Build it, Break it, Fix it

Fix it
Today

- Break It Presentations
- Theoretical Part: How to Approach Vulnerability Fixing
- Hints for Fix It
How to Approach Vulnerability Fixing
“Your notice of insecure password and/or log-in automatically appearing on the log-in for my website, Oil and Gas International is not wanted and was put there without our permission. Please remove it immediately. We have our own security system and it has never been breached in more than 15 years. Your notice is causing concern by our subscribers and is detrimental to our business.” – Quote from bug report

They were down shortly after. Surprise.
**Vulnerability Lifecycle - Ideal**

- Vendor finds out
- Notification
- Public knows

Nobody knows → Vendor knows → Fix

- Silent Update

Danger
Vulnerability Lifecycle – Common Case in Reality

Nobody knows → Vendor knows → Public knows

White Hats know → Confidential Disclosure → Full Public Disclosure

Danger

No fix within time $X$ (typ. 90 days)
Google and Microsoft in war of words over bug disclosure

The search engine company publicised a critical Windows bug 10 days after informing the software firm about it.

Google and Microsoft are in a war of words after the search engine company publicised a critical Windows bug just 10 days after telling the software firm about it.

The bug, which allows privilege escalation in Windows, was discovered by Google on 21 October. An attacker can use it to access things they should not be able to, and according to Google, it is already being actively exploited in the wild.

That caused Google to declare the bug “particularly serious”, and give Microsoft just seven days to fix it before it would go public and tell the world about the error. The rationale for such disclosure programmes is part preventative and part punitive: it allows others using unpatched versions of Windows to be aware of the danger, even if they cannot fix it until Microsoft releases a patch, but it also serves to publicly berate the developer for their delay in fixing the bug.
Vulnerability Lifecycle – Worst Case: Zero Day

 Nobody knows  
  Vendor knows  
  Public knows  
  Fix

Extreme Danger
Only Black Hats know
Black Hats find out
White Hats know
Danger

Vendor knows
Public knows
Fix

Nobody knows

Prof. Eric Bodden - Build It, Break It, Fix It SS 17
Vulnerabilities

- No non-trivial system is completely free of vulnerabilities

- **Common Vulnerabilities and Exposures (~ 80k)**
  - Managed and hosted by MITRE
  - [https://cve.mitre.org/](https://cve.mitre.org/)
  - Each known vulnerability is assigned an identifier
  - E.g., CVE-2011-1153
How Bad is Bad?

- We’ve seen many vulnerabilities
  - Many of them *can* do catastrophic things
  - Danger really “depends on the situation”

- Many, many situational factors, such as:
  - Asset exposed, and its relative importance
  - Remotely, or locally exploitable?
  - Expertise needed to exploit the vulnerability?
  - Affects all deployments?
  - Impact on CIA properties
  - How much traction did the problem have already?
Rating Vulnerabilities vs. Risk Management

- Similar to one another, you have to rate found vulnerabilities
- How crucial is the vulnerability?

Risk Management
- Starts in early development phases, e.g., design
- Based on potential threats to the system
- Goal: Prevent (important) vulnerabilities

Vulnerability Assessment
- Only applicable for existing systems
- Applied to concrete vulnerabilities and (in the best case) corresponding exploits
- Goal: Fix and prevent further (important) vulnerabilities

If risk management is used/updated throughout the lifecycle, it can also support vulnerability assessment.
Rating Vulnerabilities

- We need a method to rate discovered vulnerabilities
  - Should take all essential factors into account
  - Should be repeatable and deterministic (to a certain degree)
  - Should result in comparable results (order of importance)
  - Should be approved by experts / industry

→ **Common Vulnerability Scoring System (CVSS)**
CVSS
Common Vulnerability Scoring System

- An open scoring system from FIRST
  - FIRST: Forum for Incident Response & Security Teams
  - http://www.first.org/cvss
  - A group of researchers & practitioners

- Adopted by NIST
- CVSS added in CVE descriptions
  - NVD (NIST) provides CVSS scores for all CVE
- Mostly applied in industry: v2

- Provide a set of metrics, and corresponding values and weighting functions
Temporal Metrics
Change over time.

Base Metrics
Core aspects of the vulnerability.

Environmental Metrics
Your own organization’s priorities. May vary in different deployments.

https://www.first.org/cvss/specification-document
**CVSS**

**Metric Groups**

Typically done by vulnerability bulletin analysts, security product vendors, or application vendors.

To be done by “user” organisation.

Base Metric Group

- Exploitability metrics
  - Characteristics of *how* a given thing is vulnerable

- Impact metrics
  - Represent the consequence to the thing that suffers the impact

- Scope
  - Which parts of the system are affected?
Base

Attack Vector (AV)

- Through what entry gates can an attacker exploit the vulnerability?
- Metric Value:
  - (P) Physical
  - (L) Local only
  - (A) Adjacent network (e.g. wi-fi, local IP subnet)
  - (N) Network: fully remotely exploitable

- More than one level affected? Go with the worse one
- Client that opens stuff from an untrusted internet source? Go with Network (e.g. zip utility with a buffer overflow)

1. XSS in a webapp? (N)
2. Lack of SSL encryption on Facebook? (A)
Base
Attack Complexity (AC)

- How complex would the vulnerability be to exploit?
  - One step? e.g. buffer overflow
  - Multiple steps?
    e.g. convince an email user to download a sketchy attachment

- Metric value
  - (H) High: Specialized access conditions
    - e.g. overcoming advanced exploit mitigation techniques
    - e.g. man in the middle attack
  - (L) Low: no specialized conditions
    - e.g. default configuration
    - e.g. requires little skill to perform

Note: Low complexity is bad
Base
Privileges Required (PR)

- Level of privileges needed for exploit?

- Metric value
  - (H) privileges that provide significant control
    - (e.g. administrative)
  - (L) privileges that provide basic user capabilities
  - (N) No authorization needed

- In an authentication system itself? Go with (N)

1. Path traversal in photo upload for a Twitter client? (L)
2. Insecure PRNG for session IDs? (N)
Base
User Interaction (UI)

- a user, other than the attacker, participates in the exploit

- Metric value
  - (N): can be exploited without interaction from any user
  - (R): requires a user to take action

  E.g. exploit may only be possible during the installation of an application by a system administrator.

Exploitability

1. Reflected XSS? (R) – must click on a link
2. CSRF? (R) – need victim to create the http request & be logged in
Base CIA Impact

- Any impact on
  - confidentiality, integrity, and/or availability?
  - These are three separate metrics

- Metric Value (for each metric)
  - (N) None
  - (L) Low
    - e.g. disclosing a few database tables
    - e.g. temporary DoS
  - (H) High
    - e.g. reading arbitrary memory locations is High confidentiality impact
    - e.g. full bypass of plug-in sandbox is High integrity impact
    - e.g. root-level access? High on all three metrics

Hardcoded root credentials in blogging software?

C = High | I = High | A = None
The ability for a vulnerability in one software component to impact resources beyond its means, or privileges.

Metric Value
- (U): Unchanged
  The vulnerable component and the impacted component are the same.
- (C): Changed
  The vulnerable component and the impacted component are different.

1. Vulnerability in a Linux VM that compromises the host OS?  
   (C)
2. Using crafted office file to cause a DoS in office suite?  
   (U)
Temporal
Exploitability (E)

- Is there a public exploit known?

Metric Value

- (U) Unproven, entirely theoretical exploit
- (POC) Proof-of-concept exists out there, no known maliciously used exploits
- (F) Functional exploit is available
- (H) Functional Exploit is widely disseminated
- (X) Not defined (skip this part of the metric)

Notes

- Being temporal, this could change very quickly
- Many white-hats will write exploits to make this score go up so that it becomes fixed
Temporal Remediation Level

- What is the level of remediation (RL)?
  - How has the vendor reacted?
  - Levels
    - (O) Official Fix is available
    - (TF) Temporary fix is available
    - (W) Workaround is available.
      - Unofficial, non-vendor patches,
      - Temporary change in configuration
    - (U) Nothing is released yet
    - (X) Not defined
Temporal Report Confidence

- What is the report confidence (RC)?
  - (U) Unconfirmed by the source, or there are multiple conflicting reports
  - (R) Significant details are published e.g. proof-of-concept
  - (C) Confirmed by the source
  - (ND) Not defined
Modified Base Metrics

- Metrics according to modifications that exist within the environment
  - Loss of life, physical assets, productivity

- Levels:
  - None
  - Low
  - Medium
  - High
  - Not defined
Environmental Security Requirements

- Security Requirements
  - Confidentiality Requirement (CR)
  - Integrity Requirement (IR)
  - Availability Requirement (AR)
- Reweighting the modified CIA impact metrics
- Metric Value:
  - (X): Not defined. It will not influence the score.
  - (H): A catastrophic adverse effect
  - (M): A serious adverse effect
  - (L): A limited adverse effect
Scoring Hints

- Ignore interactions with other vulnerabilities, score each individually
- Assume the most common or default configuration of the server
- Score the greatest exploitation impact, if there are many
What’s the problem here?

mTimeToSampleCount = U32_AT(&header[4]);

mTimeToSample = new uint32_t[mTimeToSampleCount * 2];
size_t size = sizeof(uint32_t) * mTimeToSampleCount * 2

Integer Overflow!
Fix attempt

```c
mTimeToSampleCount = U32_AT(&header[4]);
uint64_t allocSize = mTimeToSampleCount * 2 * sizeof(uint32_t); // Overflow!
if (allocSize > SIZE_MAX) { Always false!
    return ERROR_OUT_OF_RANGE;
}
```

```
Still Overflow!
mTimeToSample = new uint32_t[mTimeToSampleCount * 2];
size_t size = sizeof(uint32_t) * mTimeToSampleCount * 2
```

- This mistake made 95% of all Android devices vulnerable against remote code execution! (CVE-2015-1538)
- How should fixing actually be done?
Change Management

- Goal: apply changes in an efficient and controlled way, while minimizing risks of negative operational impact
- Industry best practice and widely in place
Fixes Are Changes

- … and need to be coordinated through Change Management as well!

- Problem:

  Vulnerability present vs Change Management Overhead

- Allow prioritization for vulnerability fixes in Change Management process!
Fix Implementation

- Even “trivial” fixes may have side effects...

  ```c
  if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
  goto fail;
  if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
  err = sslRawVerify(ctx, 
                     ctx->peerPubKey,  
                     dataToSign,       
                     dataToSignLen,    
                     signature,        
                     signatureLen);    
  ```

- Fixes can also become very complex!
- Ensure that...
  - ... the fix solves the issue in itself, not just a particular exploit
  - ... the fix does not break anything else
- Have your fixes reviewed and tested properly

sslRawVerify() not dead anymore! New vulnerability?
Recommendations regarding Updates

- Software without support is a time bomb that is going to explode sooner or later!

- As a customer:
  - Demand and use updates
  - Accept to pay for them
  - Accept that software reaches end of life eventually

- As a vendor:
  - Provide support for a decent amount of time, at least if customers ask
  - Consider support costs in your pricing

- Provide usable and secure means for your customers to obtain and install updates
  - All security considerations in your software are useless if you carelessly use HTTP for providing updates…

- Allow automated and manual updates
Patch Management

- Patches/Updates should not always blindly be installed
  - What if a patch breaks some critical business application?
  - What if a patch fails on a special configuration the vendor did not consider during testing?
- Organizations should have Application Management, Patch Management, Vulnerability Management to clarify …
  - Which applications are used in the organization?
  - Which vulnerabilities are relevant?
  - How can and should they be dealt with?
  - What is the impact when the update is installed? What is the impact when the update is not installed?
- Don’t forget about Patch Management for your private systems!
Conclusion

- Fixing Vulnerabilities takes a lot of time, is expensive and tedious
- Yet is it required, since bugs and vulnerabilities are present
- Change Management keeps modifications controlled

Try to avoid vulnerabilities as early as possible!
Fix It Phase

- Get points back by fixing issues found during Break It


- Fix exactly one bug per fix submission! (multiple Break submissions might target the same bug/vulnerability)

- If you believe a break targeting your team is incorrect, you can dispute by sending a mail to manuel.benz@upb.de and martin.mory@upb.de

- We will judge your fixes for correctness, i.e., fix submissions target exactly the core issue revealed by the corresponding break submission