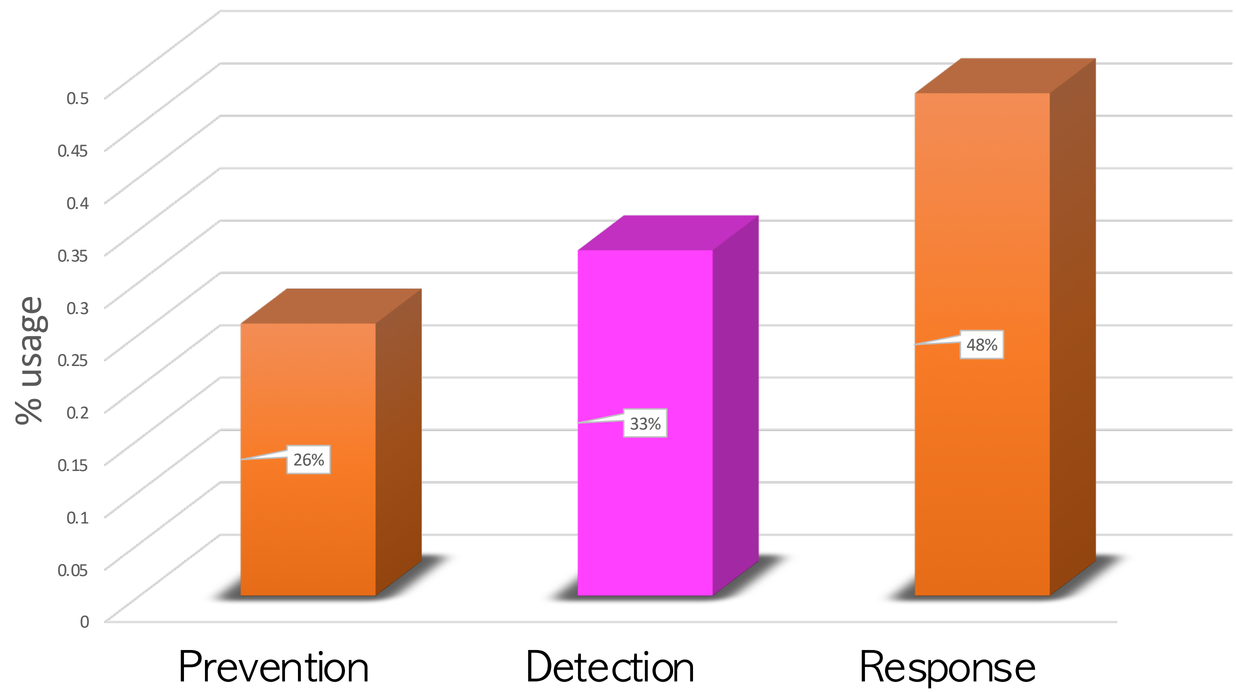


Preliminary Findings

- 5 projects – Linux, Firefox, Samba, Qt, Kodi
  - Median alert count: 10171
  - Median Triage Rate: 17.5%
  - Median Fix Rate: 51.3%
  - Median Unactionable* Rate: 45.9%
  - Median Bug Rate: 23.6%
  - Median Lifespan: 33 weeks
- Security alerts are *Not* likely to be *fixed more often* than non-security alerts
- Security alerts are *Not* likely to be *fixed quicker* than non-security alerts

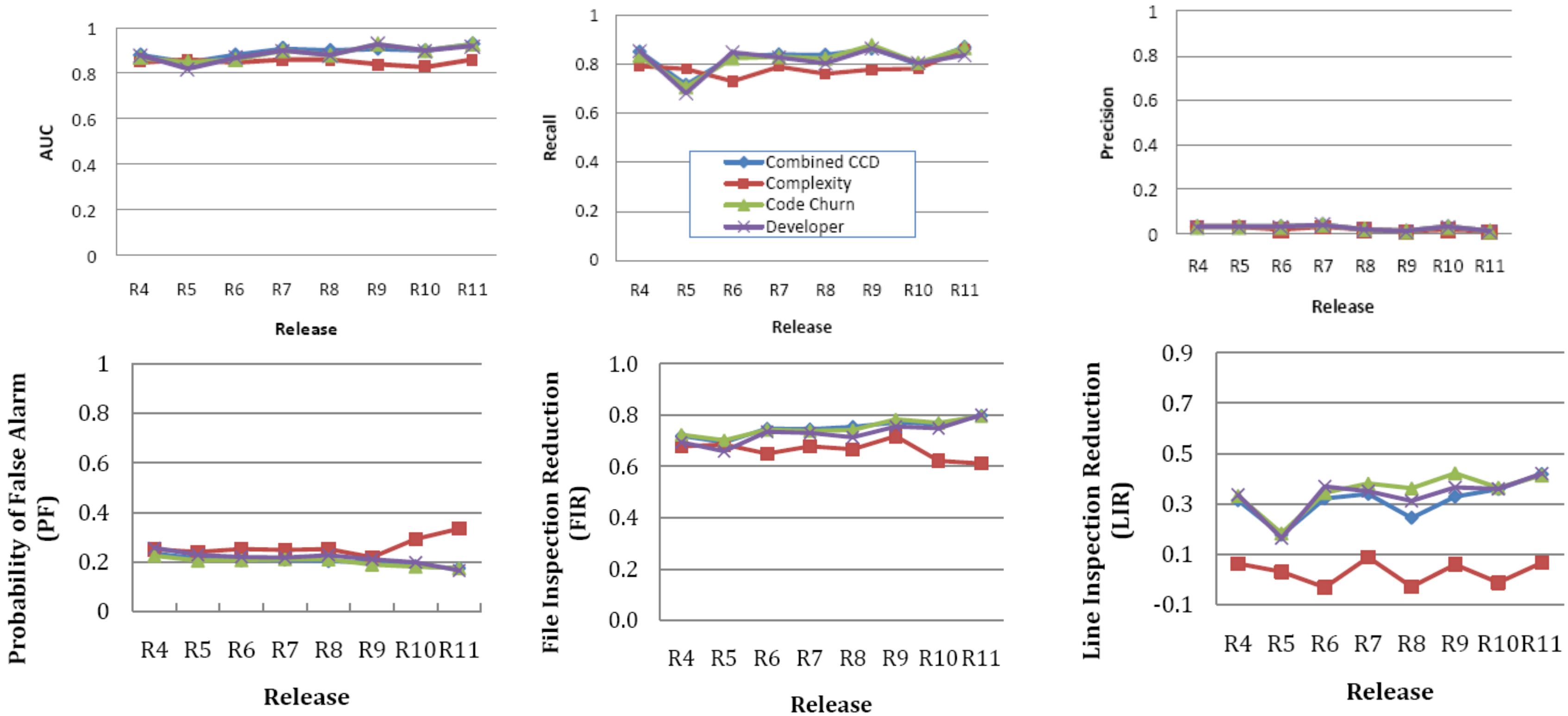
*marked by developer as false positive or intentional

What we currently do with vulnerabilities (BSIMM8)



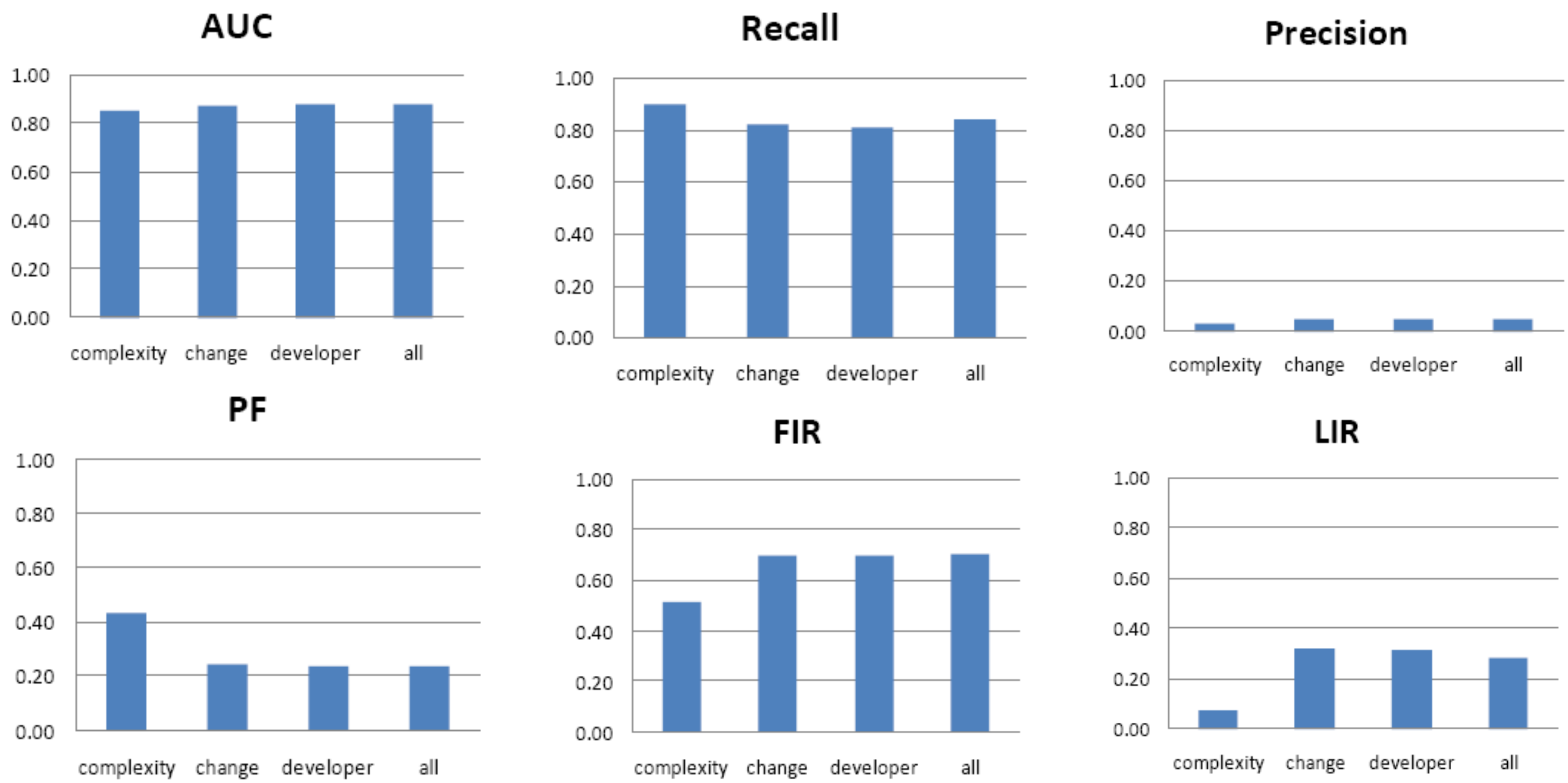
Stage | Complications | Where | How | Exploited | Future

Results: Predictability (11 releases Firefox)



Stage | Complications | Where | How | Exploited | Future

Results: Predictability (RHEL)



Vulnerability Resolution

Vulnerabilities are fixed at a faster rate than defects

In Mozilla, vulnerabilities are resolved 33% more quickly than defects.