# Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries

Ruian Duan, Ashish Bijlani, Yang Ji, Omar Alrawi, Yiyuan Xiong, Moses Ike, Brendan Saltaformaggio, Wenke Lee

Georgia Tech

NDSS 2019

# Background

- Open Source Software (OSS) is gaining popularity, e.g. GitHub reported 31M users and 100M repos
- App marketplace is growing quickly with over 2M apps on Play Store and Docker Hub
- App developers reuse OSS for many benefits, meanwhile, OSS **security flaws** are inherited

# Background

**Can OEM vendors or end users take any action?**

**4K+ Security Patches**

***From NVD***

**100K+ Vulnerable Apps**

***High risk only***

# Goal

- **OSSPatcher**: an automated system that fixes n-day OSS vulnerabilities in **app binaries** using publicly available **source patches**
- Prototype scope: fix vulnerable OSS written in C/C++ for Android apps
- Assumptions
  - App developers compile OSS directly from release versions
  - NVD is accurate, including vulnerable versions and patch commits of CVEs

## Challenges

- Source patch analysis: configurable OSS variants
- Source-to-binary matching: stripped binaries
- App patching: statically linked binaries

17% apps use non-default config

## OpenSSL: CVE-2014-3509, patch fb0bc2b

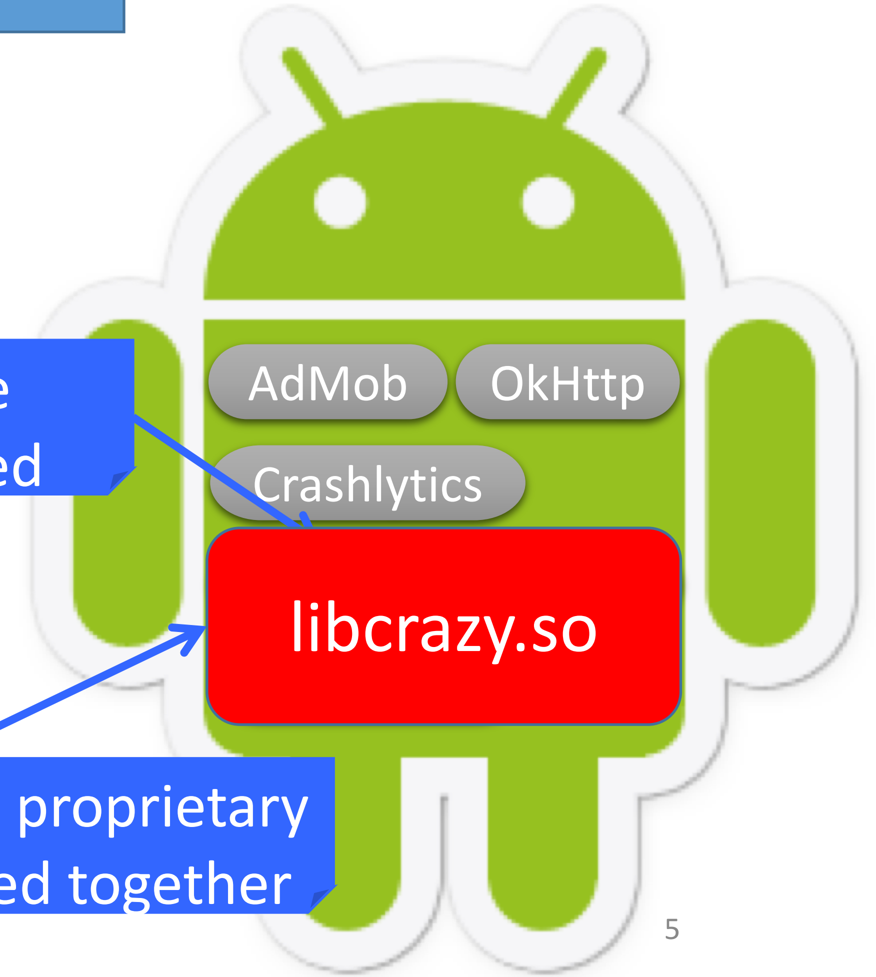
```
@@ static int ssl_scan_serverhello_tlsext(SSL *s,…
#ifndef OPENSSL_NO_EC
    *al = TLS1_AD_DECODE_ERROR;
    return 0;

-                       rmatlist_length = 0;
- if (s->session->tlsext_ecpointformatlist != …
-                       intformatlist = …
+
+  *al = TLS1_AD_INTERNAL_ERROR;
```

Function and variable symbols can be stripped

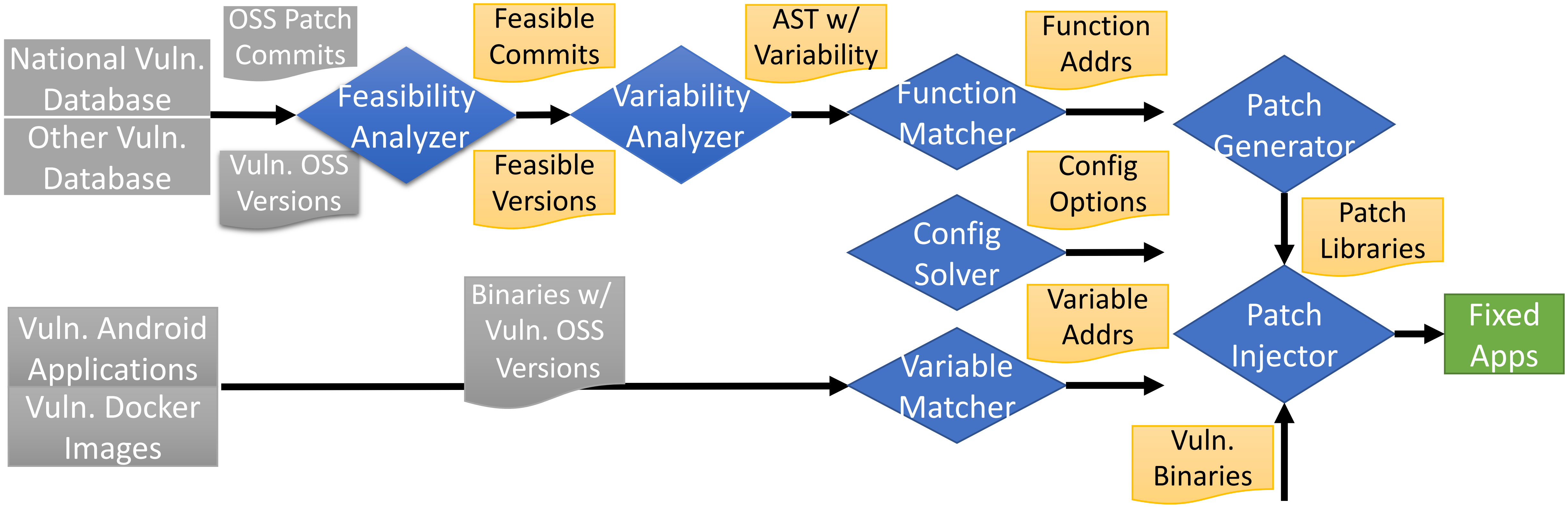
Conditional directives allow users to customize the final binary

OpenSSL 1.1.0h contains 160+ config options

Multiple OSS and proprietary code can be linked together



# Design


National Vuln. Database
Other Vuln. Database
OSS Patch Commits
Vuln. OSS Versions
Feasibility Analyzer
Feasible Commits
Feasible Versions
Variability Analyzer
AST w/ Variability
Function Matcher
Function Addrs
Config Solver
Config Options
Variable Matcher
Variable Addrs
Patch Generator
Patch Libraries
Patch Injector
Vuln. Binaries
Fixed Apps
Vuln. Android Applications
Vuln. Docker Images
Binaries w/ Vuln. OSS Versions

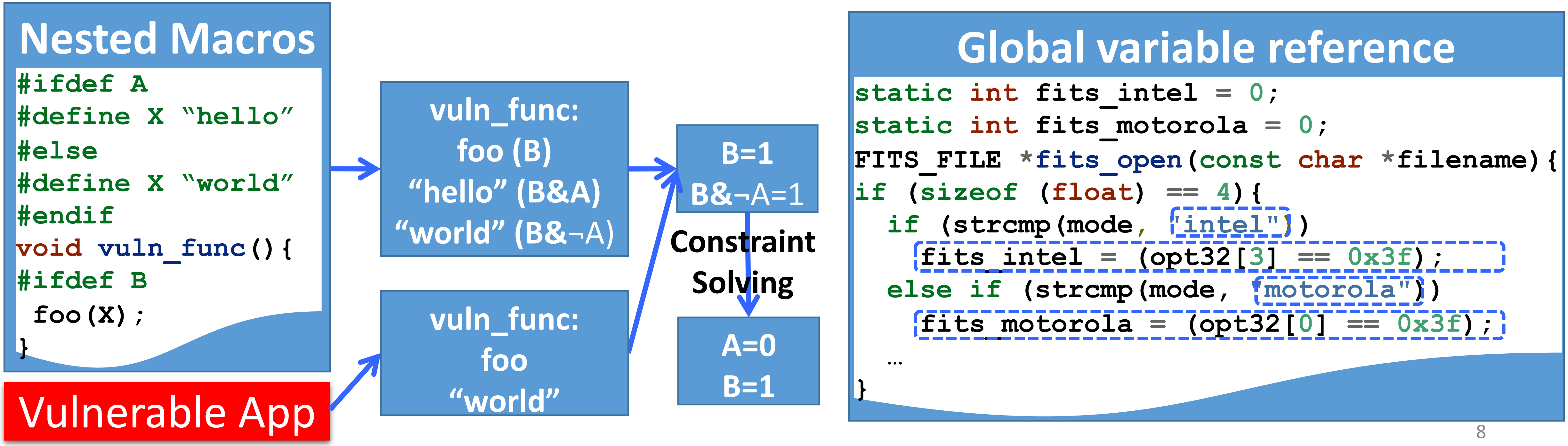
# Feasibility Analysis

- Ensure patched lines are within functions
- Apply patches to vulnerable versions (i.e. git apply)
- Check for referenced types, structures and functions

```
@@ static int ssl_scan_serverhello_tlsext(SSL *s,…
#ifndef OPENSSL_NO_EC
    *al = TLS1_AD_DECODE_ERROR;
    return 0;
  }
- s->session->tlsext_ecpointformatlist_length = 0;
- if (s->session->tlsext_ecpointformatlist != …
- if ((s->session->tlsext_ecpointformatlist = …
+ if (!s->hit)
+  *al = TLS1_AD_INTERNAL_ERROR;
```

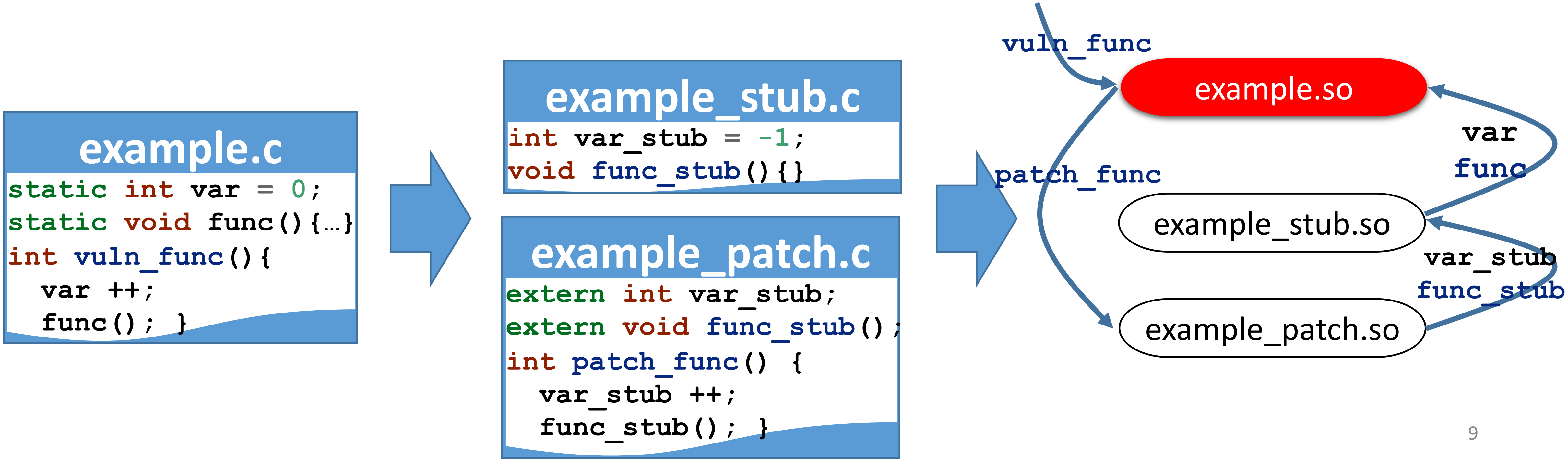
# Source vs Binary Matching

- ***Function matching***: function names or reference/call relationship
- ***Config inference***: variability-aware source features
- ***Variable matching***: variable names or related features in PDG



# App Patching

- OSSPatcher performs in-memory patching when the app launches
  - Patching techniques can be hot-patching at runtime or binary rewriting
- Detour-based function patching and fix references via stub libraries



# Implementation

- Data collection
  - cve-search for CVEs, OSSPolice for vulnerable apps, and OSSFuzz for compile commands
- Source patch analysis
  - Clang-based feasibility analysis, and TypeChef for VAST building
- Source-to-binary matching
  - IDA Pro for function identification, Angr for binary feature extraction, and Z3 to solve configurations
- App patching
  - Clang-based patch generation, and Criu for patch injection

# Evaluation - Source to Binary Matching

- Ground truth
  - Built 174 binaries from 6 selected OSS (e.g. OpenSSL, FFmpeg)
  - Compiled with default configuration (*./configure*) and turned on/off one feature to get customized binaries (e.g. *--enable-dumpcap* for wireshark)
- Experiments
  - Variability-aware featur
  - Feature extract from st
  - Matching results are co
- Results
  - 95% precision and 82% recall

- Fallback mechanism for false positives
- Missed functions remain functional and vulnerable
- A richer set of features such as control-flow features may help reduce false negatives

# Source Patch Measurement

- 60% of 1,140 patches from 39 OSS are feasible
- 77% of 251 FFmpeg patches and 83% of 97 OpenSSL patches were automatically applied to at least one vulnerable version

No need to deal with the whole OSS since only a few functions are vulnerable

- **197** functions in FFmpeg were changed across **193** feasible patches, **145** functions in OpenSSL were changed among **80** feasible patches

Vulnerabilities are located in medium to large functions, with abundant features

- Average function sizes of FFmpeg and OpenSSL were **102** and **153**, average feature sizes were **25** and **31**

# Patched Exploit Showcase

- Ran OSSPatcher on 10 vulnerabilities with public exploits and feasible patches, and thwarted their exploitation after patching
- Android Chrome (use after free)
  - Used Chrome to open a malicious xml file (calls libxml2) → use after free
  - Patched functions: xmlXPathCompOpEvalPositionalPredicate
- Stagefright (remote code execution)
  - Fed Hangouts app of Android 5.0 with malicious mp4 file → reverse shell
  - Patched functions: SampleTable::setSampleToChunkParams
- Heartbleed (stealing data in memory)
  - Setup Apache Httpd with OpenSSL 1.0.1f → steal information
  - Patched functions: dtls1_process_heartbeat and tls1_process_heartbeat

# Related Works

- Kernel patching
  - Ksplice (EuroSys’09), Karma (Security’17)
- App patching
  - PatchDroid (ACSAC’13), Instaguard (NDSS’18)
- N-day OSS vulnerability detection
  - LibScout (CCS’16), OSSPolice (CCS’17)
- Source patch analysis
  - A Large-Scale Empirical Study of Security Patches (CCS’17)

# Conclusion

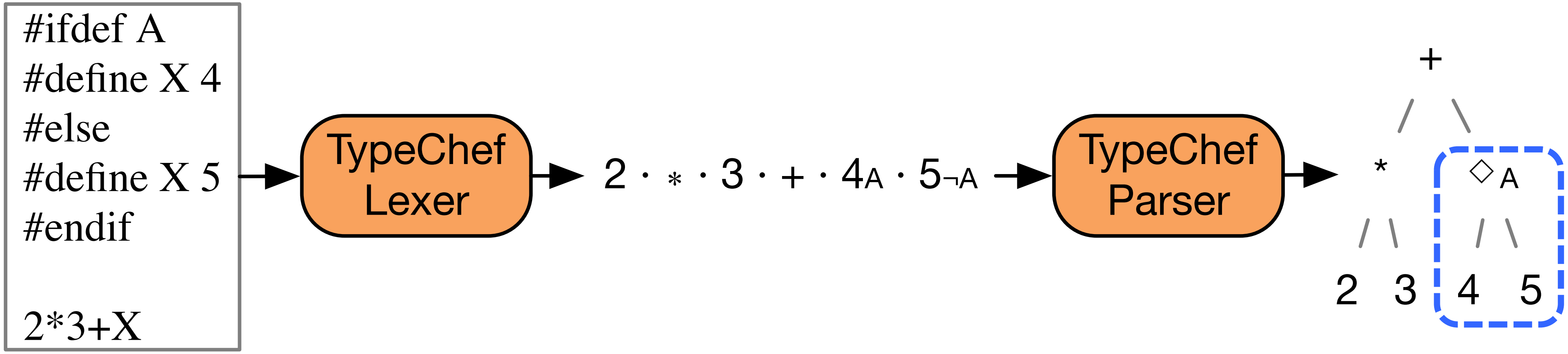
- OSSPatcher: an automated system that fixes n-day OSS vulnerabilities in **app binaries** by automatically converting feasible **source patches** into binaries and performing **in-memory** patching
  - Variability-aware patching feasibility analysis
  - Variability-aware source-to-binary matching
  - Non-disruptive in-memory patching

- Measurement
  - 675 source patches (60%) from 39 OSS are feasible
  - Incurs negligible memory and performance overhead
  - Apps are functional and exploits are thwarted after patching

# Q&A

# Feature Extraction

- Features that are present in both source code and binaries, e.g. strings, constants, function calls and external variables
- Build Variability-aware Abstract Syntax Tree (VAST) to get conditional features, e.g. 4 (A), 5 (¬A)
- Feature-based summarization for functions, e.g. foo contains 4



# Source and Binary Variability Measurement

- OpenSSL uses 55 macros in vulnerable functions, which further expands to 82 in VAST
- FFmpeg uses 25 macros in vulnerable functions, which expands to 30 in VAST
  - FFmpeg uses a configure script to allow conditional compilation at the module or folder level
- Configurations of function ssl3_get_key_exchange for 2,340 Apps using OpenSSL 1.0.1e, 17% apps use non-default config

# Source Patch Measurement

- Cross-version portability of patches
  - 80% of patches has <40 VV and can be applied to <15 FV
  - 50% of patches have >35% FV/VV ratio

Newer versions are more likely to be feasible!

- Distribution of function sizes and patch sizes
  - 80% of patches changes <40 and <10 lines in OpenSSL and FFmpeg
  - 50% of vulnerable functions have >90 and >70 lines of code in OpenSSL and FFmpeg

Patches are small fixes in large functions!

# Performance and Efficiency

- Tested 1,000 patched apps with Monkey for 5 minutes
- Memory Overhead: less than 80KB (0.1%) for 80% of apps
  - Zygote process consumes roughly 50MB of memory
- Performance Overhead
  - Before-patching (loading): less than 350 milliseconds for 80% of apps
  - After-patching (runtime): empirically conclude as negligible
- Dynamic coverage: 32% apps invoked at least one patched functions

# Discussion

- Patching techniques can be hot-patching at runtime or binary rewriting
- OSSPatcher could be applied to other Linux-based apps, e.g. Docker Hub apps
- Limitations
  - NVD information can be inaccurate
  - Cannot perform source-to-binary matching for C++, due to TypeChef
  - Dynamic code coverage for patched functions is low (32%)