QRS 2023

RUSPATCH: Towards Timely and Effectively Patching Rust Applications

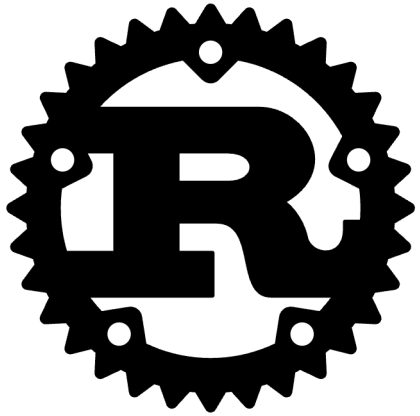
Yufei Wu Baojian Hua*

University of Science and Technology of China

Computer Systems and Security Group

Rust for safe system programming

- Robust type system
- Ownership model
- Strict security checks at compile time



aws Azure Google Cloud

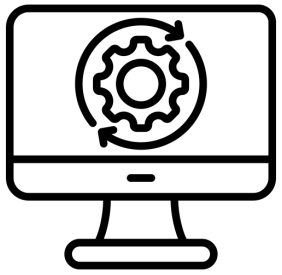
Unsafe Rust: an escape hatch

- low-level system programming
- programming flexibility
- bypass static analysis

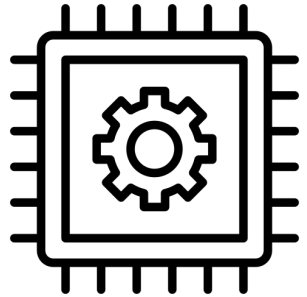
```
fn from(buffer: Buffer) -> Vec<u8> {
    let mut slice =
        Buffer::allocate(buffer.len);
    let len = buffer.copy_to(&mut slice);
    unsafe {
  Vec::from_raw_parts(slice.as_mut_ptr(),
        len, slice.len())
    }
}
```

use-after-free is triggered

Memory-mapped I/O



OS Interaction

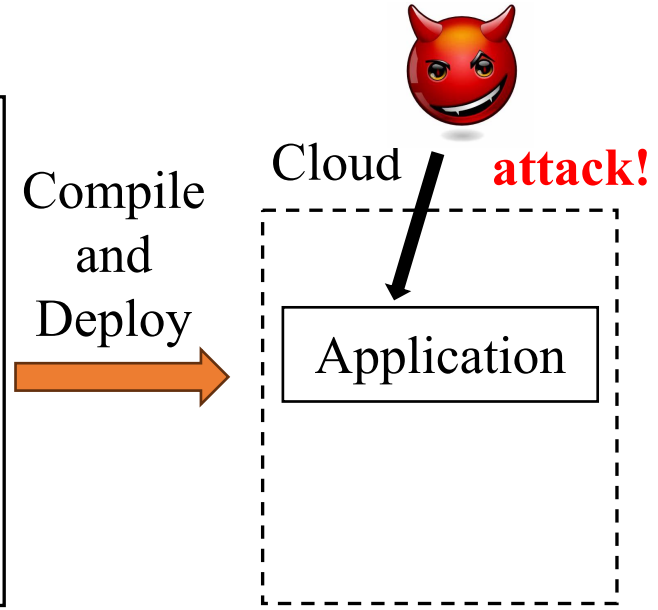


Hardware Abstraction

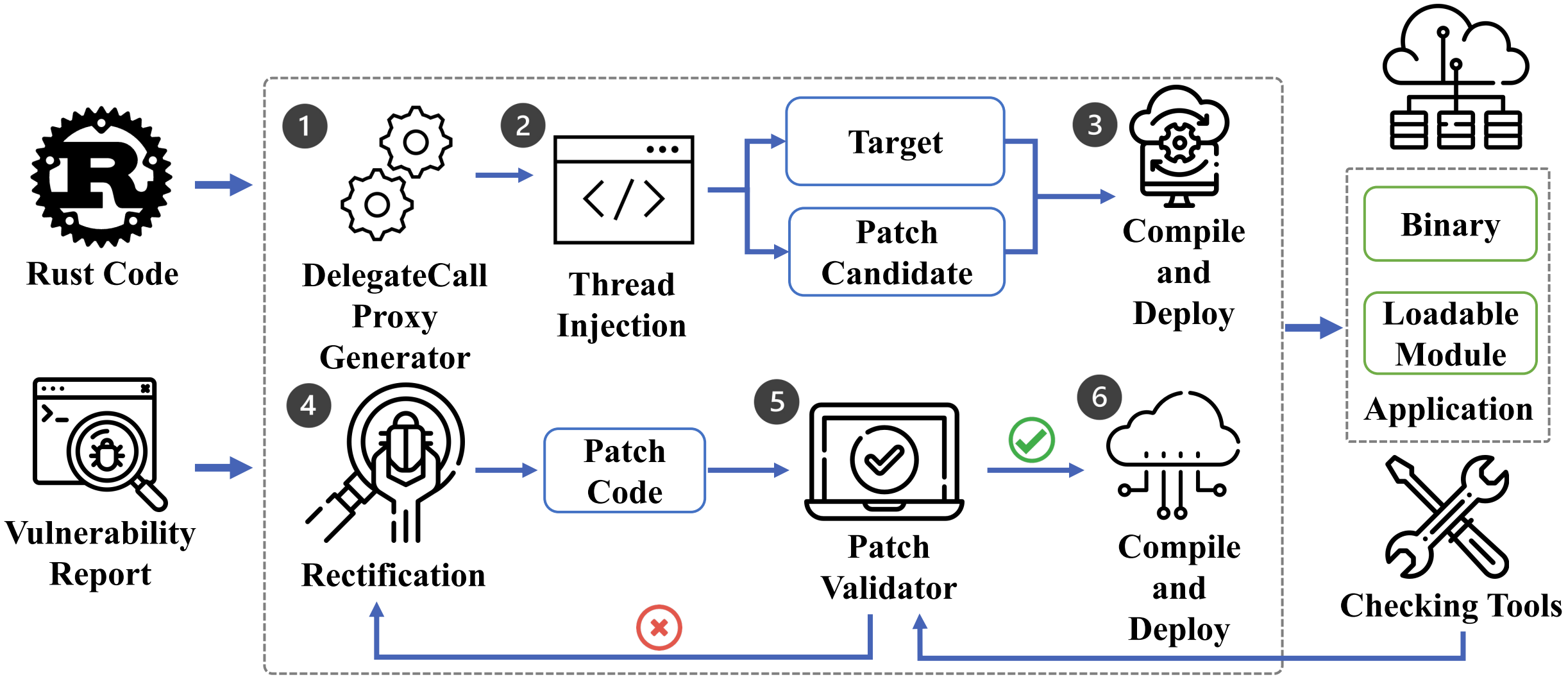
Motivation

- High availability of Cloud service
- Dynamic Software Updating (DSU) in Rust

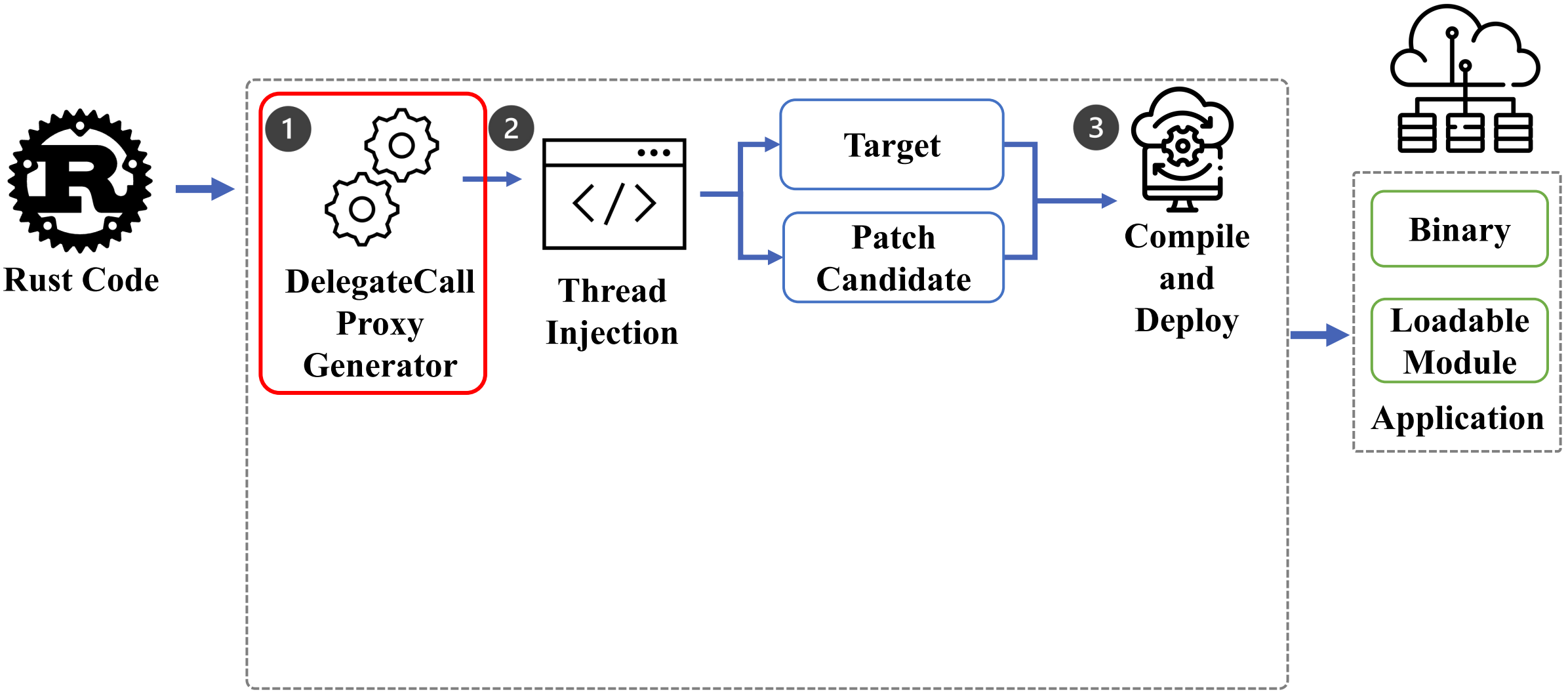
```
fn from(buffer: Buffer) -> Vec<u8> {
    let mut slice =
        Buffer::allocate(buffer.len);
    let len = buffer.copy_to(&mut slice);
    unsafe {
  Vec::from_raw_parts(slice.as_mut_ptr(),
        len, slice.len())
    }
}
```



RUSPATCH


Rust Code
Vulnerability Report
1
DelegateCall Proxy Generator
2
Thread Injection
Target
Patch Candidate
3
Compile and Deploy
4
Rectification
Patch Code
5
Patch Validator
6
Compile and Deploy
Binary
Loadable Module
Application
Checking Tools

RUSPATCH's Design



DelegateCall Proxy Generator

- Patch Candidate Analysis
  1. identify all unsafe functions and their corresponding call sites
  2. rewrite all such call sites to use delegatecall proxy patterns
- Translation Functions
  - To traverse and rewrite Rust AST

$$[\![[\texttt{unsafe}]\ y(\tau\ x)\{\vec{s}\ t\}]\!]_f \Rightarrow [\texttt{unsafe}]\ y(\tau\ x)\{[\![\vec{s}]\!]_s\ [\![t]\!]_t\}$$

$$[\![p = r]\!]_s \Rightarrow p = r$$

$$[\![\texttt{storageLive}(x)]\!]_s \Rightarrow \texttt{storageLive}(x)$$

$$[\![\texttt{storageDead}(x)]\!]_s \Rightarrow \texttt{storageDead}(x)$$

$$[\![f(x)]\!]_t \Rightarrow \begin{cases} f' = \texttt{findSym}(\text{“}f\text{”}); f'(x) & f \in U \\ f(x) & f \notin U \end{cases}$$

$$[\![\texttt{drop}(p)]\!]_t \Rightarrow \texttt{drop}(p)$$

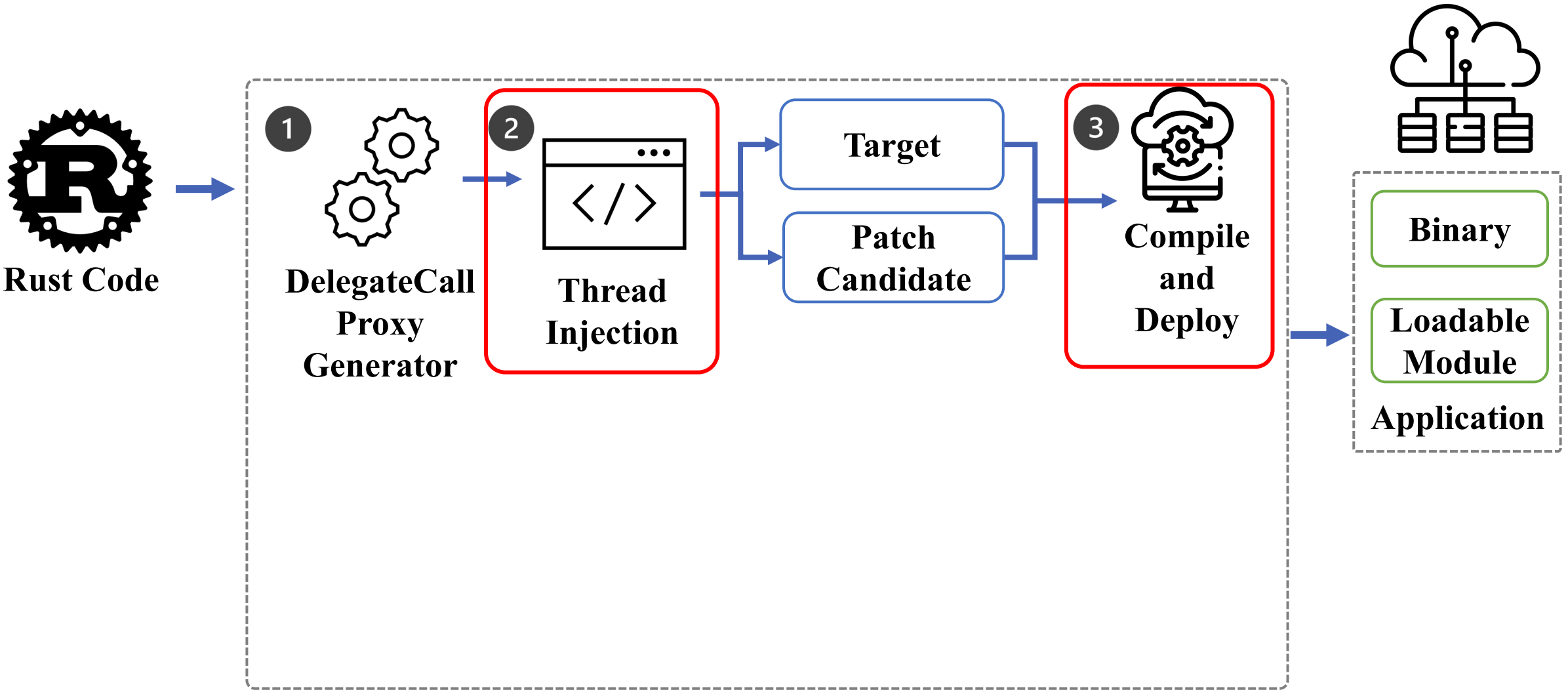
$$[\![\texttt{goto}(b)]\!]_t \Rightarrow \texttt{goto}(b)$$

Algorithm 1 : Delegatecall proxy pattern generation

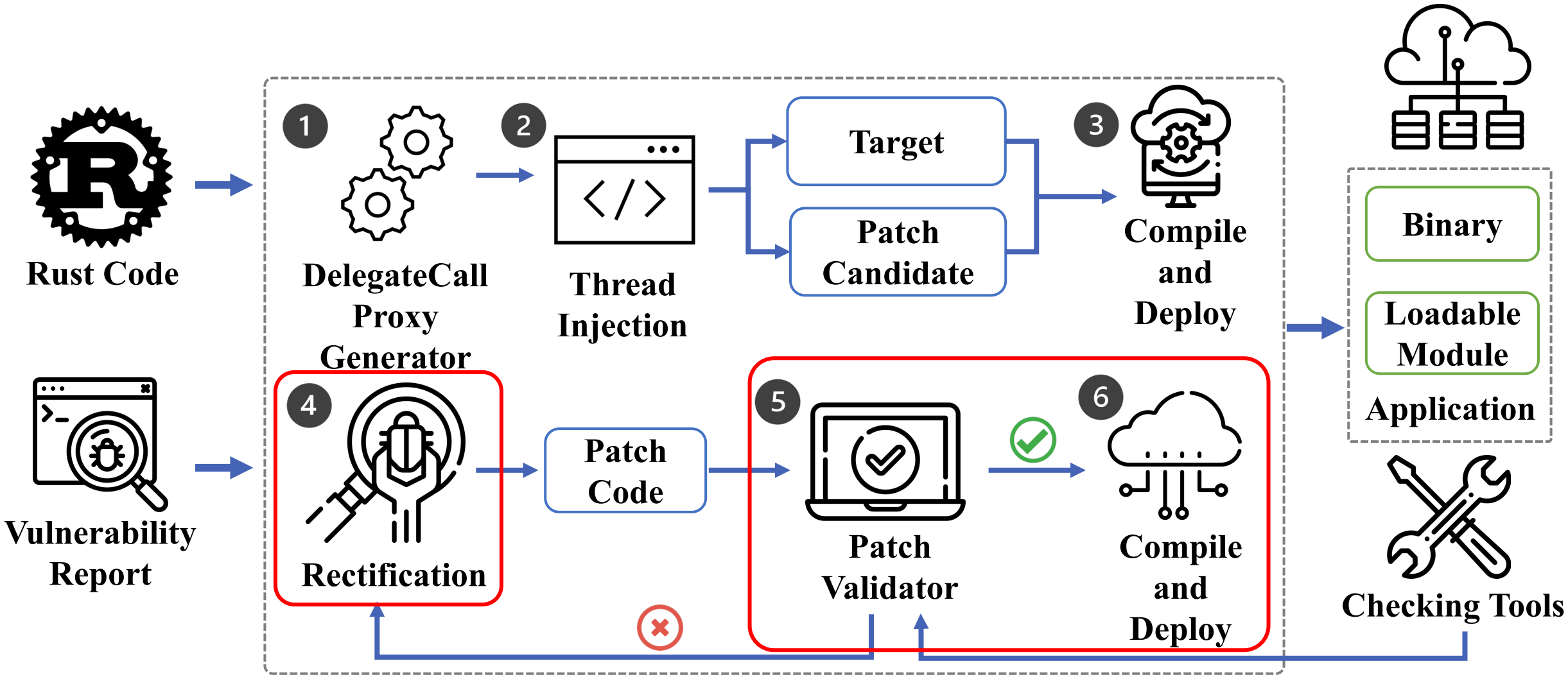
Input: P: The Rust program
Output: (P', U): P' is Delegatecall proxy injected Rust program; and U is a set of patch candidates

```
1:  procedure GEN-DELEGATECALL(P)
2:      P' = P
3:      U, C = IDENTIFY-PATCH-CANDIDATE(P)
4:      for each function f ∈ U do
5:          P' −= f
6:      for each call site h ∈ C do
7:          P' = ADDDELEGATECALL(P', h)
8:      return P', U
9:  procedure IDENTIFY-PATCH-CANDIDATE(P)
10:     U, C = ∅
11:     A = BUILDAST(P)
12:     G = BUILDCALLGRAPH(P)
13:     for each unsafe function f ∈ A do
14:         U ∪= f
15:         for each call graph edge (h → f) ∈ G do
16:             C ∪= h
17:     return U, C
```

RUSPATCH's Design



RUSPATCH's Design



Evaluation

- **Research Questions**
  - Effectiveness
  - Compiling performance
  - Runtime overhead
  - Usefulness
- **Dataset**
  - **RUSBENCH:** 10 CVEs and program bugs
  - 5 real-world applications
- **Environment**
  - CloudLab c220g5 server

Evaluation: Effectiveness

Step 1. compiled and executed all the benchmarks, triggered a crash
Step 2. processed them with RUSPATCH, then compiled and executed them for a second time, patched them during execution

Project	Type	Patched?
CVE-2017-1000430	OOB	✓
CVE-2018-1000810	OOB	✓
CVE-2019-15551	DF	✓
CVE-2019-16140	UAF	✓
CVE-2019-16880	DF	✓
CVE-2020-25794	UNINIT	✓
Servo-1	UAF	✓
Servo-4	UNINIT	✓
Tock	UNINIT	✓
Redox	UNINIT	✓

Evaluation: Compiling performance

Calculate the average compilation time of ten times and compare with rustc

Project	Type	Original (s)	RUSPATCH (s)
CVE-2017-1000430	OOB	0.538	0.582
CVE-2018-1000810	OOB	0.553	0.559
CVE-2019-15551	DF	0.844	0.897
CVE-2019-16140	UAF	0.522	0.557
CVE-2019-16880	DF	0.838	0.906
CVE-2020-25794	UNINIT	0.596	0.611
Servo-1	UAF	0.591	0.593
Servo-4	UNINIT	0.543	0.572
Tock	UNINIT	0.876	0.882
Redox	UNINIT	0.552	0.563

Evaluation: Runtime overhead

- Three *unsafe* usage scenarios
  1. unsafe memory access `slice::get_unchecked()`
  2. traversing an array by raw pointers `ptr::offset()`
  3. unsafe memory copy `ptr::copy_nonoverlapping()`
- $Overhead = \frac{RUSPATCH}{Original} - 1$
- The overhead is less than 3.28%

Operations	Original (ms)	RUSPATCH (ms)	Overhead
unchecked	1709.97 ± 3.19	1722.36 ± 14.05	+0.72%
offset	1424.64 ± 3.01	1434.14 ± 2.15	+0.67%
copy	252.61 ± 1.30	260.92 ± 0.58	+3.28%

Evaluation: Usefulness

- 6 Rust developers
- Task1: Converting to a DSU program
- Task2: Dynamic deployment of patches

CVE	Task1 (m)		Task2 (m)	
	Manual	RUSPATCH	Manual	RUSPATCH
CVE-2017-1000430	39.3	4.2	12.3	2.6
CVE-2019-16140	35.7	2.8	12.0	1.8
CVE-2019-16881	47.7	2.8	39.3	2.1
CVE-2020-25794	26.1	2.1	23.3	1.7

Evaluation: Real-world applications

- fault injection
- original *rustc* and RUSPATCH
- two version: prog.unsafe, prog.dsu

Program	Build Time (s)		Size (MB)	
	Original	RUSPATCH	Original	RUSPATCH
RisingWave	280.41	281.44	1809.31	1810.08
Polars	36.37	36.76	927.38	927.98
Wasmer	587.99	598.88	11.54	11.62
Diem	186.18	188.82	477.61	470.62
Rocket	43.85	46.08	111.69	112.58

Conclusions

We present ***RUSPATCH*** as a *first* step towards addressing Rust dynamic software updating problems

- Infrastructure design
- Prototype implementation
- Extensive evaluations

Thank you for your attention!

CONTACT

- Yufei Wu (wuyf21@mail.ustc.edu.cn)
- Baojian Hua (bjhua@ustc.edu.cn)

University of Science and Technology of China

Computer Systems and Security Group