Familiarity Breeds Contempt

The Honeymoon Effect and the Role of Legacy Code in Zero-Day Vulnerabilities

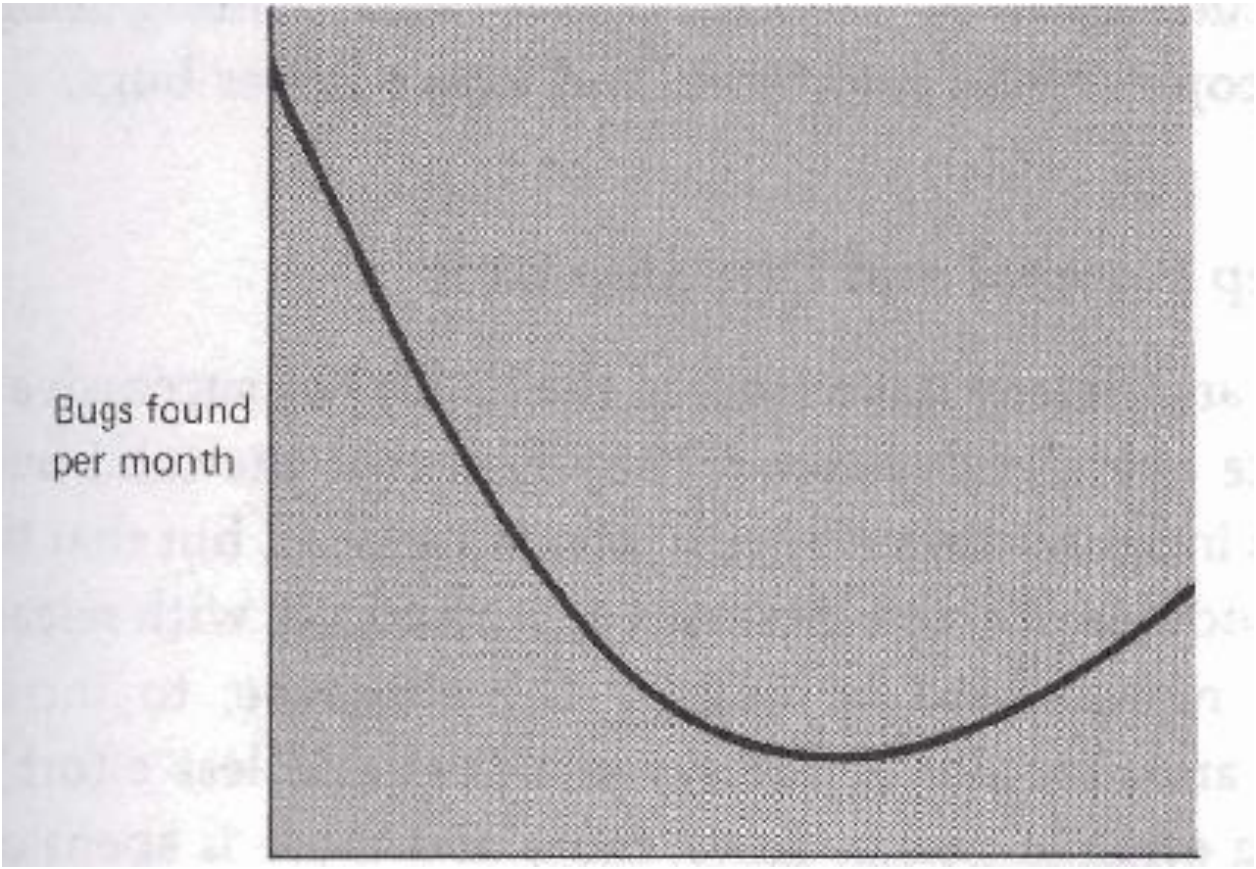
Sandy Clark
University of Pennsylvania

Stephan Frei
Secunia

Matt Blaze
University of Pennsylvania

Jonathan Smith
University of Pennsylvania

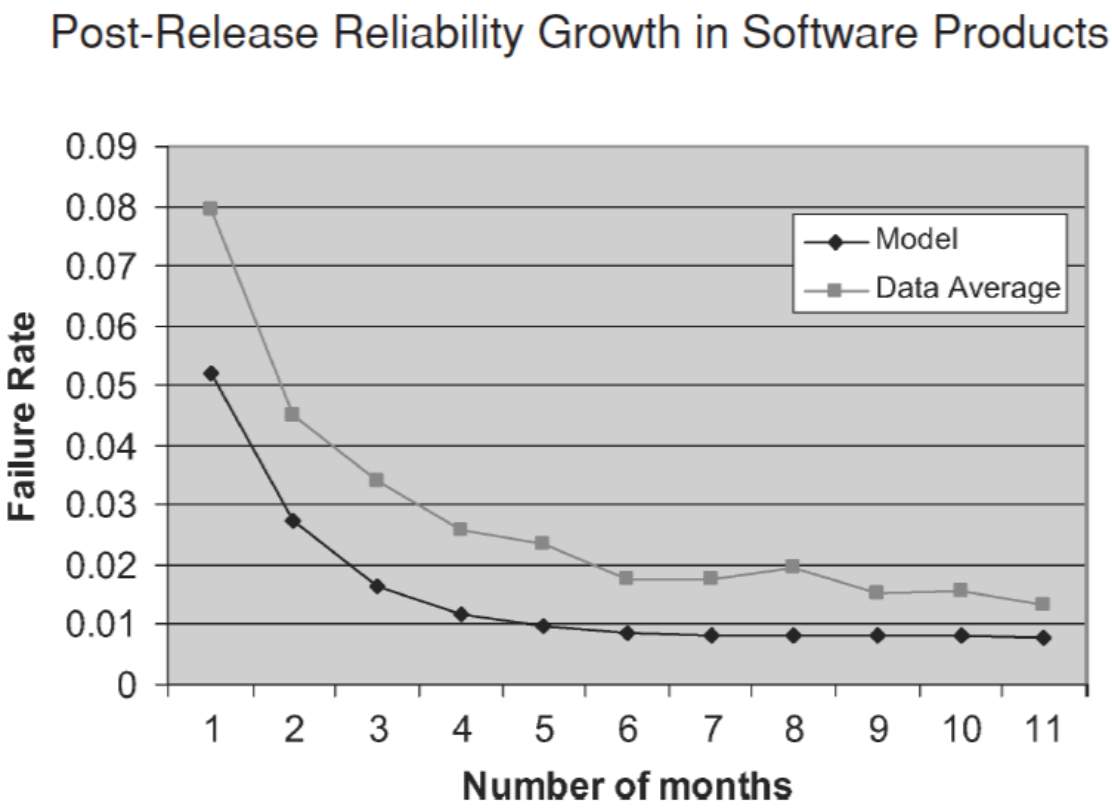
Classic Bug Discovery Model



Brooks: The Mythical Man Month, 1975

This has been highly stable over time

Jalote, et al 2008



Are Vulnerabilities just bugs?

- Conventional wisdom:
  - Vulnerabilities are just bugs with nasty side-effects
  - Software reuse improves security
  - Better software is less vulnerable

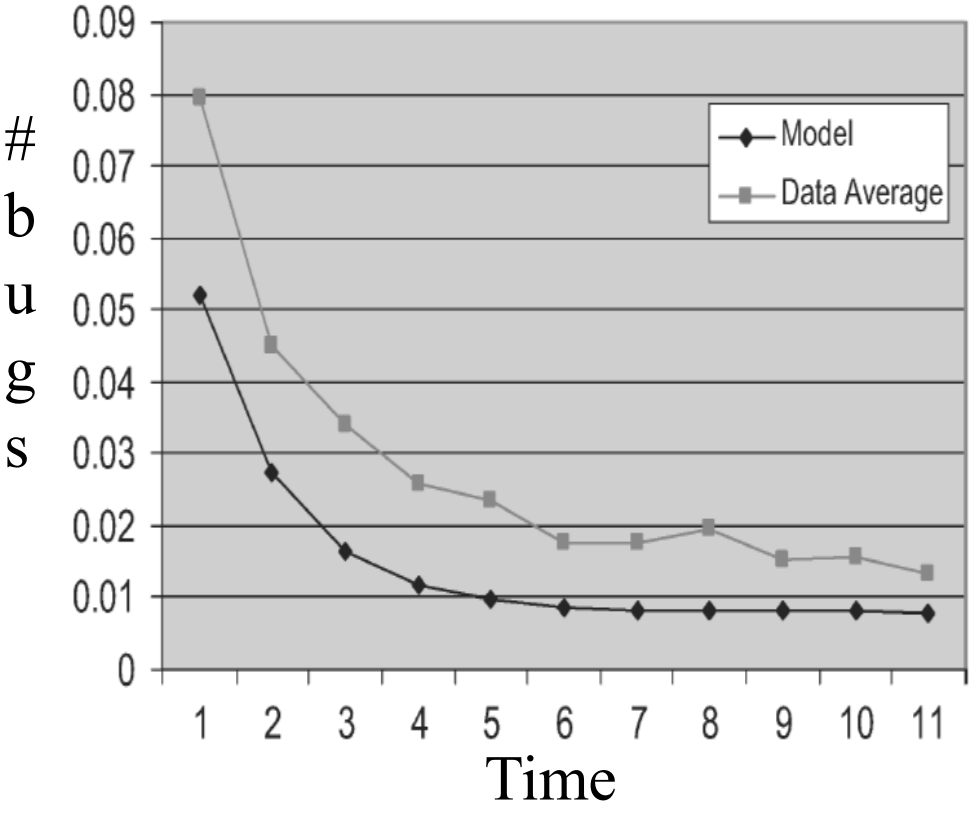
No!

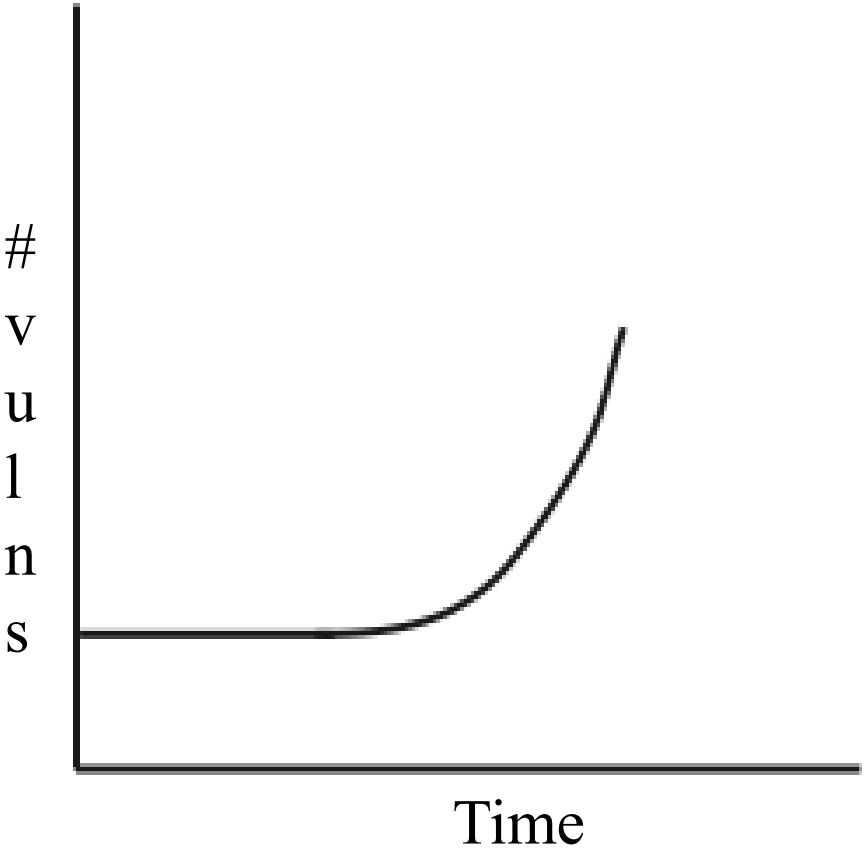
(At least not in the early life-cycle)

Testing the conventional wisdom

- We examined the vulnerability discovery rate of popular mass market software
  - We focused on the early life-cycle after each software version release
- Surprisingly, we found a *post-release Honeymoon*
  - Vulnerabilities discovery rate is *slowest* immediately after release

The Honeymoon Effect





Bugs: Starts fast,
then ***slows down***

Vulnerabilities: Starts slow,
then ***speeds up!***

Our Data

Over 30,000 publicly disclosed vulnerabilities from NIST's NVD and CVE correlated with 7 security info providers - Bugtraq, Secunia, etc

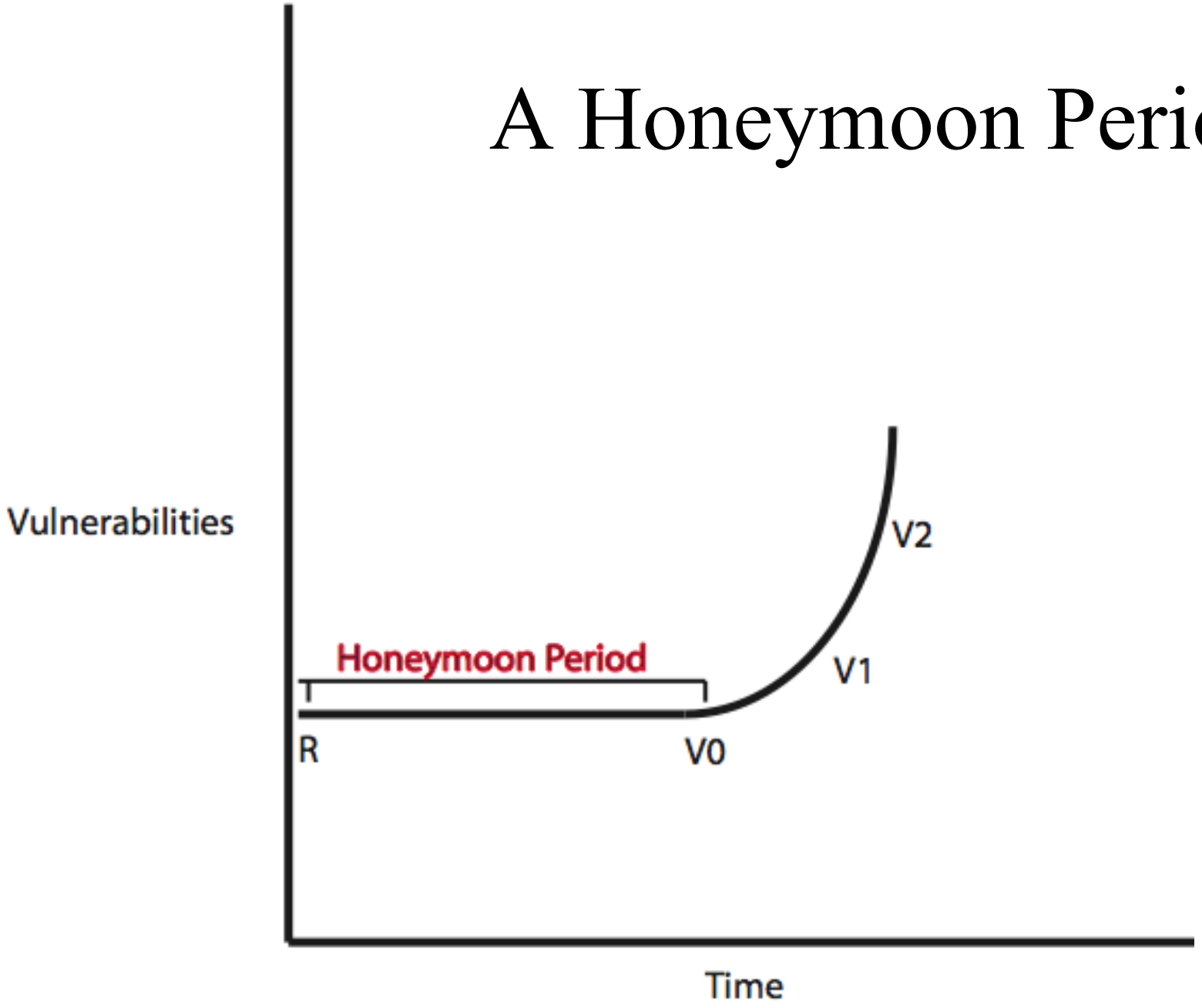
Most popular programs: Operating Systems, Server applications, User applications - open source and closed source

We found 700 software release versions of the 37 most popular mass market products for which accurate release date information was available.

We correlated the disclosure date of a vulnerability with the date of release for all versions of the products affected

We measured the Days-to-discovery for first 4 vulnerabilities of both the major and minor versions

A Honeymoon Period?



The Honeymoon Ratio:

Positive Honeymoon Ratio: $V_0 - R / V_1 - V_0 > 1$

Negative Honeymoon Ratio: $V_1 - V_0 / V_0 - R < 1$

Honeymoon Period:

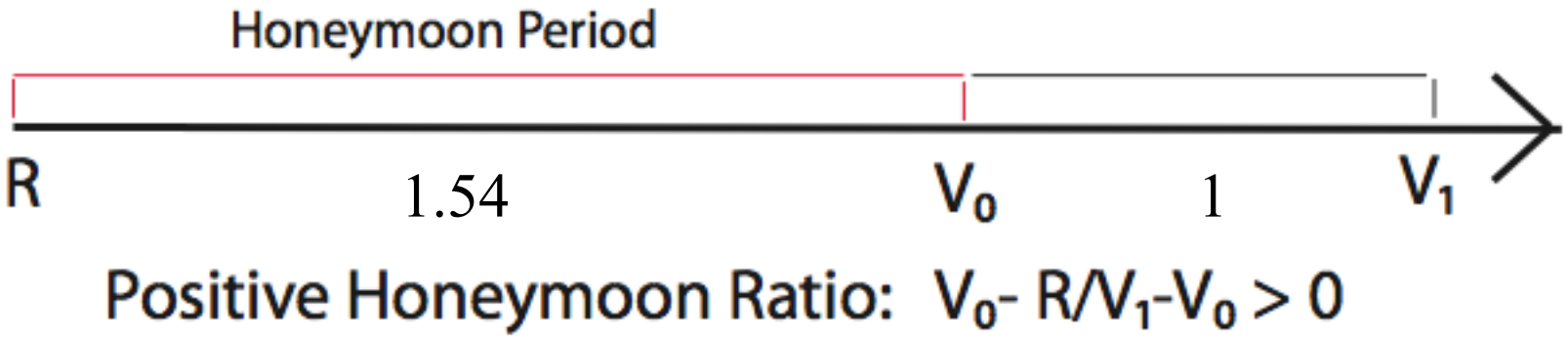
In 62% of releases the Honeymoon Ratio was *positive!*

The median length of the Honeymoon Period was *110 days*

High variance - some found quickly

The median overall Honeymoon Ratio $(V_0\text{-}R)/(V_1\text{-}V_0) = 1.54$

(Includes both positive and negative honeymoons)



Okay, there's a Honeymoon Effect

Let's figure out why.

Percentage of Positive Honeymoons by Year

Year	Positive Honeymoons
1999	56%
2000	62%
2001	50%
2002	71%
2003	53%
2004	49%
2005	66%
2006	58%
2007	71%

The Honeymoon Effect is consistently positive across different Software Environments

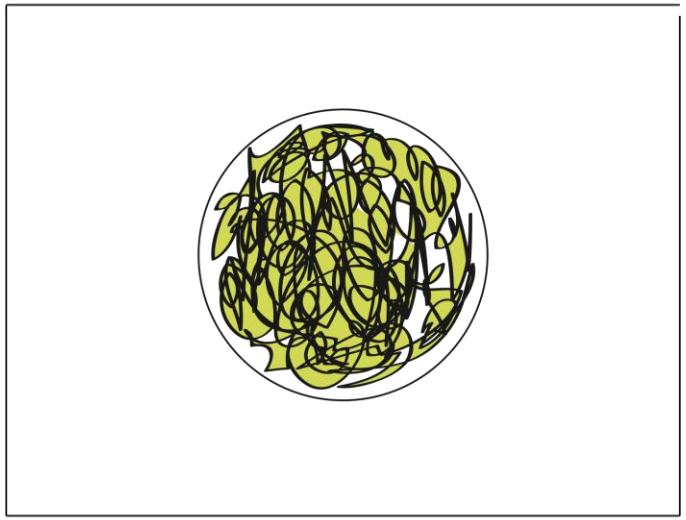
Positive Honeymoon Effect:

- Operating Systems
- Server Applications
- User Applications
- Open Source
- Closed Source
- Major Releases vs Minor Releases

So, what's going on?

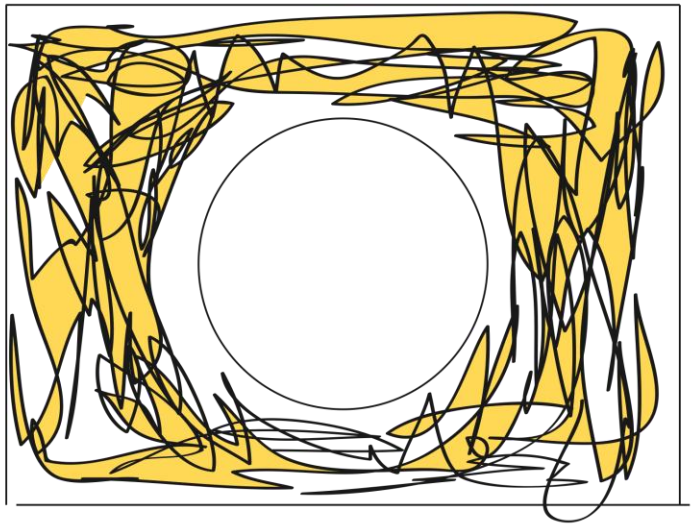
Intrinsic Properties

Things the Programmer can Control ***BEFORE*** Release



Extrinsic Properties

Things Entirely Outside the Programmer's Control



Extrinsic vs. Intrinsic Properties - A consistently positive honeymoon

Intrinsic:

- The Honeymoon Effect occurs within a **single** release. Improvements in software quality occur **only between** releases.

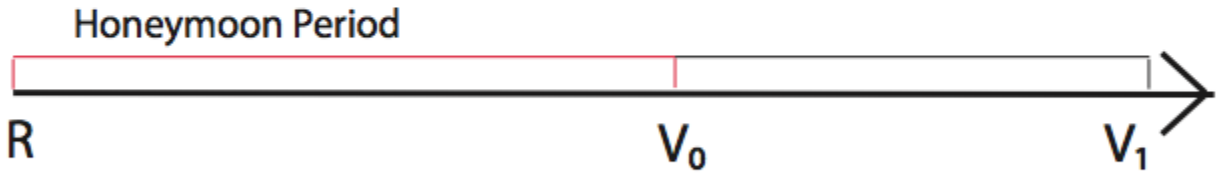
Extrinsic:

- FBI 2009 Internet Cyber Crime Report:
  - "…a 22.3% increase as compared to 2008"
- SANS 2009 Trend Report
  - Rising numbers of zero-day vulnerabilities
- Symantec 2009 Trend Report:
  - 71 percent increase in 2009 in new malware signatures

What about Major Releases? (eg. 2.0)

Percentage of Positive Honeymoons rose from 62% to 72%
Median length of honeymoon rose by 9%
Honeymoon Ratio rose from 1.54 to 1.8

Major Releases Honeymoon Ratio: 1.8



All Releases Honeymoon Ratio: 1.54

Open Source vs. Closed Source?

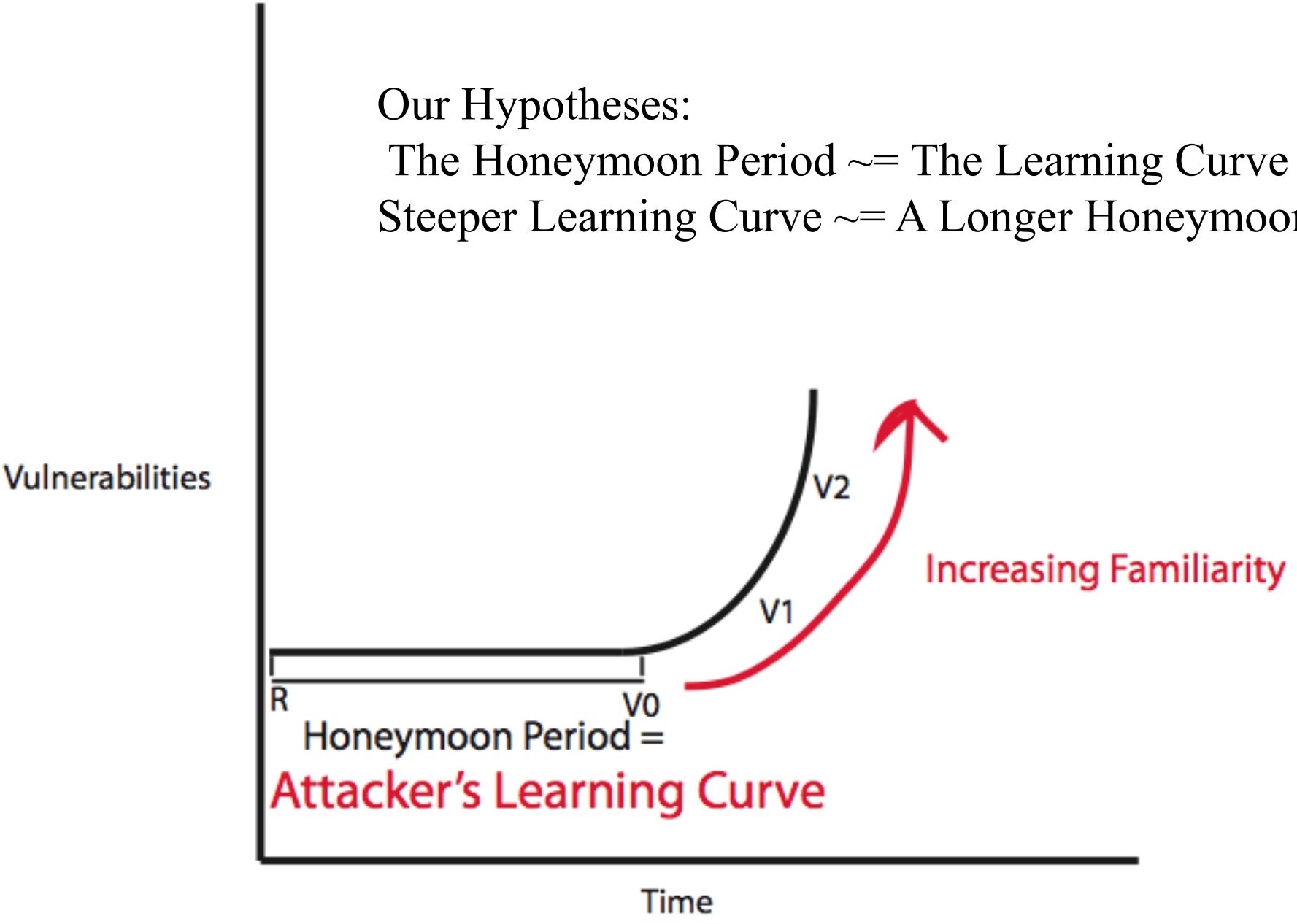
Type	Honeymoon Period	Honeymoon Ratio
Open Source	115 Days	1.23
Closed Source	98 Days	1.68

More new code, release more often, less backward compatibility:
The Longer the Honeymoon Period

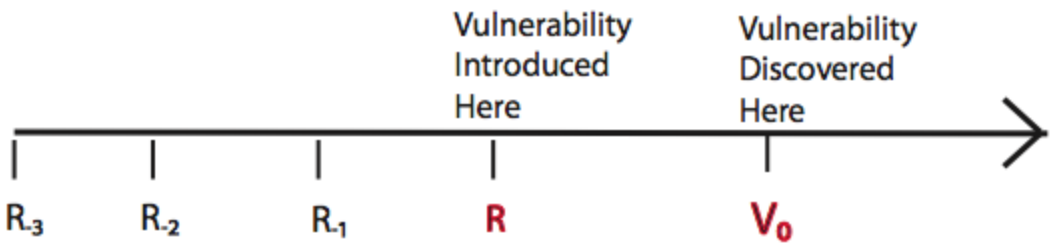
Our Hypotheses:

The Honeymoon Period ~= The Learning Curve

Steeper Learning Curve ~= A Longer Honeymoon



Two types of V_0



Progressive Primal

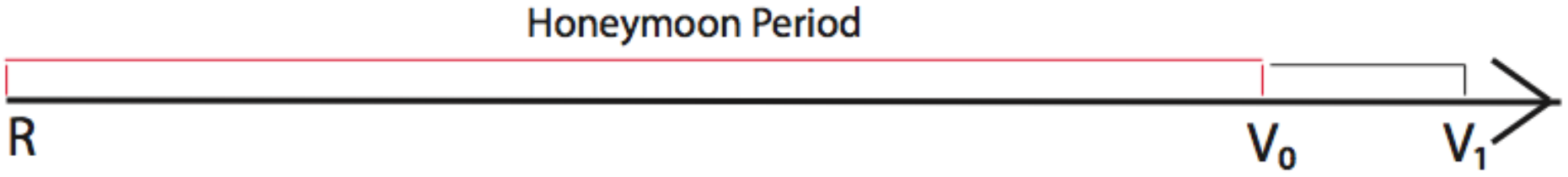
Regressive Primal

Regressive Vulnerabilities

Type:	Total Primals that are Regressives:	Total Positive Regressive Honeymoons:
ALL	72%	63.2%
Open Source	83%	62%
Closed Source	59%	64%

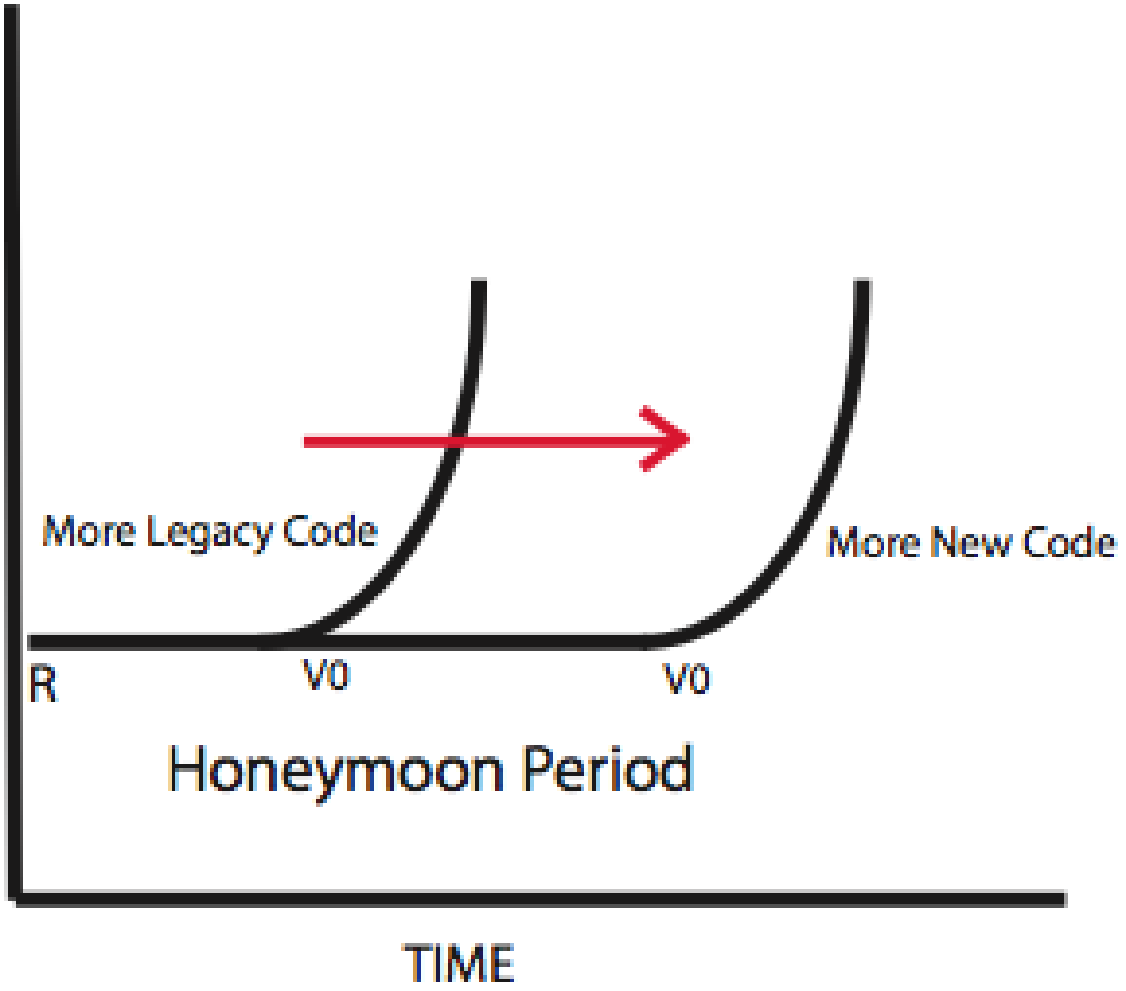
Progressive Honeymoons last longer

Regressives Only Honeymoon Ratio: 1.4



Progressives Only Honeymoon Ratio: 3.1

New code *even with new bugs* has a longer Honeymoon Period!

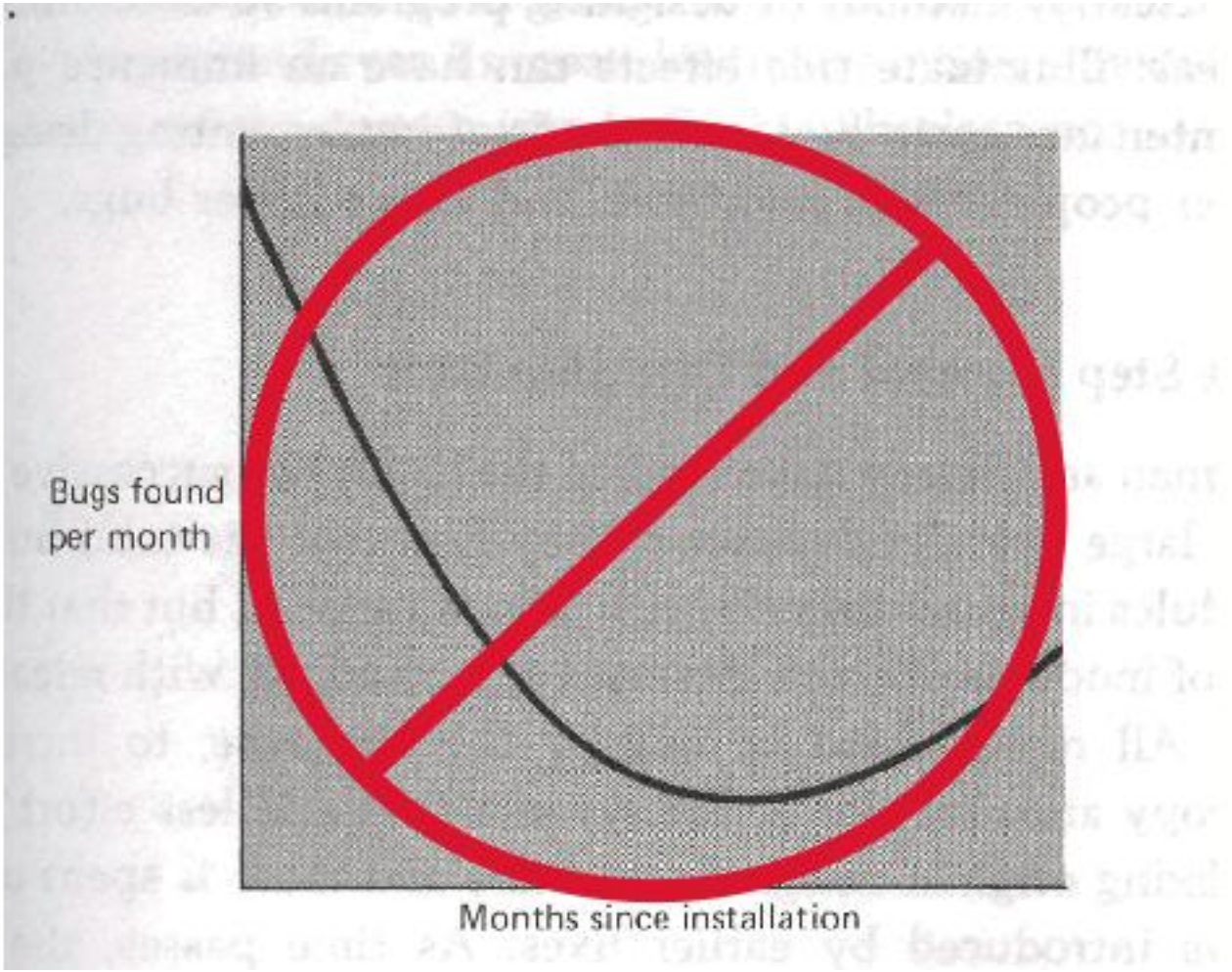


A Look at the Honeymoon Effect from the Attacker's Perspective

Take it away, Renderman

The Honeymoon Effect teaches us:

Vulnerabilities != Bugs



So What?

- The Honeymoon Effect is caused by an ***Extrinsic Property - the effects of which can be measured!***
  - Identify all the Extrinsic properties
  - Understand the role they play
- Then:
  - Develop predictive metrics - quantify risk and relative security
  - Develop ways to prolong the honeymoon period for new software
  - Develop ways to reduce the value of familiarity to the attacker

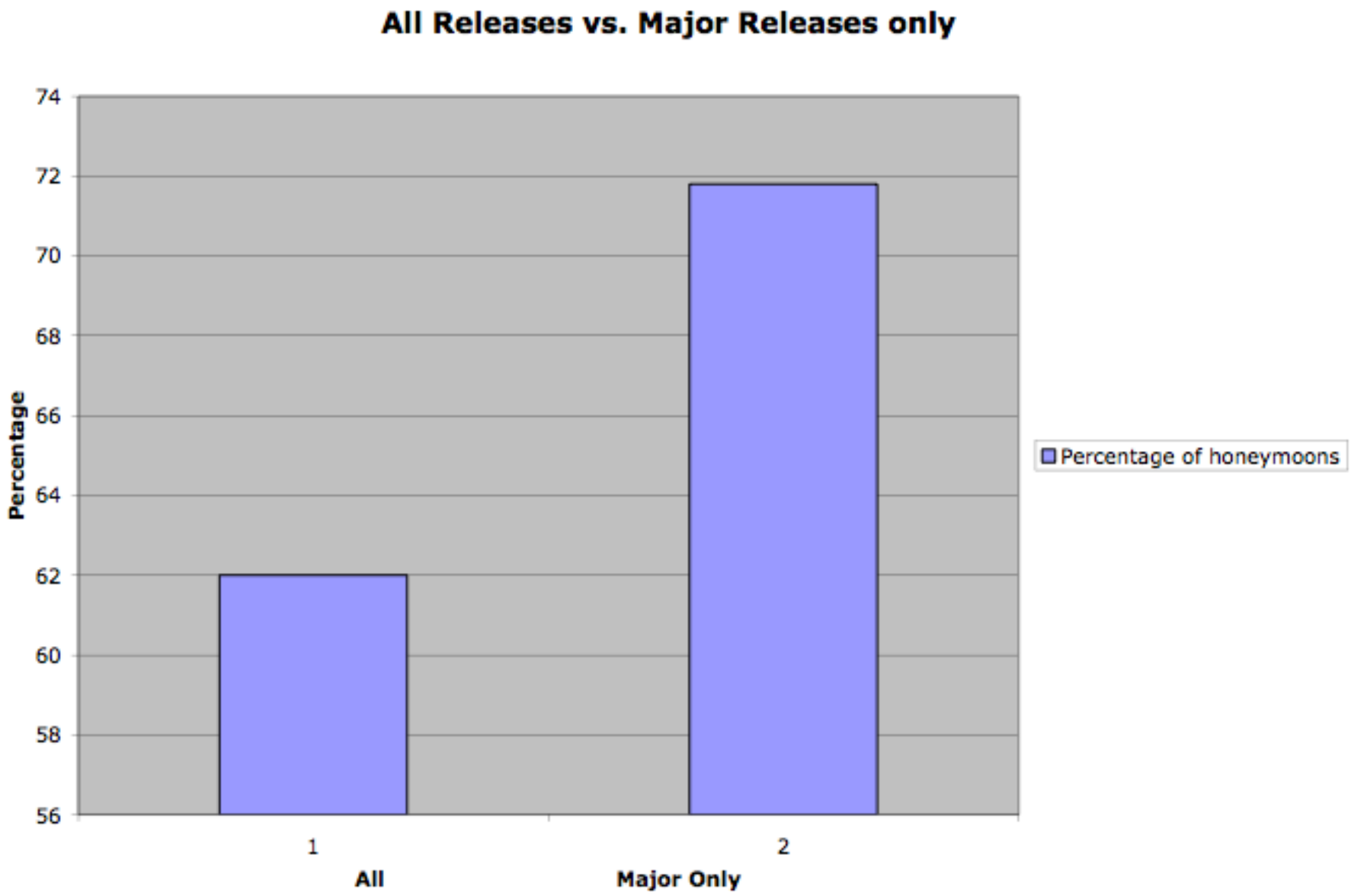
The Honeymoon Effect teaches us (cont.)

- Evidence for something security people have long suspected - *security vulnerabilities are different than software defects*
- Why? Because the *Extrinsic Properties* are different

This suggests we might aim to alter the Security Arms Race to break the ‘patch and pray’ cycle

Open Source vs. Closed Source?

Type	Honeymoon Period	Honeymoon Ratio
Open Source	115 Days	1.23
Closed Source	98 Days	1.68


All Releases vs. Major Releases only
74
72
70
68
66
64
62
60
58
56
Percentage
1
2
All
Major Only
Percentage of honeymoons

Our Hypotheses:

The Honeymoon Effect ≈
The attacker's Learning Curve

The Higher the Learning Curve,
the Longer the Honeymoon Period!

Progressive and Regressive Primals

Progressive vulnerability: A primal resulting from new code (new features = progress!)

ie: if the current version is N, the primal vulnerability only affects version N and does not affect any previous versions

Regressive vulnerability: A primal which originates in a version released earlier than the one in which it was discovered.

ie: if the current version is N, the primal vulnerability also affects versions N-1, N-2…K

Note: If a regressive vulnerability affects versions N through N-2, but not N-3 it almost certainly originated in the code new to version N-2. *BUT, was not discovered until version N.*

Understanding the Honeymoon Effect and get a perpetual honeymoon

Vulnerabilities in Legacy Code