The vulnerability process: a vulnerability tiger team approach

Marko Laakso

Secure Programming Group

University of Oulu

Finland

Motivation

- Insecure (Vulnerable):
  - *2. Not effectually guarded, protected, or sustained; unsafe; unstable; exposed to danger or loss.*
    - From Webster's Revised Unabridged Dictionary (1913)
  - In context of computer software:
    - may endanger **integrity**, **confidentiality** and **availability**
- Software (InfoSec) vulnerabilities do matter!
  - SANS'99 award given to Bugtraq and NTBugtraq
- Whilst developing preventive tools our group has discovered and reported a couple of vulnerabilities - mostly by accident

The Vulnerability Process

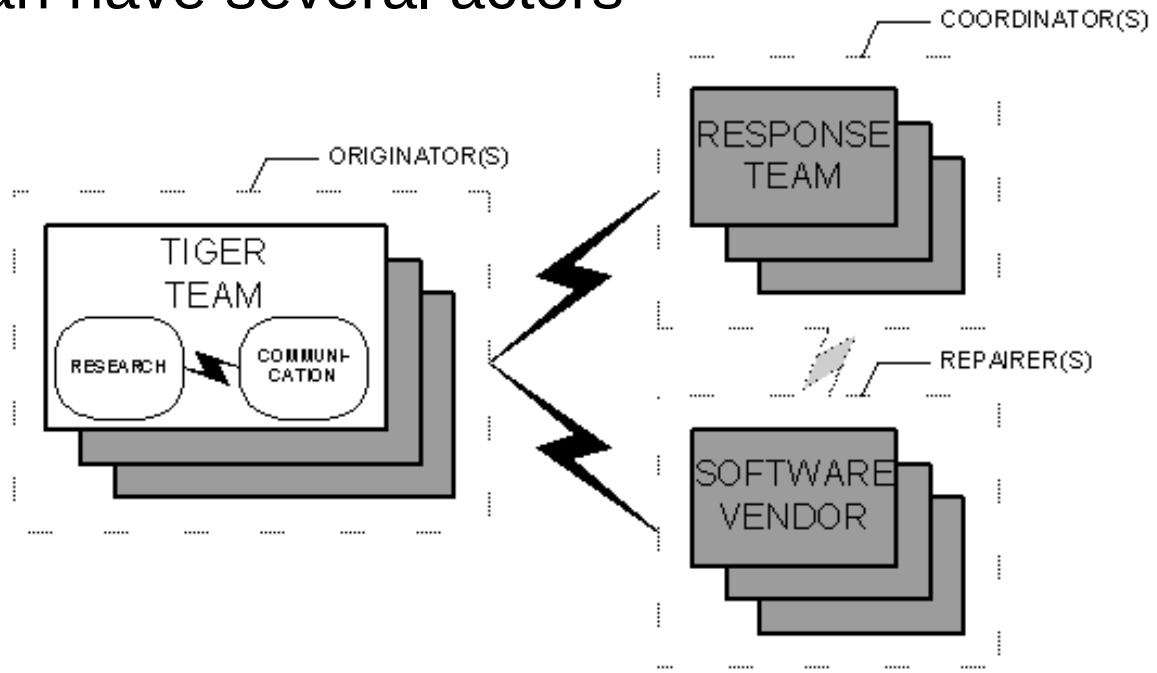
- Presentation contents:
  - Chapter 1: Orientation
    - Roles and Actors
    - Vulnerability Tiger Teams
    - The Life-Cycle model
  - *Chapter 2: Case history*
  - *Chapter 3: Statistics and Future work*

Chapter 1: Orientation

- *a (potential or verified) vulnerability* - *Achilles' heel*
  - in our context: a security related software failure
- *vulnerability case*
  - basic processing unit in vulnerability work
- *vulnerability life-cycle*
  - emergence of a vulnerability case, its evolutionary phases and termination
- *vulnerability process*
  - carries out the life-cycles

Roles and Actors

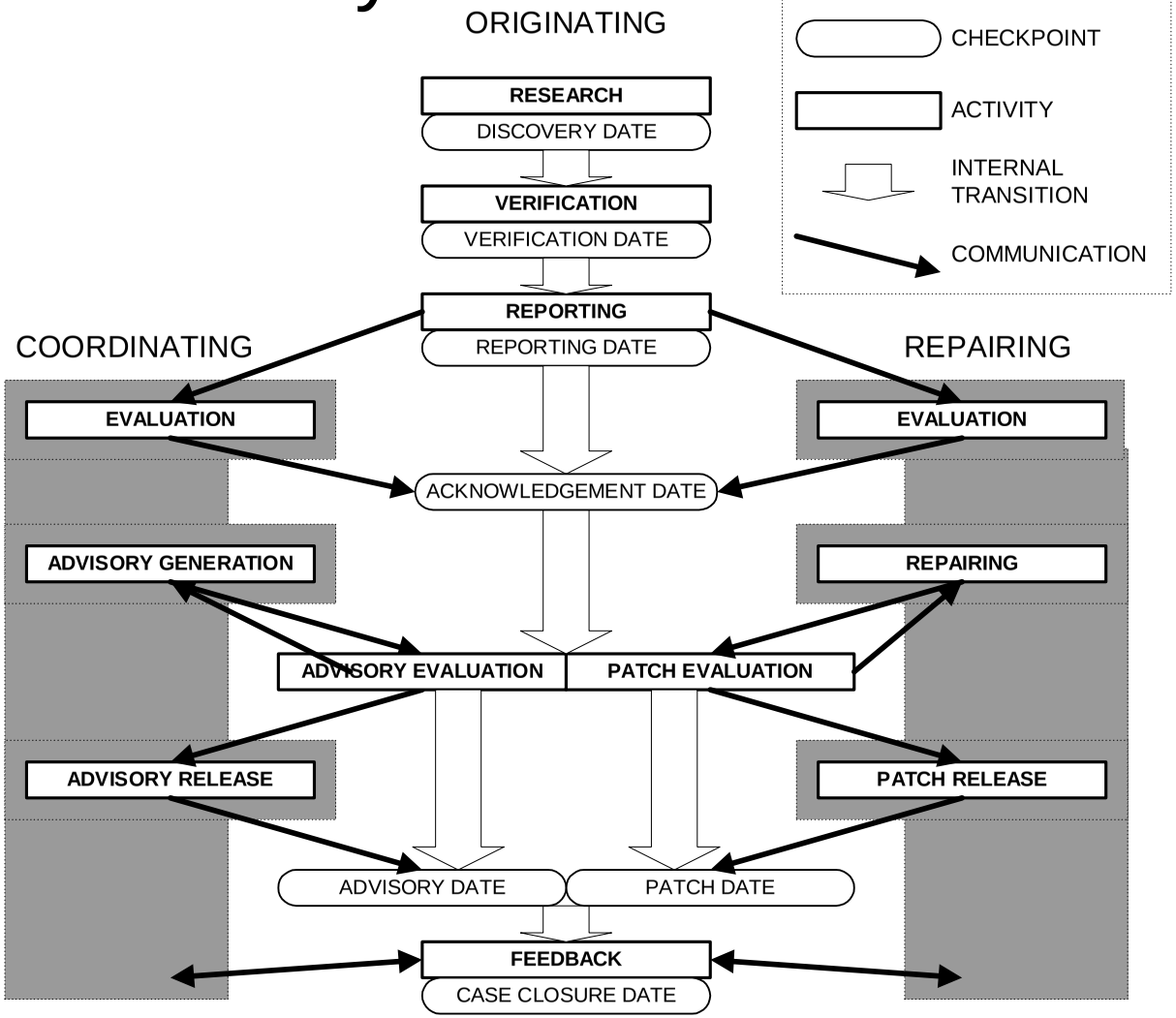
- Three roles in the vulnerability process:
  - Originating, coordinating and repairing
  - Each role can have several actors



Vulnerability Tiger Teams

- Software Vulnerability Tiger Teams?
  - L0pht - 'Cyberspace Underwriters Laboratory'
  - eEye, Netect …ISS, Vendor Teams, OUSPG ...
  - *Government and industry - sponsored teams of (computer) experts who attempt to break down the defenses of (computer) systems in an effort to uncover, and eventually patch, security holes.*
    (Slightly Modified from NSA Glossary)

The Life-Cycle model



Chapter 2: Case history

- OUSPG has history of ~50 vulnerability cases
  - Things we happened to stumble upon while developing tools for proactive vulnerability testing
- It was a learning experience:
  - Observations were gathered on different activities and stages in the vulnerability life-cycle
  - Some of them are covered as rules of thumb herein, rest can be found from accompanying paper in the proceedings

Overall observations

#01 Be patient!

Timezones, holidays, the workload and the number of parties involved contribute to the observed lag.

#02 Seek confirmation before rash conclusions

Large delays may lead to misunderstandings

#03 Be prepared for the "accelerated time frame"

A case considered dormant for months may reactivate and demand fast reaction (e.g. patch evaluation)

#04 Don’t be too paranoid in initial communications

Recipients may not accept PGP not S/MIME and too cryptic reports may be dropped on the floor.

Overall observations

#05 Track and archive everything from day zero!

It may all start as "just one vulnerability" but later on turn into a unmanageable mess.

#06 Identify multi-vendor cases

A disclosure about one product may leave vendors of similar products in an awkward position. Release of vulnerability details tends to stimulate testing of similar products.

#07 Expect prioritisation disagreements

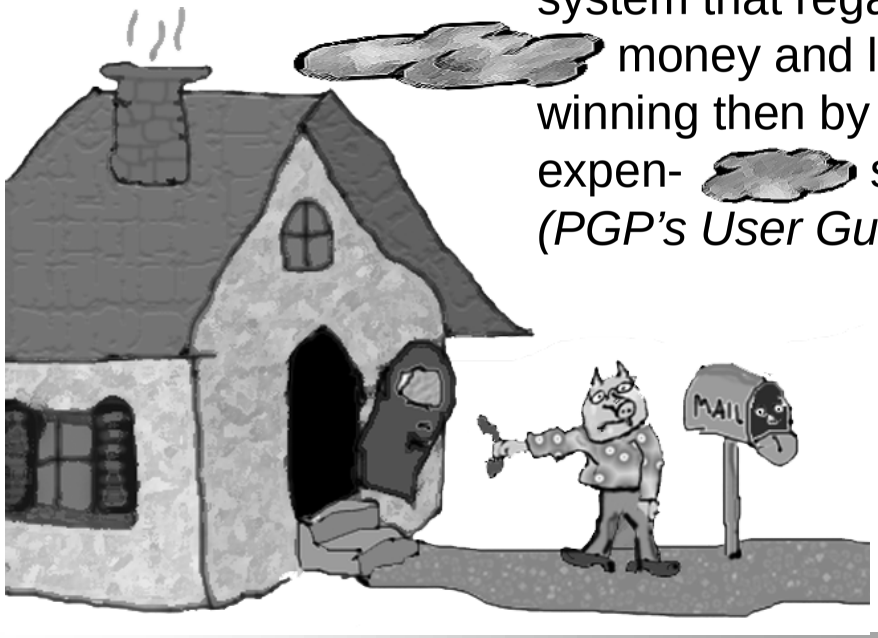
For example local vs. remote UNIX vulnerabilities may result in conflicting impact assessments.

Observations on vulnerability research

#11 Beware of reverse engineering!

License agreements, commercial treaties and modern laws may prohibit reverse engineering. It might be safer just to observe vulnerable behaviour.

- However, it is a well known axiom in the US legal system that regardless of the law, he with the most money and lawyers prevails, if not by actually winning then by crushing the little guy with legal expen- ses.
(PGP's User Guide: Volume II by Zimmermann)



Reporting related observations

#21 Write complete but compact reports

A structured report template is a good starting point.

#22 Involve an independent coordinator

Coordinators such as FIRST members have vendor contacts, case histories and familiarity with variety of environments. Their impact assessment is most trustworthy.

#23 Beware of persistent databases

Sometimes your vulnerability reports may even end up on CDROMs distributed to the customers.

#24 Indicate clearly ’company confidential’ information

Since your report will be more or less public don’t send out core dumps with your production root password.

Reporting related observations

#25 Be prepared to encounter unmaintained software

There might not be an official repairer.

#26 Be ready to demonstrate the exploitability

Although writing yet another buffer overflow exploit may appear to be a waste of effort, it may be required to get a fix. If you don't do it then someone will sooner or later do it for you in public.

#27 Be innovative in getting through the front-line support

Nothing is more frustrating than pasting 2000 lines long vulnerability report into a WWW-form.

#28 Never propose a specific fix

In the worst case the repairers will blindly adopt the proposed (incomplete) fix just to keep you happy.

Observations on acknowledgements

#31 Demand feedback on reproduction failures

Silently ignored reports may be due to unexpected reproduction failures. You should have a chance to reiterate the report. An understanding of reproduction failures may reveal valuable work-around information.

Observations on repairing failures

#41 Allow time for patch production but demand feedback

Other tasks can be of higher priority and can increase delays, and may even halt the mending activity. The originator should be provided with reasonable feedback on progress.

#42 Examine the patch quality

The patch may fix the problem at a specific line of source code, but a new vulnerability may occur a bit further down (+/- 4-lines syndrome). The maintenance people can lack security experience or have insufficient resources.

#43 Be or find a demanding customer

The patch generation process may be excessively customer-driven. A customer has to demand the fix and has to accept it before it is published.

Observations on patch release

#51 Clarify that the initial report just scratches the surface

If a patch is released without verification, it may lead to further vulnerability reports that could have been handled in the same case.

#52 Sometimes a work-around is the only solution

Some vulnerabilities may prove to be too extensive to patch in existing releases.

Observations on advisory generation

#61 Understand the advisory process

Some guidelines on the advisory process will help the originators and repairers to collect and provide relevant information.

#62 Be careful with draft advisories

At the draft stage the advisory may contain unconfirmed and incomplete details that may cause confusions and inconvenience if released prematurely.

Observations on advisory release

#71 A patch alone can be harmful

Descriptive documentation and the patch itself may reveal explicit vulnerability details.

#72 The advisory can be short

This may consist of just a short statement about the security nature of the patches with reference to related public disclosures.

#73 Sometimes the patch doesn't penetrate

Even security-conscious customers may be reluctant to change working systems if the security aspect is not stated clearly enough in an accompanying advisory.

#74 Upgrading isn't always a solution

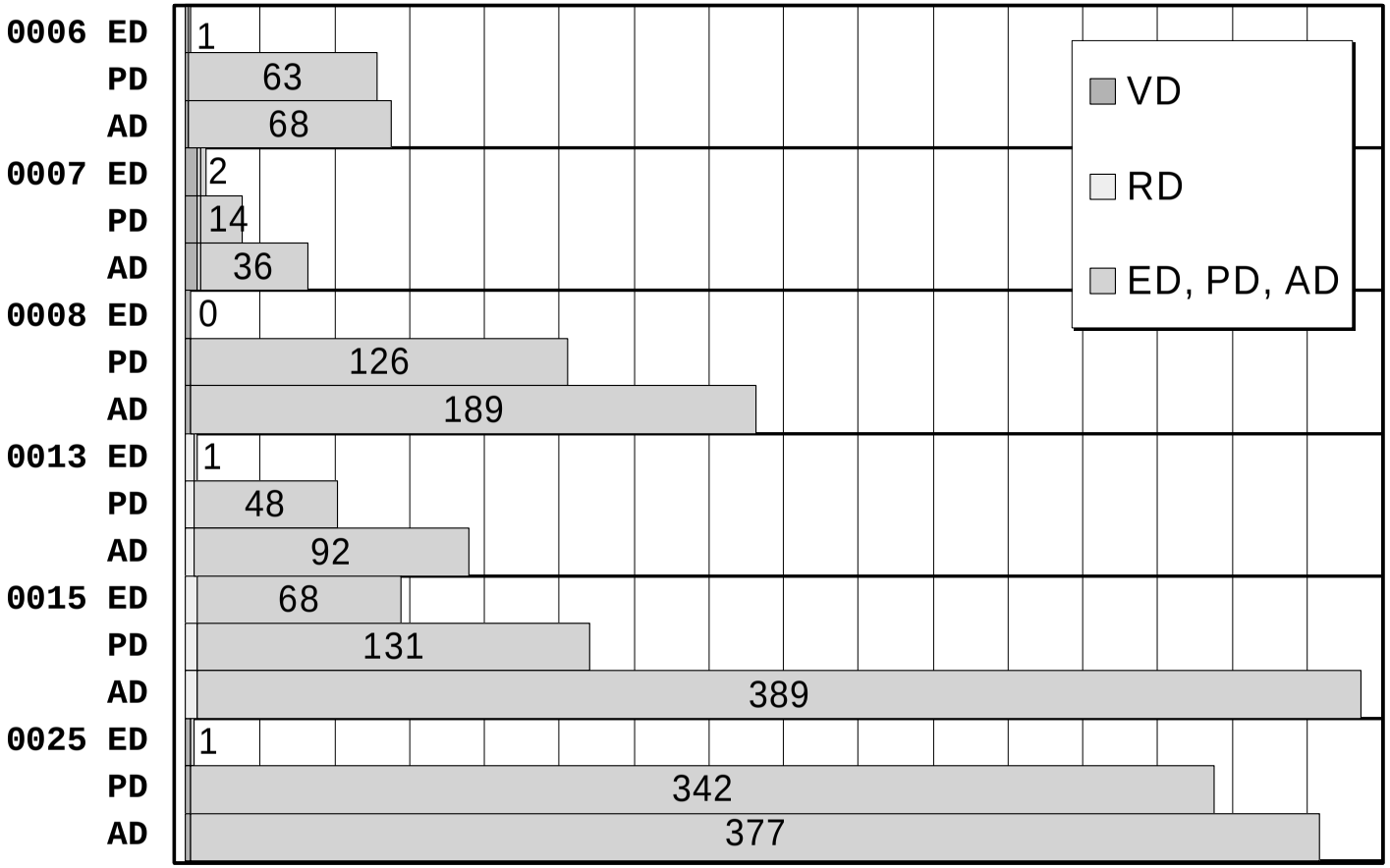
Minimalistic patches help customers who do not wish to mess with new functionality (more than you asked for).

Chapter 3: Statistics and Future Work

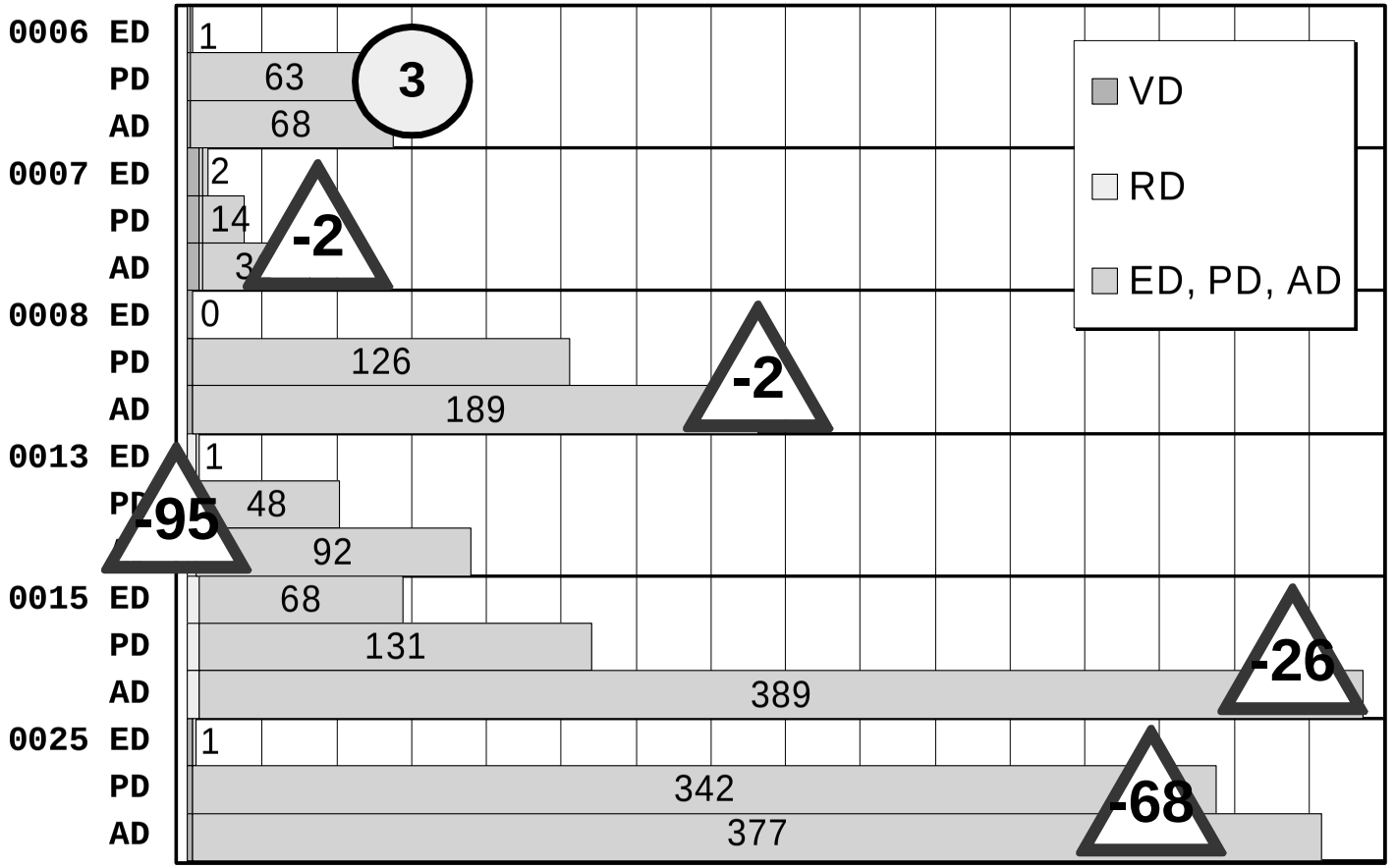
- Metrics support process improvement
- Primitive metrics were adopted by OUSPG:

Metric	Abbrev.	Definition
Verification Delay	VD	Delay between discovery and verification
Reporting Delay	RD	Delay between verification and reporting
Evaluation Delay	ED	Delay between reporting and acknowledgement
Patch Delay	PD	Delay between reporting and patch release
Advisory Delay	AD	Delay between reporting and advisory release
Curing Delay	CD	MAX(PD, AD)
Treatment Delay	TD	VD + RD + CD

Timelines for example cases



Timelines with public disclosures



Future work

- Documentation and support for new actors
  - Vulnerability process guidelines on the web?
  - Any volunteers for the coordinating role? ;)
- Elaboration of the statistics
  - Tracking of public disclosures
  - Combining with vulnerability classification
- Software vulnerability tools
  - Grace-period enforced by license terms

As easy as making the pigs fly?

- Real Life™ processes are complex
- Professional approach required and promoted

Questions and Feedback:

http://www.ee.oulu.fi/research/ouspg

ouspg@ee.oulu.fi