

Working Title: Compiler for Return Oriented Programming

Student Name: Peter Samuel

Supervisor Name: Martin Berger

Aims and Objectives:

In brief, the overall aim of this project is to create a toy compiler for a simple language which can be used to smash the stack and begin an exploit known as Return Oriented Programming (ROP) [1].

Aims:

One of the main aims will be to build a compiler for most likely a stripped-down version of C. This version will contain only the bare bones needed to write the ROP exploit. That'll involve removing features such as: Typedefs and Macro pre-processors as they aren't overall necessary.

For the implementation of the compiler, Scala will be the language of choice. This is choice is primarily so that pre-existing work in Java (compiler project in [2]) can be reused easily, and new tools such as parser generators won't have to be learnt.

Another aim will be to target the compiler to output code for the x86 architecture. This is choice is made since x86 is the easiest to exploit for ROP. This is because fixed boundaries on instructions aren't enforced like they are in other architectures such as MIPS [1].

Additionally, the codegen strategy will most likely targeting a register machine that handles spilling by switching to an accumulator machine. This will be done since the efficiency gained by a proper register allocator and liveness analysis is not necessary and hence will be relegated to the extensions section.

A final aim is to then target this exploit to run on a virtualised Linux distro. This distro will have to lack ASLR, so the exploit can execute with the guarantee that the gadgets remain in the same position in memory. This will require a short investigation into potential candidate distros.

Primary Objectives:

So, to summarise from the aims, the objectives then are to:

1. Formulate a stripped-down version of C, with enough functionality to enable the exploit to be written.
2. Develop the main components of the compiler targeting the x86 architecture:
 - Lexer
 - Parser
 - Type Checker
 - Code generator
3. Write the exploit in the newly created language, this will involve:
 - Initially manually inserting the code to begin ROPing.
 - Designing a short gadget chain which will open a root shell.
4. Demonstrating the Exploit works on a Linux distro, presumably with no ASLR.

Extensions:

Possible extensions to this project include the following:

- Implementing the Galileo algorithm, and no longer relying on manually inserted gadgets.
 - Possibly implementing a gadget compiler.
- Investigations into defeating ASLR [3]
- Implementing a proper register allocator to the language and introducing some optimisations.
- Implementing run-time systems such as:
 - Garbage collector
 - Return count monitor for detecting ROPing

Relevance:

- Essentially this project is an extension on the compilers course in my second year 2, building a working albeit toy compiler will demonstrate my abilities in such areas.
- Challenge me to learn a slightly different language: Scala
- Ties into what I want to go into: pen testing and to some extent the computer security module.

Resources Required:

Minimal:

- A copy of the Tiger book [2]
- Access to the mentioned papers in the bibliography
- A running version of an Linux distro w/o ASLR [4]

All the above I currently possess or could easily get access to w/o funding.

Bibliography

- [1] H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," in *Proceedings of the 14th ACM conference on Computer and communications security*, Alexandria, Virginia, 2007.
- [2] A. Appel and J. Palsberg, *Modern Compiler Implementation in Java* (2nd Edition), Cambridge University Press, 2002.
- [3] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu and D. Boneh, "On the Effectiveness of Address-Space Randomization," in *Proceedings of the 11th ACM conference on Computer and communications security*, Washington DC, 2004.
- [4] Adrián, "How Effective is ASLR on Linux Systems?," 03 February 2013. [Online]. Available: <https://securityetali.es/2013/02/03/how-effective-is-aslr-on-linux-systems/>. [Accessed 14 October 2019].