ICE: Circumventing Meltdown with an Advanced Binary Analysis Framework

Dean Pucsek
University of Victoria
dpucsek@uvic.ca

Jonah Wall
University of Victoria
jojonah@uvic.ca

Celina Gibbs
University of Victoria
celina@cs.uvic.ca

Jennifer Baldwin
University of Victoria
jebaldwin@cs.uvic.ca

Martin Salois
Defence Research & Development Canada
martin.salois@drdc-rddc.gc.ca

Yvonne Coady
University of Victoria
ycoady@cs.uvic.ca

ABSTRACT

Given the spread of software from the desktop PC to mobile devices, GPUs and embedded systems it is no longer a question of *if* a new architecture will be encountered it is a question of *when*. Consequently, program comprehension tasks now require support for arbitrary analyses to take place on arbitrary instruction set architectures. This paper proposes *ICE*, an Integrated Comprehension Environment, intended to address challenges associated with binary analysis that is independent of both the instruction set and platform, and provides support for interoperable tools.

1. INTRODUCTION

Though the prevalence of software is a direct result of the many advantages it provides, it has also resulted in codebases with properties that are no longer possible to comprehend by traditional means. That is, manual inspection assisted by comprehension tools targeting specific high level languages cannot currently scale to most applications built with a variety of components. This lack of understanding is exacerbated by the absence of source code, in addition to subtle and complex interactions between the software and the rest of the system. Furthermore, even the assistance offered by low level tools that provide dynamic program comprehension support are now faced with a plethora of computer architectures (x86, ARM, PowerPC, etc.), binary file formats (ELF, Mach-O, PE, etc.), analysis techniques, and an ever growing landscape of obfuscation techniques employed by malware developers. Given the future trajectory of multicore applications that are bound to amplify these problems, we believe it is paramount to consider extensibility as a primary requirement for program comprehension tools.

Extensibility is crucial because it will allow a framework of comprehension tools to more easily adapt to both current and future technologies. Moreover, a focus on extensibility will allow us to form a community of third-party developers who are able to contribute ideas and code to better the system. To answer this need we propose an Integrated Comprehension Environment (ICE), designed specifically to assist program comprehension tasks associated with understanding software. ICE is also intended to be interoperable with tools designed for a wide variety of areas including program verification, program comprehension, malware analysis, vulnerability detection, and education.

In this paper we begin by discussing related work in Section 2 and motivate the need for extensibility in Section 3. Section 4 proposes a framework that is both extensible and capable of performing arbitrary analyses on software. In Section 5 we present an evaluation of our proposal as well as some of the associated challenges that may be encountered. Finally, we present our plans for future work and conclusions in Section 6.

2. RELATED WORK

Most people are familiar with the *telephone game*, in which a player whispers a starting sentence to a second player, who whispers it to the third player and so on until the last player speaks it out loud. More often than not, the original sentence is utterly deformed, to the general amusement of the crowd. Few people know that the same phenomenon is almost a given when translating from one computer language to another. This characteristic is due to the presence of heterogeneities between data structures. When a data type does not exist in the target structure, the human translator will still represent it somehow. This will inevitably result in information loss or, even worse, false information. Euzenat and Shvaiko [10] provide a general introduction to the subject for interoperability, and Dorion et al [8] provide a detailed characterization of the possible heterogeneity types, leading to a measure of how much information is lost.

There are multiple strategies and projects dedicated to the comprehension of software and their translation to an intermediate form or language. Formalized Intermediate Languages (ILs) have been developed for both the translation of source to a specified IL and the translation of binaries or

executables to an IL. In both cases, the IL facilitates program analysis through integration with tools that leverage the structure of the given IL.

For example, SAIL [7] is an open source, front-end which, given a program's source code, generates two levels of ILs and creates a graphical representation of the program's control flow and is conducive to static analysis. On the other hand, REIL (Reverse Engineering Intermediate Language) [9] provides an IL abstracted from native assembly code supporting the development of analysis tools and algorithms that are platform independent. REIL's limited instruction set introduces a one-to-many relation between native instructions and REIL instructions, and therefore leaves it unable to translate certain classes of instructions. Unusual instructions are essentially ignored by translating them into a variant of the `NOP` instruction, like the `UNKN` instruction in REIL that the translator uses to signal an unrecognized instruction [1]. However, if such an instruction exists in an IL, dynamic binary analysis becomes infeasible [15]. If dynamic binary analysis is required, it must then be accomplished by means of an emulator or a whole-system simulator. HLASM [13], for example, is significantly different than x86 assembly and currently has no translation to REIL.

IDA Pro [11] is a popular interactive disassembler that allows for inspection of software through analyses both built-in and external to IDA Pro. IDA Pro follows a traditional plug-in architecture and provides an API that allows third-party developers to produce more complex analyses [5]. There are several other frameworks in place, each with their own IL, and each with their own advantages and drawbacks.

HERO (Hybrid sEcurity extension of binaRy translatiOn) [12] is a promising framework that claims to support an efficient combination of static and dynamic binary analysis methods. One key advantage to HERO is that it appears to be entirely self-contained; that is to say, it does not seem to rely on any third-party components. A formal specification language is used for its IL, but no details are revealed about its implementation to verify the claim that it is a faithful representation of the assembly code.

BitBlaze [16] is a framework that was developed at UC Berkeley and aims to provide a better understanding of software through a fusion of static and dynamic analysis. It consists of three main components: VINE, TEMU, and Rudder. BitBlaze achieves dynamic analysis through the use of a whole-system emulator; a recent study revealed some emulators unfaithfully emulate instructions, resulting in system misrepresentation, emulator malfunction, or failure [14]. BAP (Binary Analysis Platform) [6], a spin-off of BitBlaze, uses a series of transformations to go from assembly code to the BAP IL; however, as noted previously in the context of the telephone game this contributes to information loss and possibly incorrect representation. One key improvement is that BAP is distributed as a VMWare [17] virtual machine, eliminating the troublesome and complicated installation required for BitBlaze.

Although there are many tools available to assist the process of understanding software they are largely based on the concept of translating from an assembly language to an IL, and are typically developed for specific architectures, platforms, and analyses. While these tools may be a step forward they are unable to cope with changes in computing, such as Microsoft Windows being ported to ARM [4], and lack the ability to incorporate new analysis techniques.

3. THE PROBLEM: EXTENSIBILITY

As established by the survey in Section 2, the primary deficiency with current support for program comprehension is the inability for these tools to keep pace with modern codebases. Unless analysis tools are designed to fluidly adapt to constantly changing circumstances they will not be able to provide adequate support and will quickly become obsolete. In order to address this challenge, we propose to design ICE to be extensible in three critical dimensions: 1) instruction set architectures (ISA), 2) software platforms and 3) state-of-the-art analysis techniques.

With respect to instruction set architectures, more software is breaking out of the desktop PC each year and forging paths in new areas such as mobile devices, computing on GPU's, and embedded systems (e.g. game consoles). As a result, it is no longer a question of *if* a new architecture will be encountered, it is a question of *when*. Knowing that new architectures are increasingly inevitable, it follows that ICE must provide a means by which to add new instruction set architectures and that this mechanism must be able to accurately and faithfully model the intricacies of the architecture in use.

In terms of software platforms, rather than dictating which platform (e.g. Microsoft Windows, Mac OS X, Linux) users of ICE must be on, users should be able to choose. As a result of loosening the platform requirement we allow ICE to fit into existing work flows and we increase the number of potential users. Unfortunately, a system as substantial as ICE will have some platform dependent components such as graphic libraries which must be accommodated. In order to provide such accommodation, it will be necessary to modularize these components such that ICE appears to be platform agnostic from the perspective of the tools provided.

Finally, and perhaps most importantly, it must be possible to easily integrate new and unanticipated analysis techniques into ICE, and specifically allow for interoperability with other tools that are part of the framework. If we were to limit the type of analysis available, we would both hinder the capability of ICE and prevent users from being able to visualize results in a manner that suits them.

4. PROPOSAL: THE ICE FRAMEWORK

With extensibility as the key motivating factor for the creation of ICE, we envision a modular implementation as depicted by the framework design illustrated in Figure 1. This layered design will meet the requirements outlined in Section 3 to provide support for new instruction set architectures, maintaining platform independence and supporting interoperability of new analysis modules.

At the lowest level, the *Platform Substrate* contains the components of the ICE that are unavoidably platform dependent. At the middle layer, the *ICE Core* oversees the entire system and provides a layer of abstraction between, and

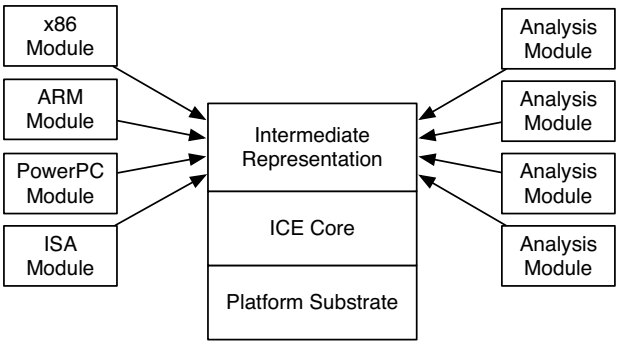


Figure 1: Proposed framework design

manages modules for, different instruction set architectures and software platforms. This technique is similar to that used in the Eclipse IDE [2] in order to achieve increased portability between platforms.

Finally, an *Intermediate Representation* would provide a normalized interface for analysis tools to integrate in a uniform way with arbitrary instruction set architectures as depicted in the Figure 1. The Intermediate Representation is critical to the extensibility of the system as it is the component that allows arbitrary analyses to take place on arbitrary instruction set architectures. Our investigation into related work has demonstrated the limitations of a strictly IL based approach, and as a result we have turned to a more general and inclusive approach based on the addition of protocols and APIs for interoperability.

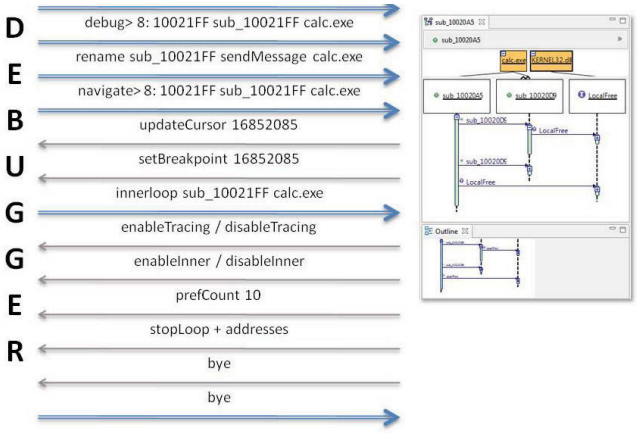


Figure 2: Tracks' debugger interaction protocol.

For example, a proof of concept tool we have developed is Tracks [5]. Tracks works with information provided by an underlying debugger, allowing for dynamic analysis and navigation in an Eclipse plug-in. As shown in Figure 2, the protocol between Tracks and the debugger, provided by IDA Pro, establishes communication for the analysis which includes system calls. This essentially extends the static capabilities of IDA Pro to incorporate dynamic analysis, external modules, and even multi-threaded code bases. This protocol and API could also be used interchangeably to provide the same support for navigation on top of other debuggers, for example, those associated with HLASM code bases.

5. EVALUATION

In Section 3 we identified three key criterion that ICE must meet to provide a suitable level of extensibility: accommodation of new ISA's, platform independence, and interoperability of arbitrary analysis tools. The proposed design in Section 4 strives to meet these criteria; however, it is worthwhile to look at how well it does this and what problems may be encountered in the future.

5.1 Accommodate New ISAs

Our proposed design accommodates new ISA's by treating each as a module that can be plugged-in to the rest of the system. This will allow for a great deal of extensibility because each ISA module will interact with an API that enables the Intermediate Representation to query it and obtain the required information for analysis. However, as we have seen in our survey of ILs, not all ISA's can fit to a single model which means we will need to pay close attention to how the more obscure instructions are handled.

5.2 Platform Independence

The second criterion for extensibility, platform independence, is achieved by minimizing the size of the Platform Substrate and having the ICE Core provide a generic view of the platform to the rest of ICE. As previously mentioned, there must be a platform dependent component of ICE that performs tasks such as rendering a graphical user interface. Since this is unavoidable, the best we can do is try to minimize the number of platform dependent components and provide a common interface to the rest of ICE as much as possible. Note that projects such as Google Chrome [3] and Eclipse [2] have clearly demonstrated that it is possible to provide a clean, intuitive, and easy-to-use interface while maintaining platform independence.

5.3 Tool Integration

Finally, our criteria for extensibility stipulates that users must be able to integrate state-of-the-art analysis techniques to create their own Analysis Modules. This is accounted for in the design through a combination of the Intermediate Representation and the use of Analysis Modules. The Intermediate Representation acts a bridge between the ISA Module and the Analysis Module by representing the details of the software in an intermediate form. This technique then allows an Analysis Module to query the Intermediate Representation and perform any arbitrary analysis.

On the surface this proposal appears to have no major issues. However, digging deeper reveals that the ISA Module in use is unknown during development of the Analysis Module. But since an Analysis Module may require the use of some particular feature in an ISA Module, we must provide a mechanism for the Analysis Module to learn which ISA is in use and to query the module directly. By doing this we will alleviate some of the pressure on the Intermediate Representation to handle all cases and still maintain a reasonable level of extensibility.

Another challenge encountered by allowing the creation of arbitrary analysis is how to provide a suitable environment

for dynamic analysis. Unfortunately, there is no guarantee that a suitable environment will be available and creating an environment that executes an intermediate representation of the software could introduce inaccuracies into the analysis. There is no clear answer to this issue but it may be solvable by leveraging extensibility once again and allowing a user to select any virtual environment they wish.

6. CONCLUSIONS & FUTURE WORK

The requirement of extensibility in ICE has raised a number of questions that will require further work to answer. As noted in Section 4 the Intermediate Representation will act as a bridge between the Analysis Modules and the ISA Modules. Currently the specification of this Intermediate Representation is under investigation. In addition to the general API described here, we believe a hierarchical IL structure may help meet our requirement for extensibility.

Along with Intermediate Representations we will also be investigating how to provide a clean method of performing dynamic analysis in a virtual environment. We are particularly interested in understanding how ICE should interact with malware, especially when malware tries to detect our system, and how to accurately monitor the virtual environment without introducing inaccuracies into the results.

In this paper we demonstrated that in order for ICE to stay relevant in a world of rapidly changing software it must be extensible. Our proposed framework is designed from the ground up with extensibility as a guiding principle. While the benefits of extensibility are large, we have identified several issues that will be investigated as they arise. Software will continue to spread into new domains and will become an ever growing part of our lives. Equipped with ICE it will be possible to understand how software operates and provide insight into other software development issues.

7. REFERENCES

[1] BinNavi. http://www.zynamics.com/binnavi.html, 2011.

[2] Eclipse platform technical overview. `http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html`, January 2011.

[3] Google chrome. `http://www.google.com/chrome`, January 2011.

[4] Microsoft announces support of system on a chip architectures from intel, amd, and arm for next version of windows. `http://www.microsoft.com/presspass/press/2011/jan11/01-05SOCsupport.mspx`, January 2011.

[5] J. Baldwin, P. Sinha, M. Salois, and Y. Coady. Progressive user interfaces for regressive analysis: Making tracks with large, low-level systems. *AUIC 2011*, pages 1–10, Nov 2010.

[6] D. Brumley and I. Jager. The BAP Handbook. 2009.

[7] I. Dillig, T. Dillig, and A. Aiken. SAIL: Static Analysis Intermediate Language with a Two-Level Representation. Technical report, 2009.

[8] E. Dorion, D. Grenier, J. Brodeur, and D. O'Brien. A Taxonomy of Heterogeneity Types for Ontological Alignment. In preparation, Dec. 2010.

[9] T. Dullien and S. Porst. REIL: A platform-independent intermediate representation of disassembled code for static code analysis. In *CanSecWest Applied Security Conference)*, 2009.

[10] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag, Heidelberg (DE), 2007.

[11] I. Guilfanov. Decompilers and beyond. *BlackHat USA 2008*, pages 1–12, Jul 2008.

[12] H. Guo, J. Pang, Y. Zhang, F. Yue, and R. Zhao;. Hero: A novel malware detection framework based on binary translation. *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, 1:411 – 415, 2010.

[13] IBM. High Level Assembler (HLASM) and Toolkit Feature. http://www-01.ibm.com/software/awdtools/hlasm/library.html, 2011.

[14] L. Martignoni, R. Paleari, G. Fresi Roglia, and D. Bruschi. Testing CPU emulators. In *Proceedings of the 2009 International Conference on Software Testing and Analysis (ISSTA)*, pages 261–272, Chicago, Illinois, USA. ACM.

[15] N. Nethercote and J. Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007)*, pages 89–100, San Diego, California, USA, June 2007.

[16] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. *Information Systems Security*, pages 1–25, 2008.

[17] VMWare. Virtualization Software for Desktops, Severs & Virtual Machines for Public and Private. http://www.zynamics.com/binnavi.html, 2011.

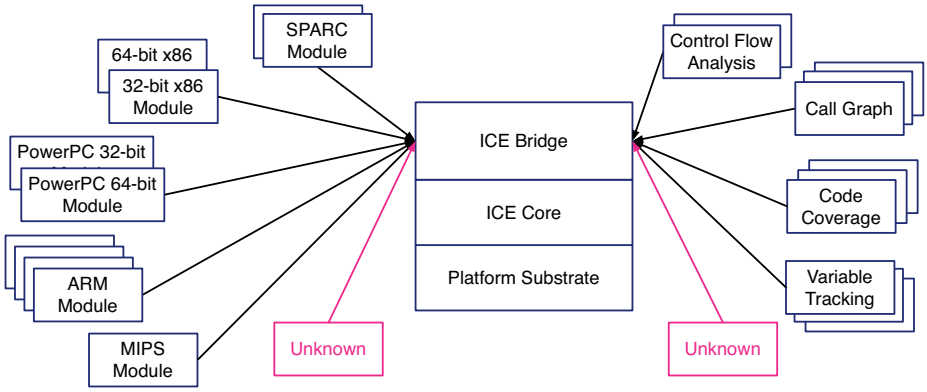
ICE: Circumventing Meltdown with an Advanced Binary Analysis Framework

The Need for Extensibility

Without extensibility, binary analysis tools are unable to adapt to the fluid world of software.

Binary analysis tools must be extensible in three key areas:

- Instruction set architectures
- Software platforms
- Analysis techniques



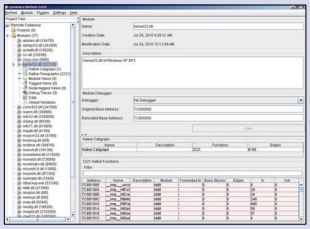
How do you understand the unknown?

Related Work

Evaluation Criteria for Binary Analysis Frameworks

1. Extensibility
2. Platform Independence
3. Static Analysis Tools
4. Dynamic Analysis Tools
5. Reliability of IL Translation
6. Architectures Supported
7. Instruction Set Completeness
8. Ease of Use & Documentation

BinNavi



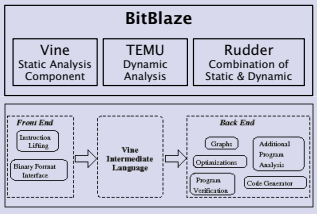
Pros

- Extensible
- Platform independent
- Visual static analysis tools
- Commercial product
- Clean GUI
- Excellent documentation

Cons

- Dynamic analysis is not feasible
- No support for IBM architectures
- No support for floating point, MMX or SSE
- No support for privileged instructions

BitBlaze



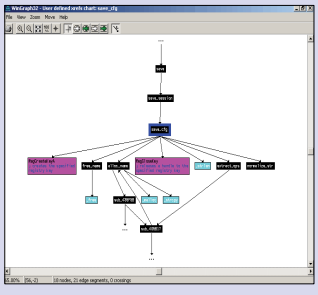
Pros

- Supports static analysis
- Supports dynamic analysis

Cons

- Extensible only in terms of analyses
- Platform dependent
- No GUI
- Several translations to generate IL
- Only 32-bit x86 is supported
- Must define external function calls
- Unclear documentation
- Intricate third-party dependencies

IDA Pro



Pros

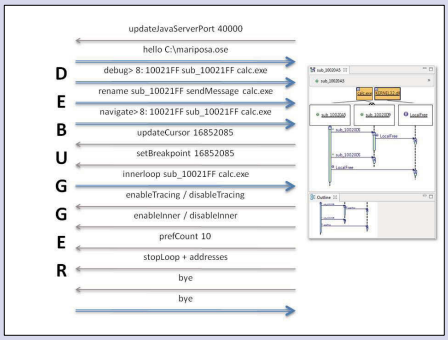
- Plug-in architecture
- Platform independent
- Visual static analysis tools
- Dynamic analysis
- Many architectures supported

Cons

- Dynamic analysis prone to errors
- IBM architectures not supported
- Many tools are difficult to use
- Difficult to use documentation

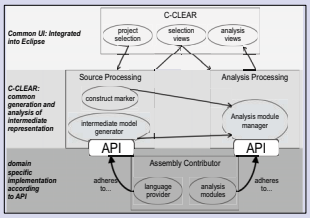
Current Progress

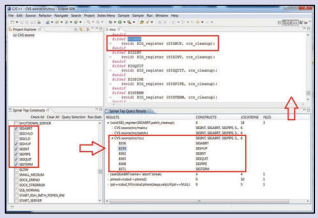
Tracks



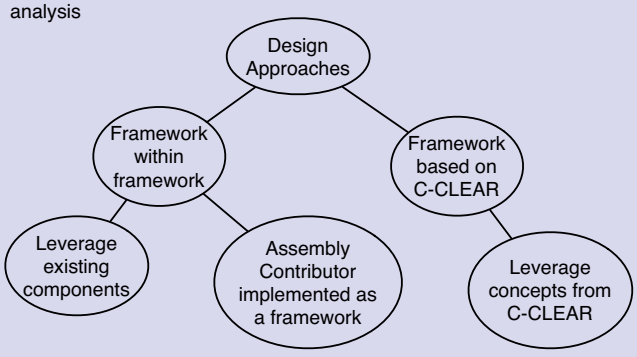
- Leverages information provided by a debugger
- Enables dynamic analysis and navigation
- Integrates with Eclipse
- Communication between debugger and Eclipse plug-in uses the Tracks protocol
- Able to analyze several binaries simultaneously
- Able to analyze multithreaded binaries

C-CLEAR





- Integrated with Eclipse
- Common UI for selecting analysis and viewing results
- Intermediate model defined as a Syntax Tree
- Assembly Contributor provides tokens and other information required for analysis



Dean Pucsek
dpucsek@cs.uvic.ca
University of Victoria

Jonah Wall
jojonah@uvic.ca
University of Victoria

Celina Gibbs
celina@cs.uvic.ca
University of Victoria

Jennifer Baldwin
jebaldwin@cs.uvic.ca
University of Victoria

Yvonne Coady
ycoady@cs.uvic.ca
University of Victoria

Martin Salois
martin.salois@drdc-rddc.gc.ca
Defence Research and Development Canada