Find My Sloths: Automated Comparative Analysis of How Real Enterprise Computers Keep Up with the Software Update Races

Omid Setayeshfar, Junghwan "John" Rhee,
Chung Hwan Kim, and Kyu Hyung Lee

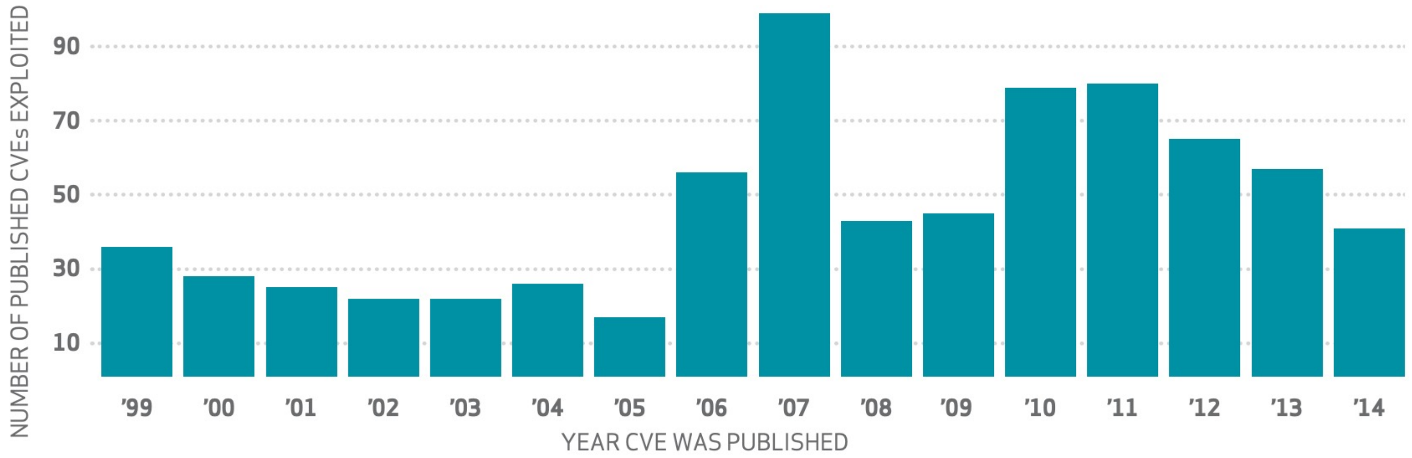
UNIVERSITY OF GEORGIA

UNIVERSITY OF Central Oklahoma

UT DALLAS
The University of Texas at Dallas

Motivation

- Updating software and applying patches is a crucial part of maintaining software ecosystem safe.
- In the Equifax breach case, the attacker exploited a known vulnerability whose patch was available a few months before the accident.
- A study [Verizon data breach report] shows that more than 99% of exploited vulnerabilities used by attackers more than one year after their public disclosure.



Count of exploited CVEs in 2014 by CVE publish date
Source: Verizon data breach report

Software Updates are Complicated

- Having all the users keep all the programs on their computer up to date is ideal, we are very far from it in the real world.
- Let us first understand the current status in our environment!
- **Challenges in understanding software updates**
  - Need to track software release, delivery, arrival, and installation
  - Software vendors perform updates with their own ways.
  - Some software tracks update information. Others don’t. We need a systematic way.

Observations on an Enterprise Software Deployment

- **Isn't it solved by previous approaches?**
  - 774 machines in an enterprise environment
  - Only 14.2% is found in National Software Reference Library (NSRL)
  - Only 75.3% is found on VirusTotal
- Need a comprehensive coverage of all installed software.
- Non-standard channels to update software
- We cannot rely on software vendors' internal information or third-party database.

Goals

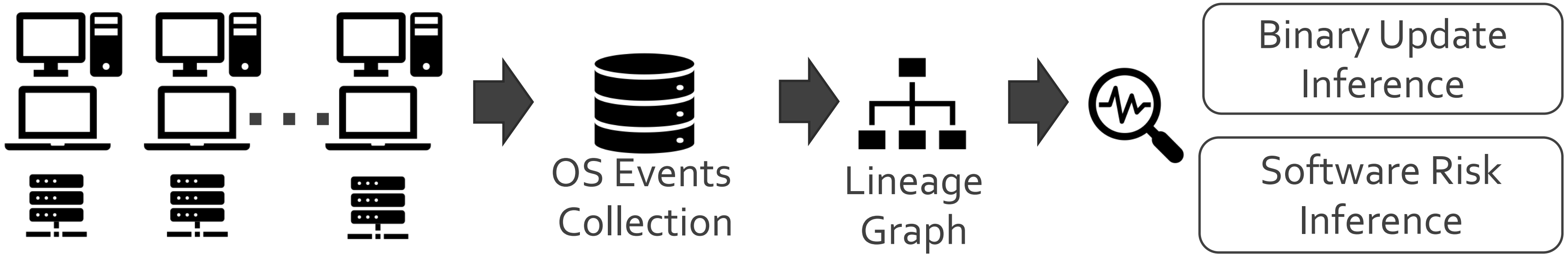
- Understanding software update behavior comprehensively
  - Without relying on internal vendor info or third-party database
- Observation of real environments and draw conclusions on desired properties of update software
- Understanding ***Sloths*** (slow software or machines in updates)

Our Work - Find My Sloth

- Systematic study of software update behavior based on real-world enterprise data.
- Covering a total of 113,675 programs in multiple platforms used by 248 people bringing observations of real-world factors in software updates.
- We propose a method to estimate software release time and update delay with only data collected inside the enterprise, without relying on software vendors' release notes or 3rd party (e.g., VirusTotal, NSRL).

Approach

- Automated tracking of software binary information
- Monitor software update using OS event monitoring
  - Windows : Event Tracing for Windows (ETW)
  - Linux: Auditd
- Inference of software risk based on update delays



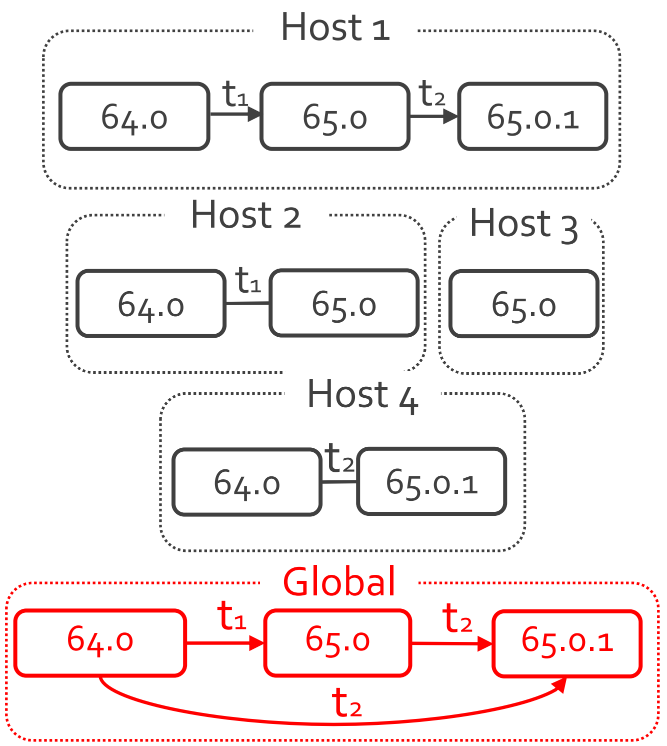
Implementation

- OS Event tracking system is installed in servers, laptops, and desktop computers with multiple OS versions
  - ETW in Windows 7, 10
  - Audit in Ubuntu, Redhat, CentOS Linux
- Tracking the history of execution and modifications of binaries

Platform	Operating System Events
Windows (ETW)	WInExec, WriteFile, WriteFileEx, WriteFileGather
Linux (Audit)	Execve, write, writev, pwrite, pwritev, pwrite64

Software Update Inference with Binary Update Lineage Graph

- **Lineage Graph Generation**
  - G(V, E)
    - V is a set of vertices representing program binaries (SHA256)
    - E is a set of direct edges where each edge shows a transition of a binary
  - Graphs are collected from each host.
  - Then a global graph is constructed by aggregating them.



Note: the version numbers inside the nodes are for illustration purposes. We use hashes.

Software Risk Inference

- **Delay in update time**
  - The estimated delay of the installation time behind the latest version in its software distribution.
- **Challenges**
  - Software vendor's information and third-party information is limited to a set of software.
  - We need a metric available for all software.
- **Solution**
  - We infer the software risk score only based on observed information.

Software Risk Inference

- **Inference of Versions**
  - Temporal appearance order of signatures relative to the prior edge nodes in an enterprise
- **Deployment time:** the timestamp of an update of a binary
- **Latest version:** the version most recently discovered in a graph
- **Update time delay** is estimated by the difference of the deployment times of the latest version and the current version in the binary lineage graph.

$$D_{p,v,m} = |d(S_{p,\hat{v},m'}) - d(S_{p,v,m})|$$

Evaluations

- **Q1: What are the observations of software updates?**
  - A1: Summary of 5 Observations
- **Q2: What is the update behavior of each machine?**
  - A2: Case study: finding a sloth in the machine level
- **Q3: What is the update behavior of each software?**
  - A3: Case study: finding a sloth in the application level

5 Observations on Software Updates

5 Observations on Software Updates

1. **Complexity of Update Transitions**
   - N versions can cause up to N^2 transitions

5 Observations on Software Updates

1. **Complexity of Update Transitions**
   - N versions can cause up to N^2 transitions
2. **Unusual Update Transitions**
   - Rollbacks or regressions are not uncommon.

5 Observations on Software Updates

1. **Complexity of Update Transitions**
   - N versions can cause up to N^2 transitions
2. **Unusual Update Transitions**
   - Rollbacks or regressions are not uncommon.
3. **Various Update Deployment Time**
   - Updates released get installed at clients after various delays from less than 10 minutes to several years.

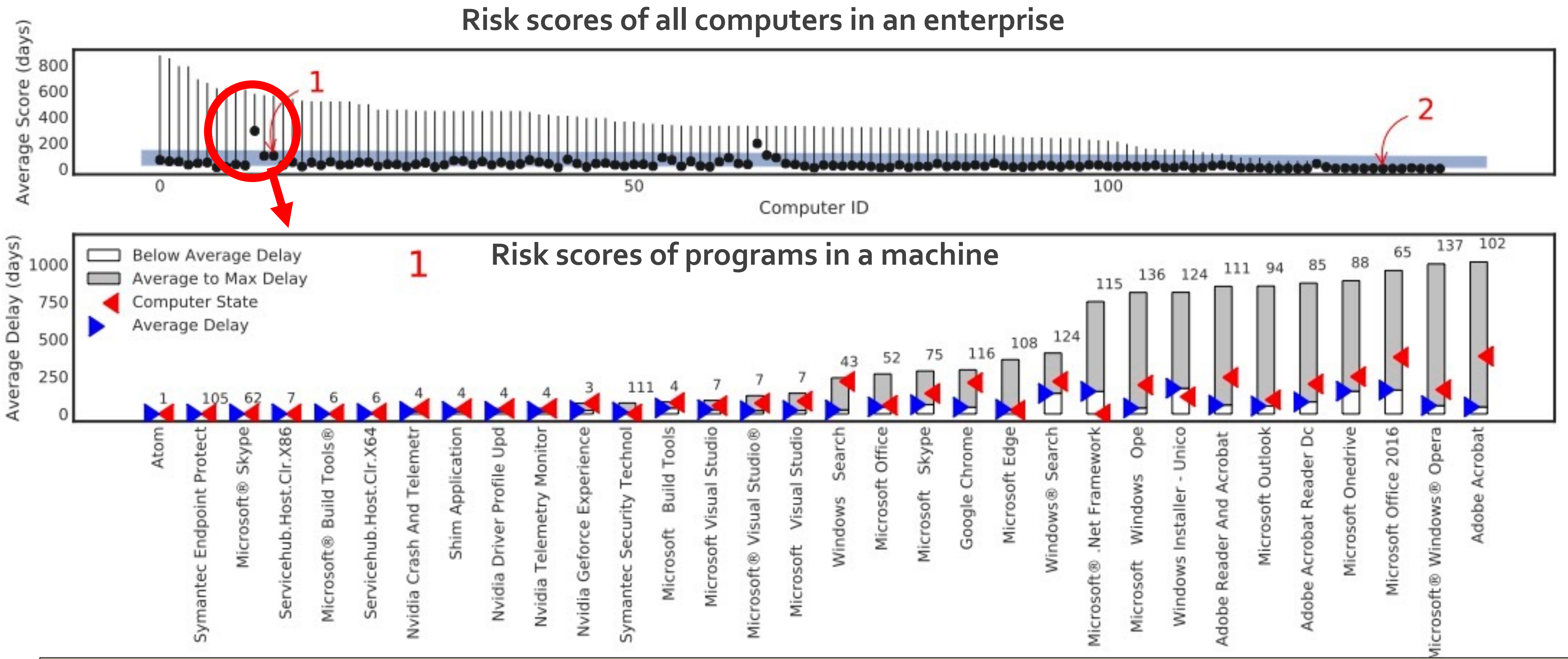
5 Observations on Software Updates

1. **Complexity of Update Transitions**
   - N versions can cause up to N^2 transitions
2. **Unusual Update Transitions**
   - Rollbacks or regressions are not uncommon.
3. **Various Update Deployment Time**
   - Updates released get installed at clients after various delays from less than 10 minutes to several years.
4. **Various Support Period of Software Update**
   - The updates could be provided from no update in years to 1441 updates in three years.
   - On average, we observe 6.4 updates for each product in our environment

5 Observations on Software Updates

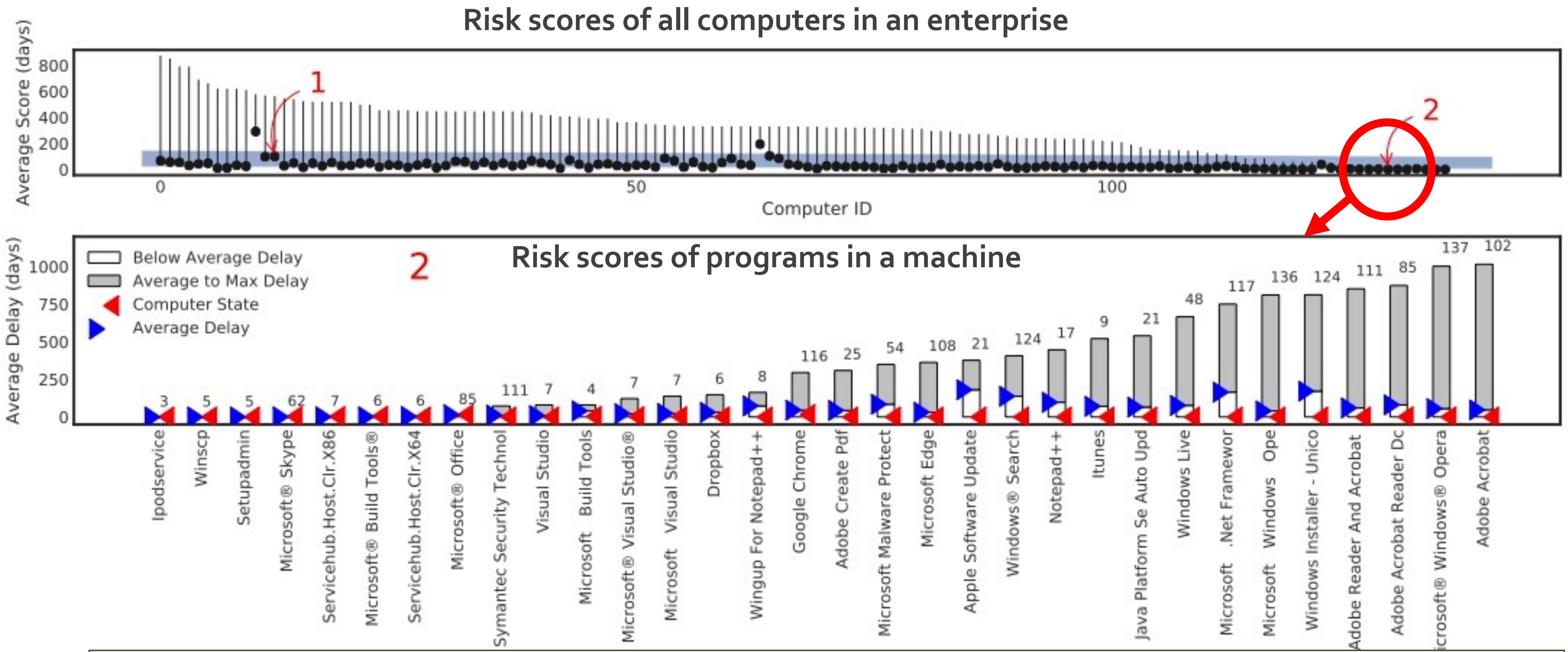
1. **Complexity of Update Transitions**
   - N versions can cause up to N^2 transitions
2. **Unusual Update Transitions**
   - Rollbacks or regressions are not uncommon.
3. **Various Update Deployment Time**
   - Updates released get installed at clients after various delays from less than 10 minutes to several years.
4. **Various Support Period of Software Update**
   - The updates could be provided from no update in years to 1441 updates in three years.
   - On average, we observe 6.4 updates for each product in our environment
5. **A Long-Term Software Usage**
   - Multiple programs are used as an outdated version even after when the update support is no longer provided posing high risk.

Case Study – High Risk Score Machine



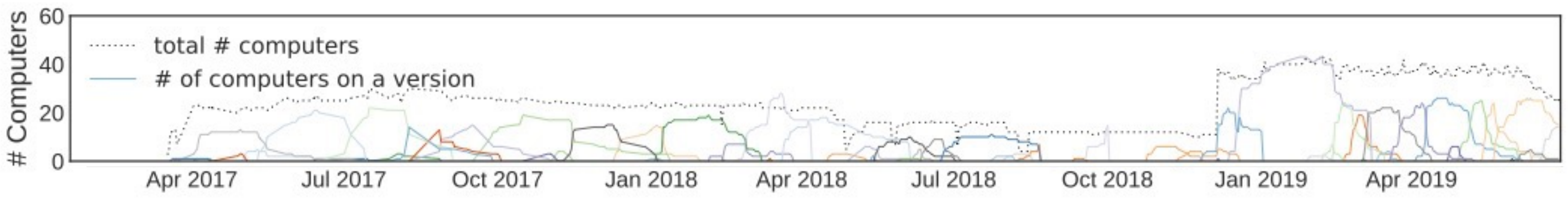
Most scores (red marks) of applications in this machine are higher than the average delay (blue marks) due to high update delays.

Case Study – Low Risk Score Machine



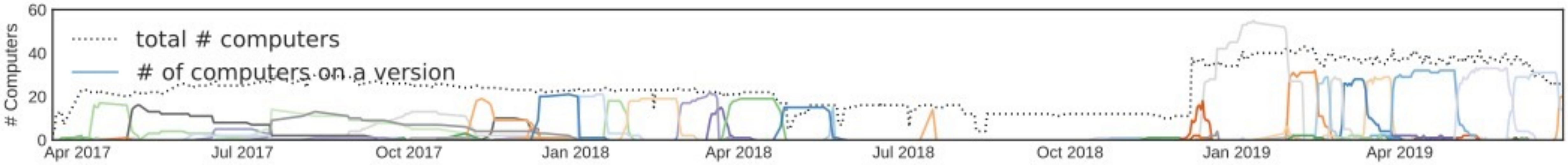
Most scores (red marks) of applications in this machine are below the average delay (blue marks) due to low update delays.

Desirable Software Management - Google Chrome

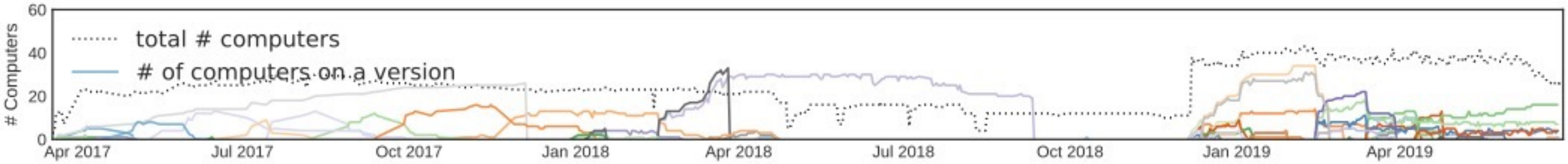


- 75 different versions of Google Chrome over 2 years
- Software update periods
  - Minimum less than a day
  - Maximum 10 months
  - Average 25 days

Undesirable Software Management - Skype, Edge



Skype– some lingering versions in 2017, better behavior in 2019



Edge – some versions present for more than 10 months

Lessons Learned

- **Minimizing Update Deployment Time.**
  - Collective effort is necessary by developers, admins, and users
- **Reliable Updates in Various Transitions**
  - A direct transition vs an incremental transition
- **Software Downgrading**
  - Some users steer away from updates
- **Automated, Enforced, Silent Delivery**
  - Google shows a successful practice of silent delivery
- **Retirement Plan and the End of Support Notice**
  - Knowing a program’s lifespan is helpful to avoid end-of-support programs

Conclusion

- We propose an automated approach to analyze the entire software updates in an organization.
- Utilize only observed metrics instead of relying on developers' info or third-party software information
- Our evaluation with update risk assessment shed light on the current industry practice on a real enterprise environment.

Thank you

Q&A