PatchDB:
A Large-Scale Security Patch Dataset

Xinda Wang*, Shu Wang*, Pengbin Feng, Kun Sun, Sushil Jajodia

Center for Secure Information Systems, George Mason University

* The first two authors contributed equally to this work.

Background & Motivation

A security patch embeds both vulnerable code and corresponding fix.

- ➢ Vulnerability detection
- ➢ Patch presence testing

Existing open-source patch datasets have several limitations:

- Small: collected from one or few projects
- Biased: collected from specific type of projects
- Noisy: security patches labelled as non-security ones

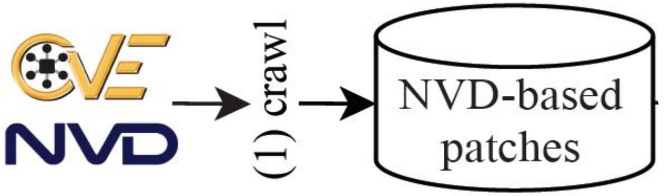
Our Work

To enable patch/vulnerability related research, we construct a new patch dataset called *PatchDB* consisting of three parts:

Our Work

To enable patch/vulnerability related research, we construct a new patch dataset called *PatchDB* consisting of three parts:

- Part I: NVD-based dataset

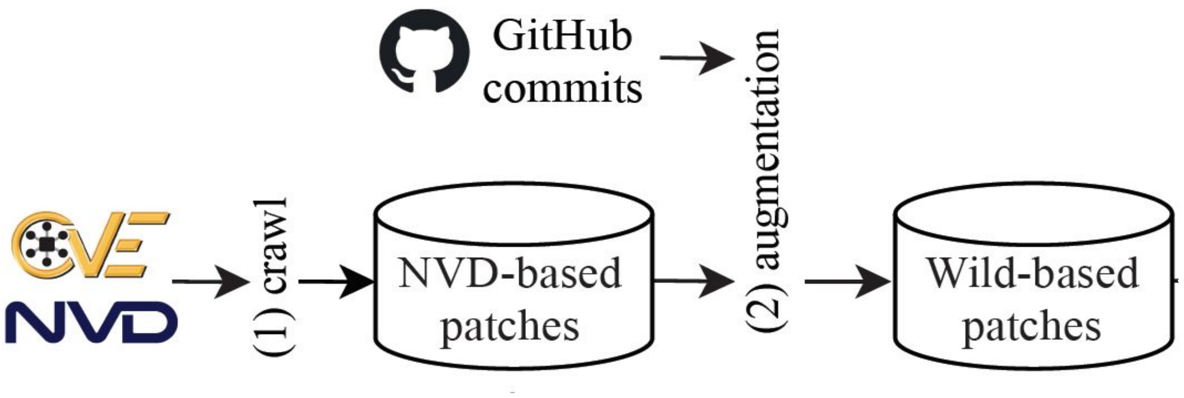


Overview of PatchDB construction.

Our Work

To enable patch/vulnerability related research, we construct a new patch dataset called *PatchDB* consisting of three parts:

- Part I: NVD-based dataset
- Part II: Wild-based dataset

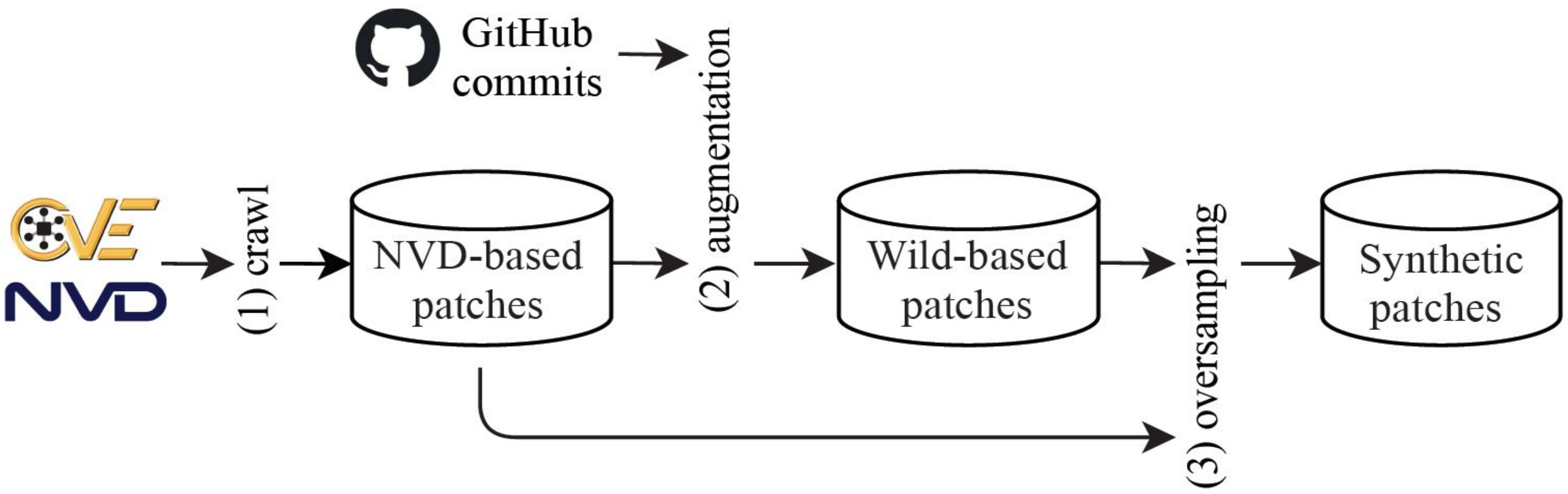


Overview of PatchDB construction.

Our Work

To enable patch/vulnerability related research, we construct a new patch dataset called *PatchDB* consisting of three parts:

- Part I: NVD-based dataset
- Part II: Wild-based dataset
- Part III: Synthetic dataset



Overview of PatchDB construction.

Part I: Extracting Security Patches from NVD

For each CVE entry, we download its security patch from the Git hyperlink labelled as “Patch” and collect 4K samples.

Hyperlink	Resource
http://www.securityfocus.com/bid/93271	Third Party Advisory, VDB Entry
https://github.com/ImageMagick/ImageMagick/commit/90406972f108c4da71f998601b06abdc2ac6f06e	Patch, Vendor Advisory

CVE-2016-7906 Detail

```
From 90406972f108c4da71f998601b06abdc2ac6f06e Mon Sep 17 00:00:00 2001
From: Cristy <urban-warrior@imagemagick.org>
Date: Sat, 1 Oct 2016 11:18:08 -0400
Subject: [PATCH] https://github.com/ImageMagick/ImageMagick/issues/281

---
 MagickCore/attribute.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/MagickCore/attribute.c b/MagickCore/attribute.c
index d9f088ad75..f6510b7bf3 100644
--- a/MagickCore/attribute.c
+++ b/MagickCore/attribute.c
@@ -1264,7 +1264,7 @@ MagickExport MagickBooleanType SetImageType(Image
*image,const ImageType type,
           status=QuantizeImage(quantize_info,image,exception);
           quantize_info=DestroyQuantizeInfo(quantize_info);
         }
-       image->colors=2;
+       status=AcquireImageColormap(image,2,exception);
        image->alpha_trait=UndefinedPixelTrait;
        break;
     }
```

Part II: Augmenting via Nearest Link Search

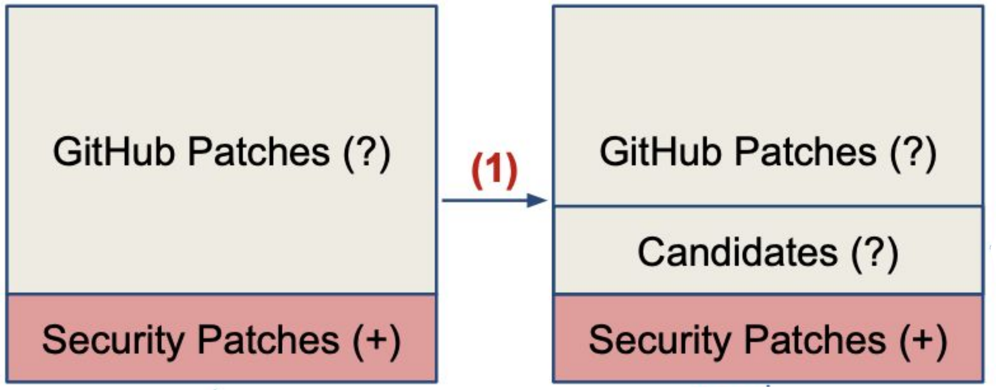
Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.

Part II: Augmenting via Nearest Link Search

Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.

Part II: Augmenting via Nearest Link Search

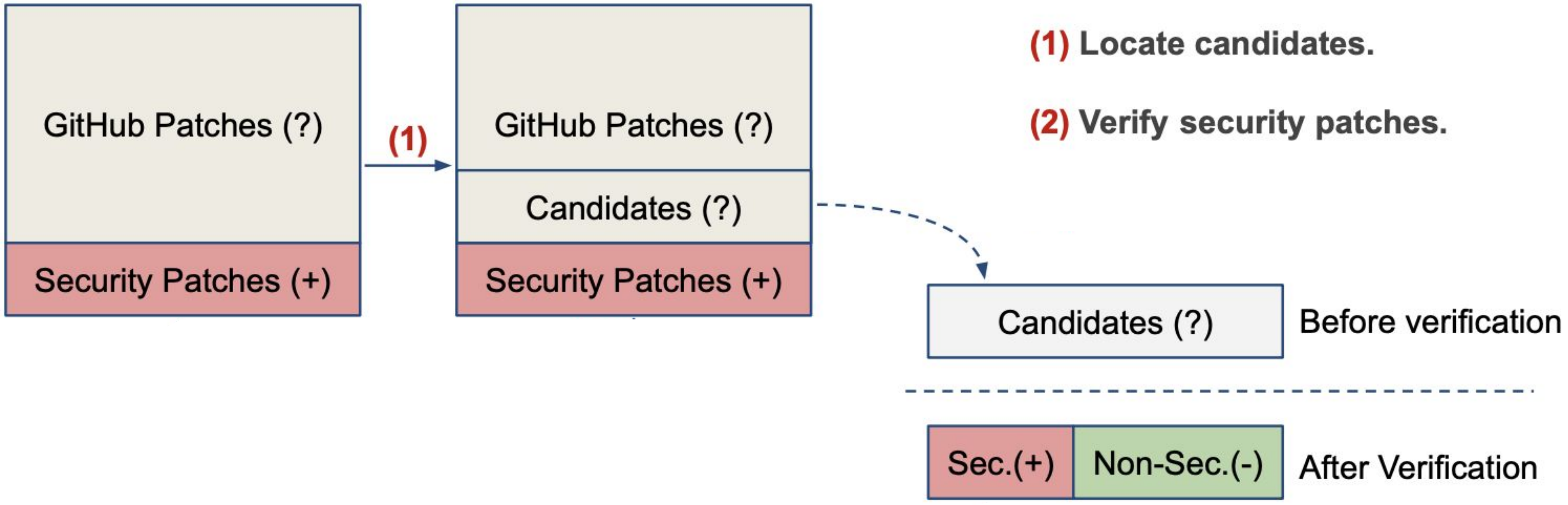
Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.



(1) **Locate candidates.**

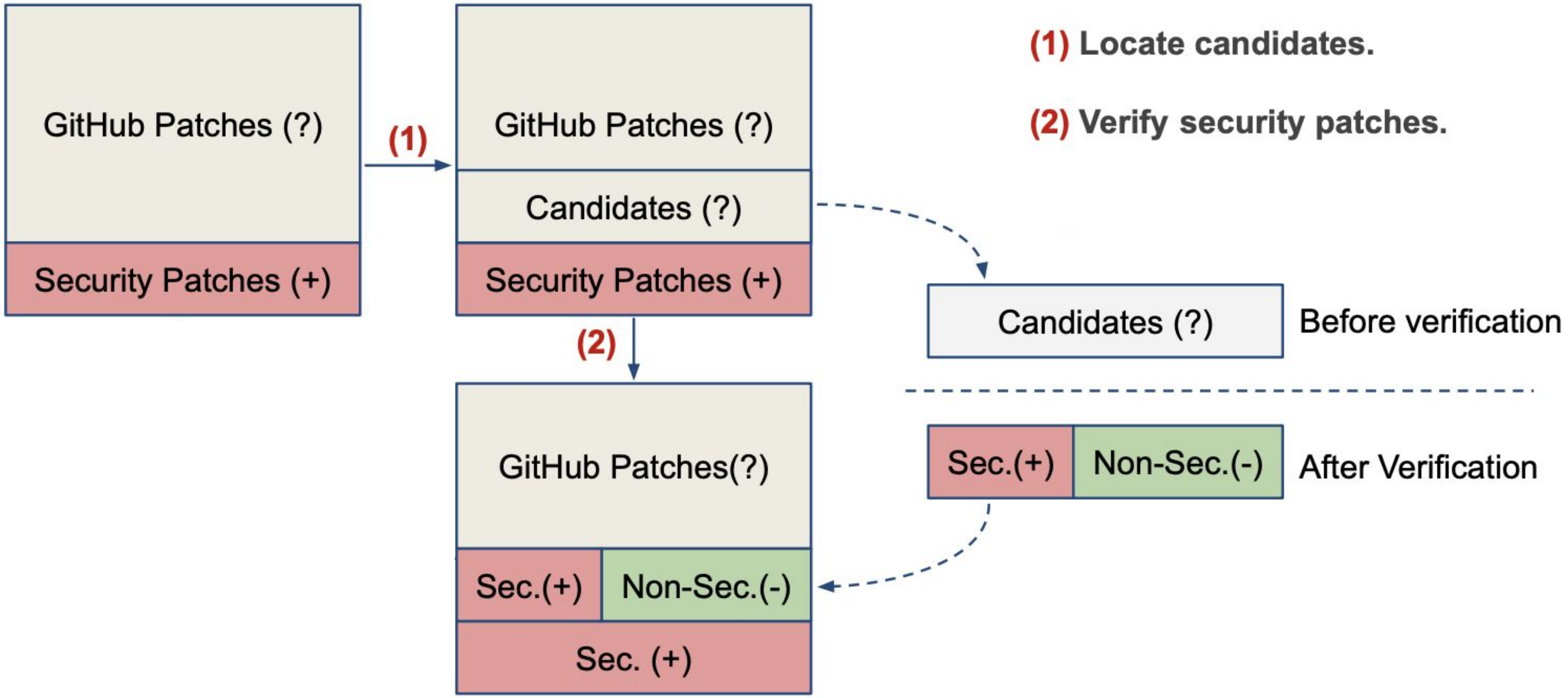
Part II: Augmenting via Nearest Link Search

Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.



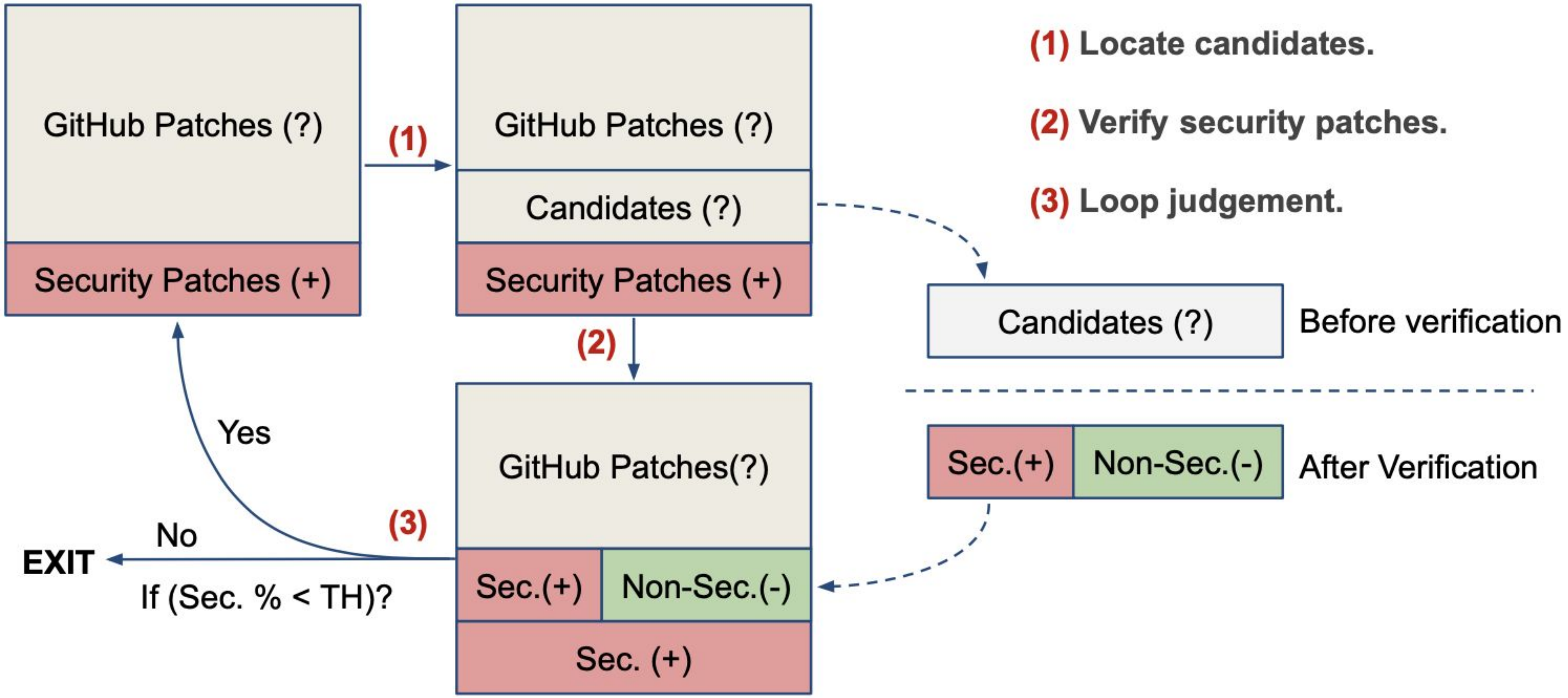
Part II: Augmenting via Nearest Link Search

Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.



Part II: Augmenting via Nearest Link Search

Rationale: 8% GitHub commits are security patches without a CVE-ID, providing a source for augmenting security patch dataset.



Nearest Link Search

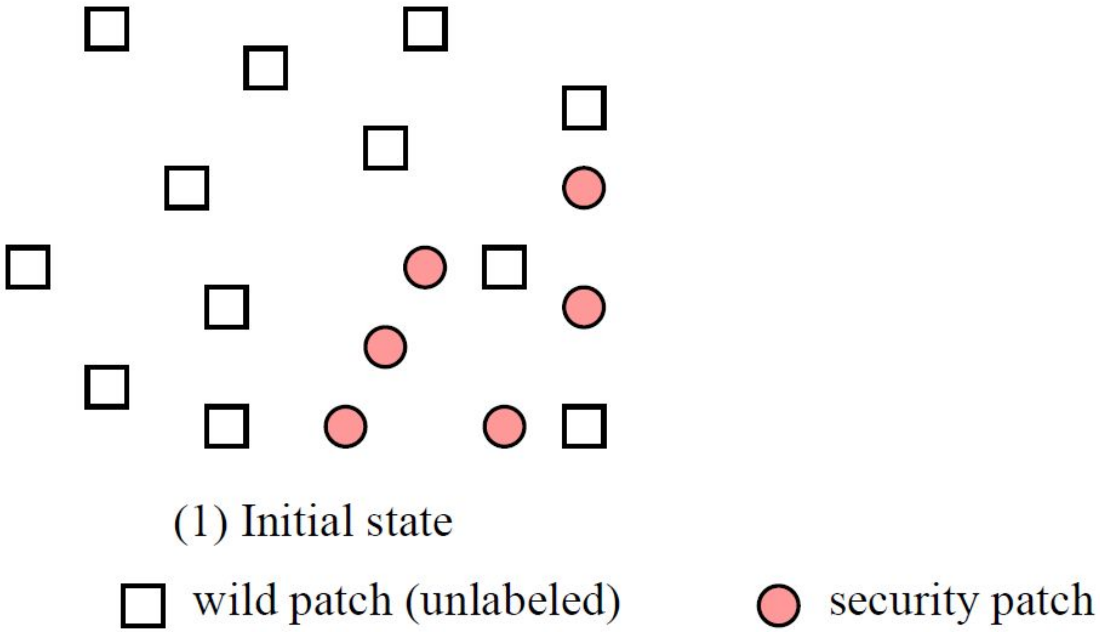
Goal: to locate the most promising candidates.

Approach: for each sample in existing security patch dataset, we search and verify its nearest neighbor from the wild (i.e., GitHub).

Nearest Link Search

Goal: to locate the most promising candidates.

Approach: for each sample in existing security patch dataset, we search and verify its nearest neighbor from the wild (i.e., GitHub).

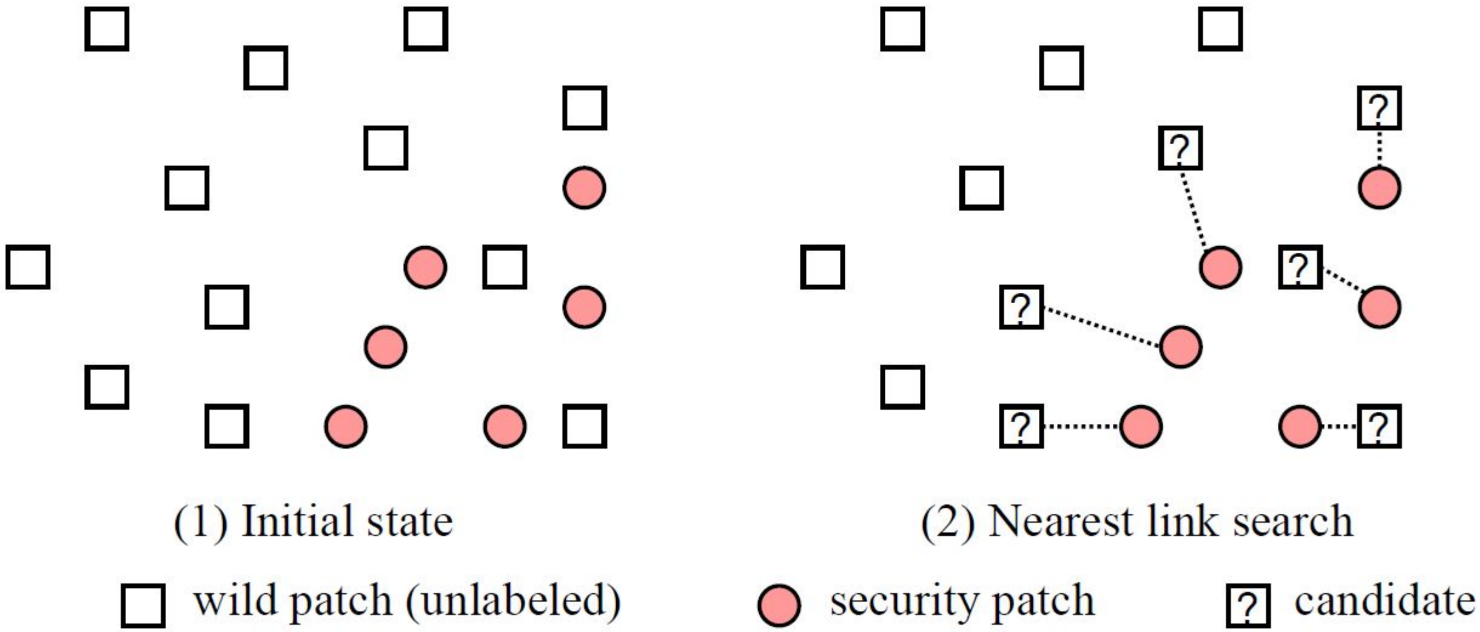


(1) Initial state

Nearest Link Search

Goal: to locate the most promising candidates.

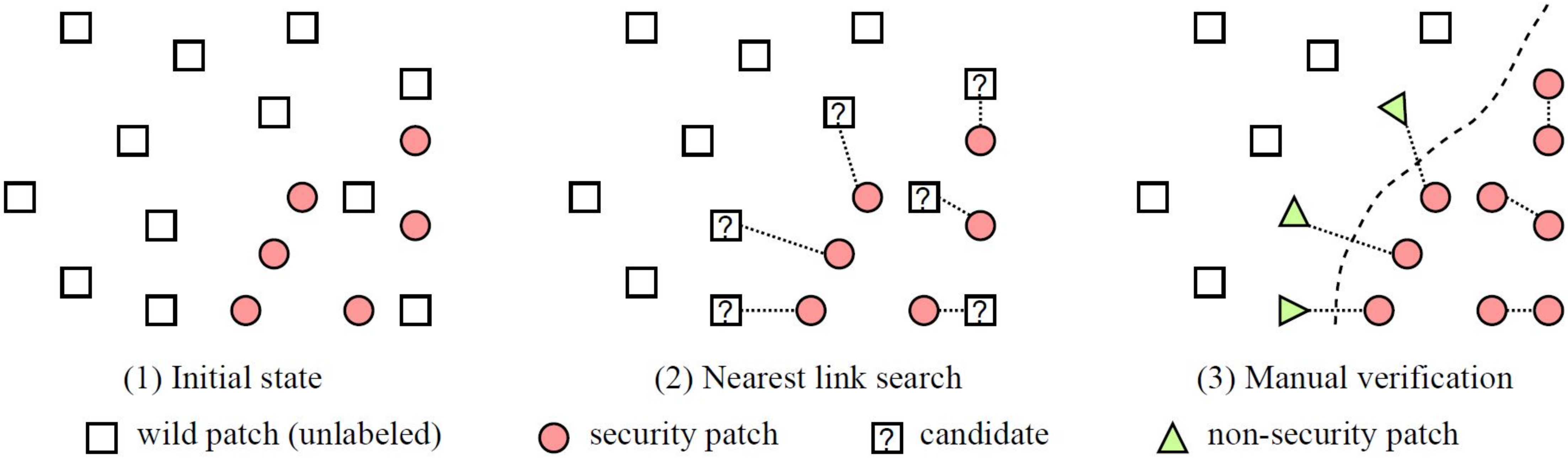
Approach: for each sample in existing security patch dataset, we search and verify its nearest neighbor from the wild (i.e., GitHub).



Nearest Link Search

Goal: to locate the most promising candidates.

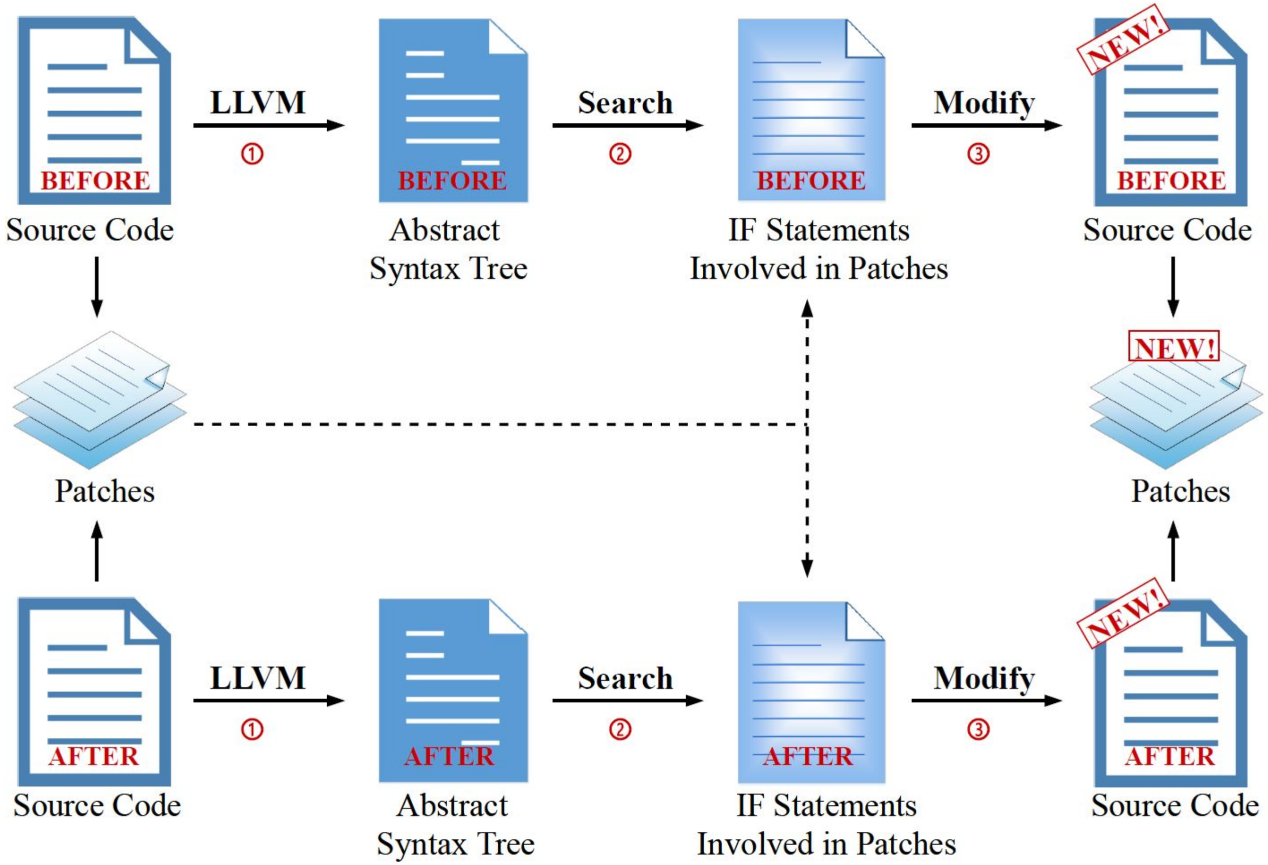
Approach: for each sample in existing security patch dataset, we search and verify its nearest neighbor from the wild (i.e., GitHub).



Part III: Synthesizing Artificial Patches

Rationale: around 70% security patches add/update sanity checks [1].

Strategy: adding variances on IF statements



[1] Zhao, Lei, et al. "Patchscope: Memory object centric patch diffing." *CCS* 2020.

Evaluation

We aim to answer five questions:

1. How to construct the wild-based security patch dataset using the nearest link search approach?
2. What is the performance of the nearest link search compared with existing methods?
3. Can synthetic security patches really help?
4. What is the composition of our PatchDB?
5. What is the quality of PatchDB?

Q1: Wild-Based Dataset Construction

Search range: 200K randomly selected commits from 300+ popular C/C++ GitHub projects

Ratio of security patches: up to ~30% after verification

Observation: ratio increases along with a larger search range

Results: 8K security and 24K non-security samples

Q2: Effectiveness of Nearest Link Search

Our nearest link search outperforms other three augmentation methods:

- *Brute force search*: directly screening security patches from the wild.
- *Pseudo labeling*: locating candidates from prediction results of single machine learning model (Random Forest) with the highest confidence.
- *Uncertainty-based labeling*: locating candidates from prediction results of multiple machine learning classifiers with the highest certainty (i.e., consensus)

Methods	% of Security Patches
Brute Force Search	8%
Pseudo Labeling	13%
Uncertainty-Based Labeling	12%
Nearest Link Search (Ours)	**29%**

Q3: Effectiveness of Synthesized Patches

Synthesizing patches is effective in the security patch identification task with a small dataset (i.e., the NVD-based dataset).

Performance w/o or w/ synthetic patches.

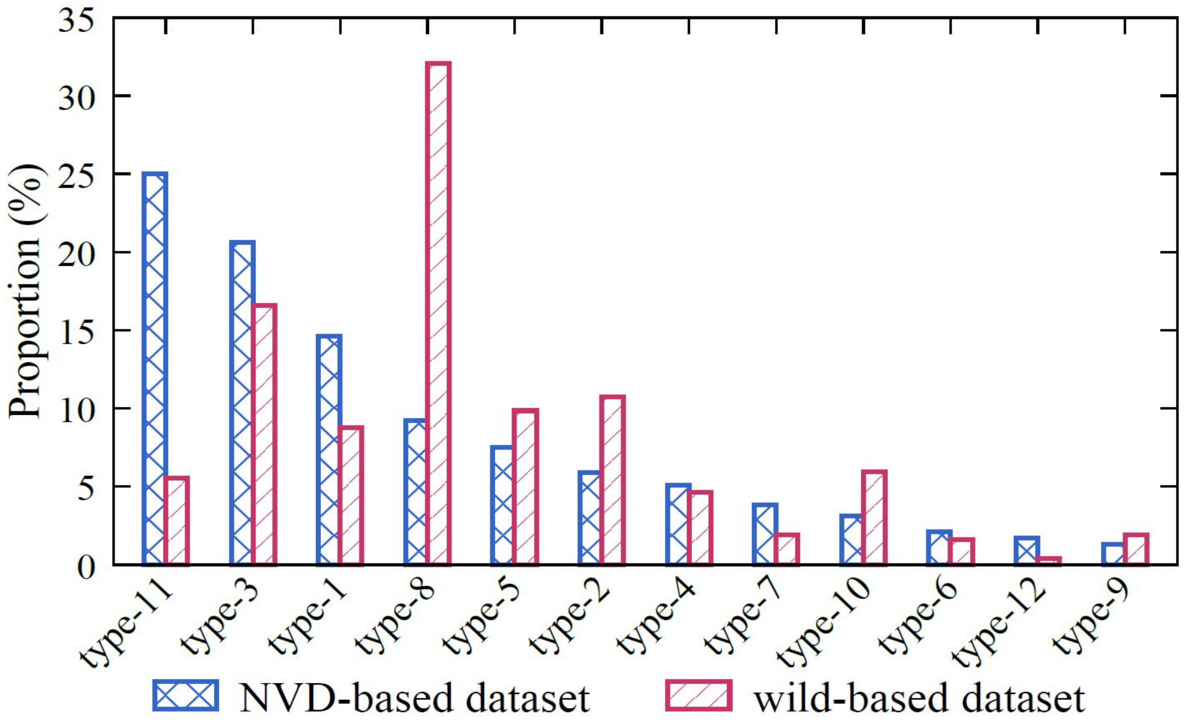
Dataset	Synthetic Dataset	Precision	Recall
NVD	-	82.1%	84.8%
NVD	17K Sec. + 20K NonSec.	86.0% **(+3.9%)**	87.2% **(+2.4%)**
NVD+Wild	-	92.9%	61.1%
NVD+Wild	58K Sec. + 129K NonSec.	93.0% (+0.1%)	61.2% (+0.1%)

Sec. = security patch; NonSec. = non-security patch

Q4: Distribution after Augmentation

We observe dissimilar distribution between wild-based dataset identified by the nearest link search and NVD-based dataset.

➢ Benefit: introduce more varieties



ID	Type of patch pattern
1	add or change bound checks
2	add or change null checks
3	add or change other sanity checks
4	change variable definitions
5	change variable values
6	change function declarations
7	change function parameters
8	add or change function calls
9	add or change jump statements
10	move statements without modification
11	add or change functions (redesign)
12	others

Q5: Performance Improvement using PatchDB

In the task of automatic security patch identification, models trained with both the NVD-based dataset and the wild-based dataset have *better generalization ability*.

Impacts of datasets over learning-based models.

Training Dataset	Algorithm	Test Dataset	Precision	Recall
NVD	Random Forest	NVD	58.4%	21.7%
		Wild	58.0%	19.5%
	RNN	NVD	82.8%	83.2%
		Wild	88.3%	24.2%
NVD+Wild	Random Forest	NVD	90.1%	22.5%
		Wild	91.8%	44.6%
	RNN	NVD	92.8%	60.2%
		Wild	92.3%	63.2%

Conclusion

- We present the PatchDB:
  - ➢ a *large-scale* dataset that contains 12K security patches
  - ➢ cover *various* types in terms of code changes
  - ➢ contain a *cleaned* non-security patch dataset of 23K samples
  - ➢ provide a *synthetic* dataset generated from real-world samples
- Use cases:
  - ➢ Detecting vulnerability/patch presence
  - ➢ Automatically generating patches
  - ➢ Compiling to binary dataset

Thank you!

Authors:

Xinda Wang, **Shu Wang**, Pengbin Feng, Kun Sun, Sushil Jajodia

Questions?

Our Emails: xwang44@gmu.edu, swang47@gmu.edu

Dataset can be accessed at:

https://github.com/SunLab-GMU/PatchDB