JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

ALLSTAR: New Challenge Problems for Static Analysis

Jailbreak Security Summit

October 11th, 2019

evm
@evm_sec

A Future Vision for Software Reverse Engineering

Imagine if Software RE could be faster, more effective and accessible

- You have a tool that produces meaningful, descriptive labels in your disassembly of a completely new binary
- You automatically get a well-labeled software architecture diagram with a description of each module when analyzing a new binary
- People with little or no training can be given an accurate description of a new binary produced by an automated system

Talk Outline

- What problems in software reverse engineering should we consider "solved" ?

- What are the gaps between our solved problems and our existing process?

- What problems should we work on next?

- How do we get started?

My Background – The Embedded Systems RE World

- One “device” – but many boards/processors/firmwares

- Bare metal and RTOS environments
  - Large fully-linked binaries
  - No distinction between OS/libraries/application code
  - No clearly labeled system calls for clues
  - Vulnerability analysis: Takes much longer to get to the entry point

- Dynamic analysis is rarely possible
  - Unreliable JTAG/debugger access
  - Debug fuses
  - Debug BGA balls not pulled out on the motherboard
  - Integrated cores with no physical connection (debugged in development on dev boards)

What problems in software reverse engineering should we consider “solved” ?

Solved Problems?

- Decompilation

- Function-to-function Matching

- *Combined Static & Dynamic Analysis Approaches**

**since dynamic analysis is rarely possible in my world I am not qualified to evaluate the state of the toolsets (which mostly exist in other environments)*

Decompilation

- IDA Pro,[1] GHIDRA,[2] RetDec,[3] JEB[4]
- Follow process first described by C. Cifuentes in 1994[5]
  - Syntax Analysis
  - Semantic Analysis
  - Intermediate Code Generation
    - SLEIGH/p-code (GHIDRA) [6]
    - Microcode (IDA) [7]
  - Control Flow Graph Generation
  - Data Flow Analysis
  - Control Flow Analysis
  - Code Generation
- Decompilers always produce blank code!
- Incremental improvements in decompilation will not lead to fundamental changes in the speed & effectiveness of RE

Function Matching

- BinDiff[8] and Diaphora[9]

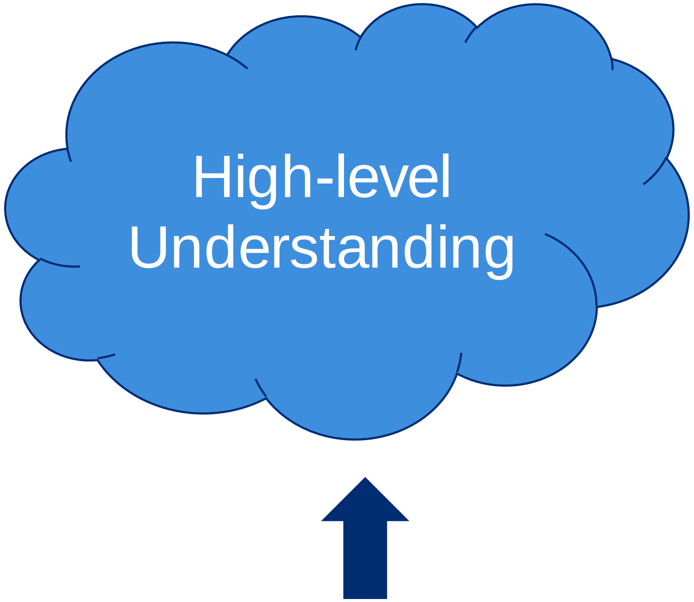
- Algorithm
  - Lossy compression function at the basic block level
    - Small primes product [10]
  - Graph-based comparison function for function flow graph
    - MD Index [11][12]
    - KOKA hash [13]
  - Graph-based comparison functions for call graph

- Works fairly well in most situations where library code is pre-identified

- Incremental improvements to function matching will likewise not lead to fundamental changes in the speed & effectiveness of RE

What are the gaps between our solved problems and our existing process?

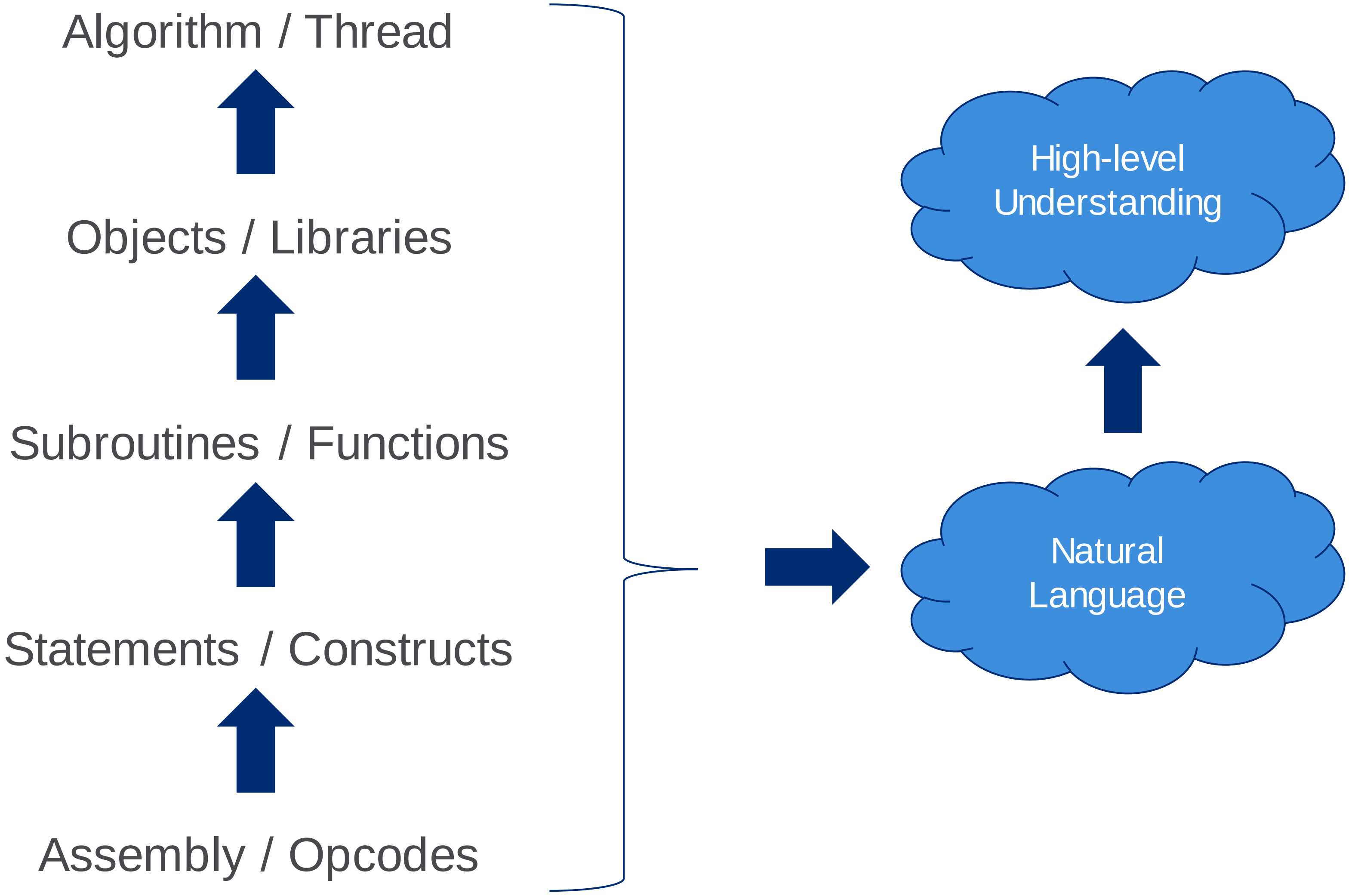
What is Software Reverse Engineering?



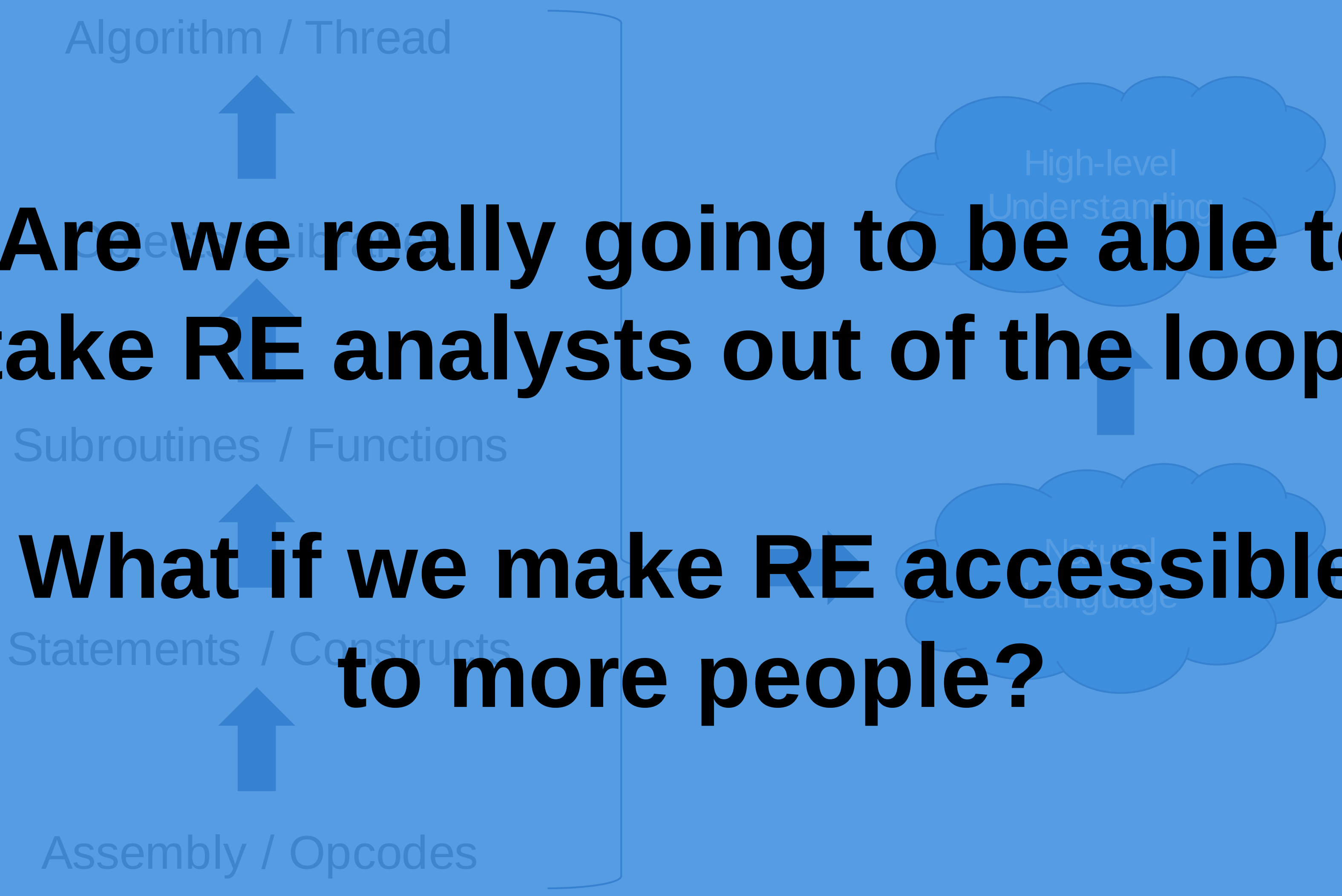
High-level Understanding

Disassembly

Software Reverse Engineering Process



Software Reverse Engineering Process



Are we really going to be able to take RE analysts out of the loop?

What if we make RE accessible to more people?

What kind of problems should we work on next?

What’s Next? – Challenge Problems

- **Variable Name Prediction** – Given a blank decompiled function, output meaningful names for the variables in that function
  - Initial research from CMU, using statistical machine translation (SMT) approach[14]
  - Inspired by work recovering identifiers in obfuscated JavaScript[15]

- **Statement Commenting** – Given a blank decompiled code statement or fragment, output comments in natural language describing it
  - What is this basic block doing?
  - Existing work in code snippet tagging/labeling[16]
  - Feed in context information from function
  - Build on variable name prediction

- **Function Summarization** – Given a blank decompiled function, output comments in natural language summarizing the function
  - Language summarization is a difficult problem due to lack of datasets
  - Existing research in source code summarization[17][18] [19]
  - Needs to be built on variable name prediction, possibly statement commenting?
  - Rely on man page / documentation descriptions?

What’s Next? – Challenge Problems

- **Library/Object Organization (“CodeCut”)** – Given a fully linked binary with no symbols, locate the boundaries between the original object files – or the boundaries between related sections of code.
  - Preliminary solution using a function affinity metric and a weighted cut graph approach[20]
  - Source code available for IDA Pro[21]

- **Library Summarization** – Given an identified set of related decompiled functions, output a name or labels describing it
  - Extension of function-level summarization work
  - Extension of code snippet tagging work

- **Foundational NLP Work**
  - asm2vec[22]
  - code2vec[23]

How do we get started?

Assembled Labeled Library for Static Analysis Research (ALLSTAR)

- NLP research will require a large (publicly available) database of:

  Open Source Code → Compile Artifacts → Binaries (with Symbols)

- There is a TON of open source code
- There is a TON of firmware out there

- But….there are ZERO debugging symbols

  (ok almost zero)

- And *ideally* it would be cross-architecture…(so we avoid overtraining on x86)

ALLSTAR Build Process

- Uses Debian “jessie” distribution

- Try to remove packages that are documentation, debug, data-only, Python, Java, etc. – leaves over 32,000 packages

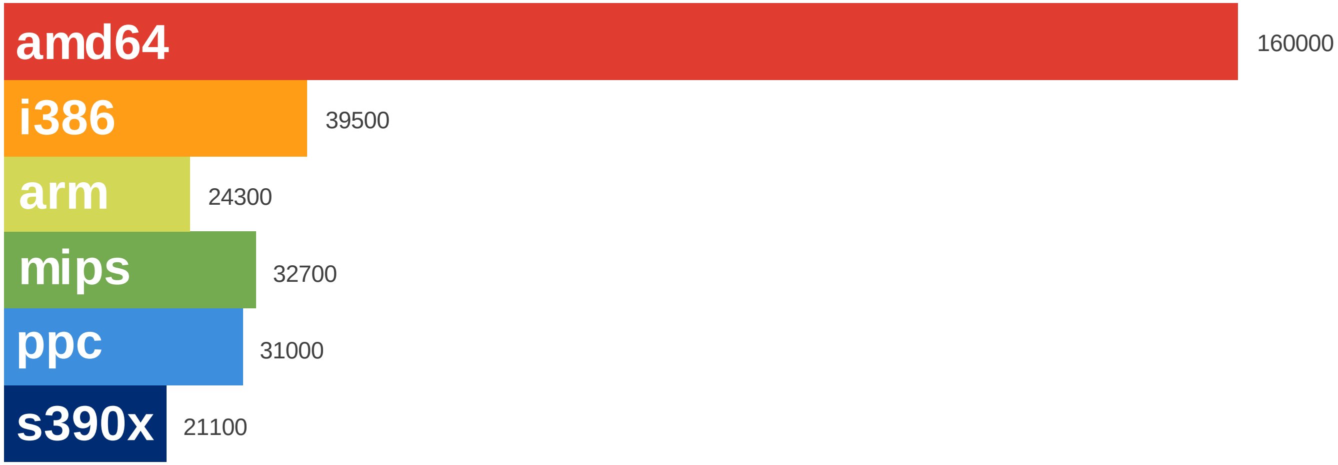
- Build for all Debian architectures (x86, amd64, ARM, PPC, MIPS, s390x) using Dockcross (Docker containers with cross compilers)

- Fairly simple technical details:
  - CFLAGS: -g -fdump-final-insns -fdump-tree-gimple
  - CXXFLAGS: -g -fdump-final-insns -fdump-tree-gimple -fdump-class-hierarchy
  - DEB_BUILD_OPTIONS: nostrip debug
  - Build with ‘debuild’ setting architecture

- Takes 5 to 6 weeks (?) to build, 35-55 packages per hour

ALLSTAR Stats*

- 1.1 TB total storage
- Binaries:



- i386, MIPS and PPC numbers reflect Debian packages that properly conform to cross-arch build spec
- ARM and s390x lower due to Dockcross' cross compiler (uses a sysroot)
- ~20K cross-platform binaries is still a nice dataset

*projected, build currently in process

Debian vs. Github

~32K packages (mostly C/C++)	~400K packages in C [14]
Structured build process (debuild)	Build by trying “configure” and “make”
Packages build cross-architecture (theoretically, if they follow spec)	Less likely to build for non-amd64
Less duplication	More duplication (up to 70%! [23])
GPL or similar open licensing	Licensing unclear / non-obvious
More serious/polished code?	A bit more random code?

What’s in ALLSTAR? A Single Data Record

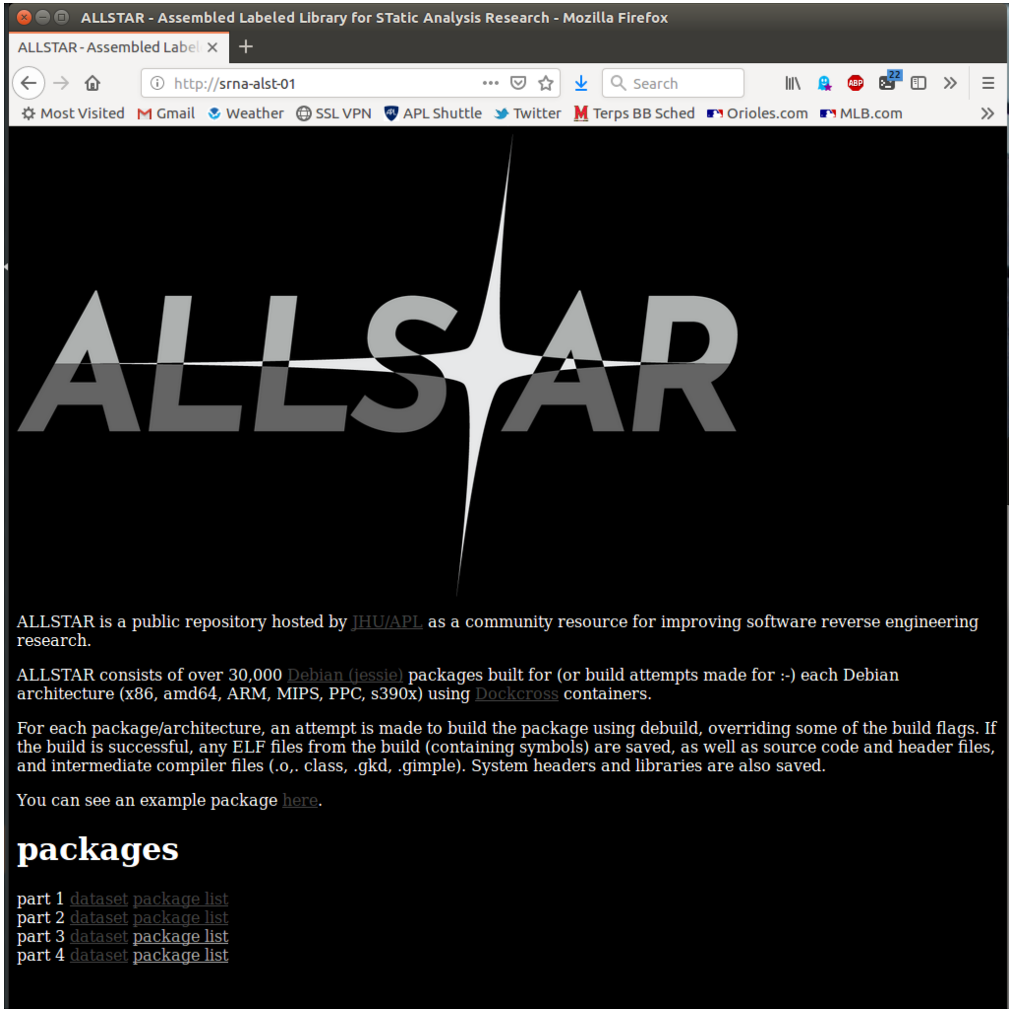
- Binary (fully linked ELF, either runnable or .so)
  - Source file
    - Header files
    - GIMPLE file (like an abstract syntax tree)
    - .class file (if C++)
    - gkd file (final internal representation in RTL)
    - Object file
  - Man page
  - Symbols (in binary) – can be used to produce a .map file
  - Any system library dependencies

- Package Documentation

- HTML index for browsing

- JSON index for code parsing

ALLSTAR – Browser Interface



ALLSTAR is a public repository hosted by JHU/APL as a community resource for improving software reverse engineering research.

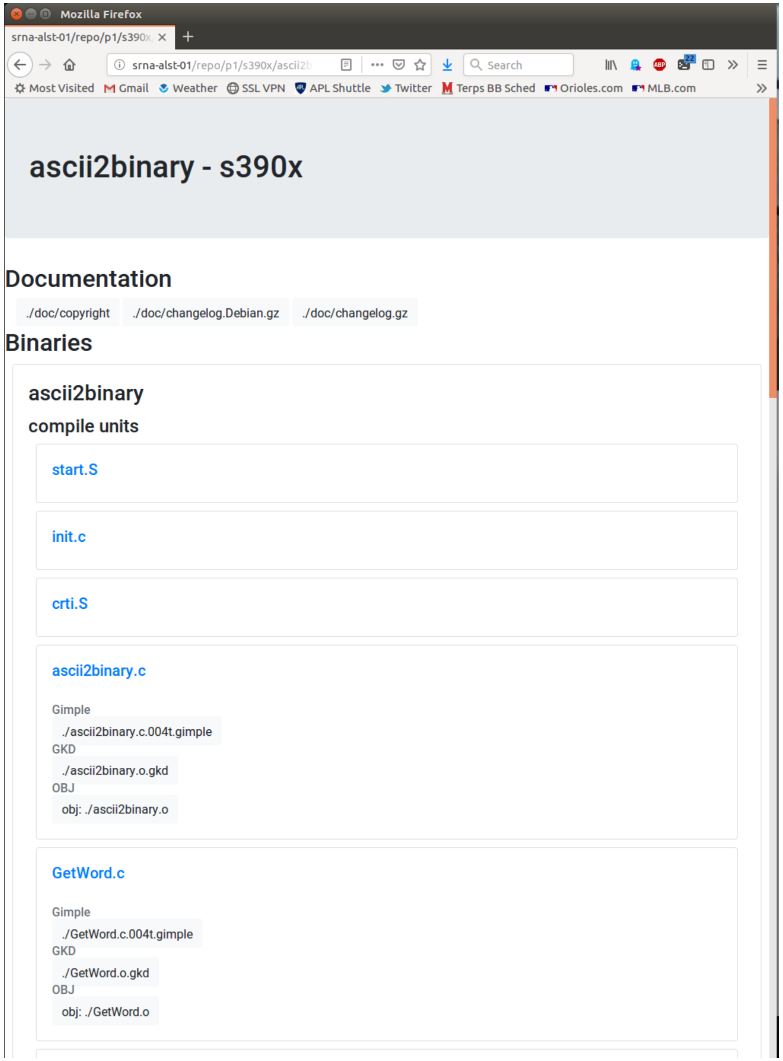
ALLSTAR consists of over 30,000 Debian (jessie) packages built for (or build attempts made for :-) each Debian architecture (x86, amd64, ARM, MIPS, PPC, s390x) using Dockcross containers.

For each package/architecture, an attempt is made to build the package using debuild, overriding some of the build flags. If the build is successful, any ELF files from the build (containing symbols) are saved, as well as source code and header files, and intermediate compiler files (.o,. class, .gkd, .gimple). System headers and libraries are also saved.

You can see an example package here.

packages

part 1 dataset package list
part 2 dataset package list
part 3 dataset package list
part 4 dataset package list



ascii2binary - s390x

Documentation

./doc/copyright ./doc/changelog.Debian.gz ./doc/changelog.gz

Binaries

ascii2binary

compile units

start.S

init.c

crti.S

ascii2binary.c

Gimple
./ascii2binary.c.004t.gimple
GKD
./ascii2binary.o.gkd
OBJ
obj: ./ascii2binary.o

GetWord.c

Gimple
./GetWord.c.004t.gimple
GKD
./GetWord.o.gkd
OBJ
obj: ./GetWord.o

Building on ALLSTAR

- Open source / publish by end of 2019
- 2 internal research projects for FY20 planned
- Hope to see a lot of research building on ALLSTAR in the future

APL

JOHNS HOPKINS

APPLIED PHYSICS LABORATORY

Special Thanks To:

- halvarflake

Additional Thanks To:

- Igor Skochinsky
- Joxean Koret
- Joan Calvet

Questions?

THE REASON IPAS ARE SO POPULAR IS THAT THEY TASTE BETTER THAN ANY OTHER STYLE OF BEER
TASTE IS SUBJECTIVE. AMERICAN IPAS TEND TO OVEREMPHASIZE HOPS, OVERPOWERING ALL OTHER FLAVORS THAT MAKE A BALANCED BEER!
NOT ALL IPAS ARE UNBALANCED - BESIDES THEY SHOWCASE UNIQUE AMERICAN HOPS VARIETIES LIKE CASCADE AND CITRA!
THEY'RE JUST WAY TOO BITTER. I DON'T WANT TO DRINK A LIQUID HOPS SALAD!
HAVE AN IPA!!!

Questions?

evm

@evm_sec

evm.ftw@gmail.com

References (1/4)

1) Hex-Rays. "IDA: About." https://www.hex-rays.com/products/ida.
2) National Security Agency. "Ghidra." https://ghidra-sre.org/.
3) Avast Software. "RetDec::Home." https://retdec.com/.
4) PNF Software. "JEB Decompiler by PNF Software." https://www.pnfsoftware.com.
5) C. Cifuentes. "Reverse Compilation Techniques." Phd diss. Queensland University of Technology, 1994.
6) National Security Agency. "Ghidra Pcode." *Ghidra source code*. https://github.com/NationalSecurityAgency/ghidra/tree/master/Ghidra/Framework/SoftwareModeling/src/main/java/ghidra/pcode.
7) I. Guilfanov. "Decompiler internals: microcode." *RECON 2018*. Brussels, Belgium. http://www.hex-rays.com/products/ida/support/ppt/recon2018.ppt.

References (2/4)

8) zynamics. “BinDiff.” https://www.zynamics.com/bindiff.

9) J. Koret. “Diaphora.” https://diaphora.re.

10) T. Dullien, and R. Rolles. "Graph-based comparison of executable objects." *SSTIC 5, no. 1*. 2005.

11) T. Dullien, et al. “Automated Attacker Correlation for Malicious Code.” *Information Systems and Technology Panel (IST) Symposium*. Talinn, Estonia. 2010.

12) H. Flake and S. Porst. “ShaREing Is Caring.” *CANSEC West 2010*. Vancouver, Canada. 2010. https://www.slideshare.net/cblichmann/shareing-is-caring-3856188.

13) C. Karamitas, and A. Kehagias. "Efficient features for function matching between binary executables." *In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 335-345. IEEE, 2018.

References (3/4)

14) A. Jaffe, et al. “Meaningful variable names for decompiled code: A machine translation approach.” In *Proceedings of the 26th Conference on Program Comprehension* (pp.20-30). ACM.

15) B. Vasilescu, et al. “Recovering clear, natural identifiers from obfuscated JavaScript names,” in *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*. 2017, pp.683-693.

16) B. Gelman et al. “A language-agnostic model for semantic source code labeling.” In Proceedings of the 1st International Workshop on Machine Learning and Software Engineering Symbiosis, pp.36-44. ACM, 2018.

17) J. Moore et al. “A Convolutional Neural Network for Language-Agnostic Source Code Summarization.” *arXiv preprint arXiv:1904.00805,* 2019.

18) S. Iyer et al. “Summarizing source code using a neural attention model.” In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016.

References (4/4)

19) X. Hu et. Al. “Deep code comment generation.” *In Proceedings of the 26th Conference on Program Comprehension*, pp.200-210. ACM, 2018.

20) evm. “A Code Pirate’s Cutlass: Recovering Software Architecture from Embedded Binaries.” *Shmoocon 2019*. Washington, DC. 2019.

21) evm. “CodeCut: Locating Object File Boundaries in IDA Pro.” https://github.com/JHUAPL/CodeCut.

22) S. D’Antoine. “asm2vec: Binary Learning for Vulnerability Discovery.” *Jailbreak Security Summit 2018*. Laurel, MD. 2018.

23) U. Alon et al. “code2vec: Learning distributed representations of code.” *Proceedings of the ACM on Programming Languages 3*, 2019.

24) P. Martins, et al. “50K-C: A dataset of compilable and compiled, Java projects.” In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR),* pp 1-5. IEEE, 2018.