LSE Systems team 2018 - 2019 projects

Violating your package manager's privacy

Lots of package managers use HTTPS mirrors, and some claim to provide confidentiality (that is, hiding the list of downloaded packages from attackers listening on the network) over your updates and installations. However, the SSL or TLS protocols used do not change the size of the transmitted data. Hence, it should be possible to calculate the size of a downloaded package and identify a list of likely matches among the public list of packages.
This project is a proof of concept of this idea : it is an automated tool capable of analyzing a pcap record of a TLS session and of guessing which archlinux package was downloaded over the course of the session.

Tree differencing for code copy detection

Vendoring consists of making a copy of a dependency and including it inside another project. It is the simplest way to distribute dependencies and make changes to their code base by versioning them in the same repository as the main project.
However, vendoring makes tracking upstream development more difficult, because local changes will often clash with upstream changes. Vendored dependencies are thus often outdated, and can lack recent security fixes.
This project implements the GumTree algorithm, and uses the Clang compiler API to perform code copy detection on large codebases to find shared code, potential vulnerabilities, small and large changes in a code base, or cheating students.

Implementing the pledge syscall on Linux

pledge(2) is a system call from OpenBSD aiming to mitigate possible exploits on the syscall interface, in a way that the application developer can control by specifying categories of system calls the program can use. pledge(2) is designed with simplicity in mind, meaning it can be added to most program with little development effort.
The Linux kernel does not currently have a similar mechanism, and the closest mechanisms currently existing in the Linux kernel are overly complicated, and are as such not broadly used.
The goal of this project is to implement this system call in the Linux kernel. After considering existing options that could be leveraged, the syscall will be implemented in the form of a prctl(2).

Implementing an FTP client for OpenBSD

FTP is an older file transfer protocol that is still used quite often. As a generic Operating System, OpenBSD features its own FTP client, implementing many features to cater for both human users and scripting. Although it works and is used by other programs, it is getting a bit old, hard to maintain and works by accident. Thus the need for a clean-slate rewrite into the modern ages.
An OpenBSD contributor started this update and wrote a nice base that handled basic operations. However, it had some behavior differences, lacked documentation and many common features.
This provided a good code base, but a lot still needs to be done.

Accelerate Boolean Constraint Propagation for Boolean Satisfiability Solvers with FPGA

Boolean Satisfiability (or SAT) problem consists of finding if a set of Boolean values that satisfies a given Boolean formula exists. SAT, even though it was the first problem to be proven NP-complete, is used in fields like circuit design, symbolic execution and automatic theorem proving. The algorithm used to solve SAT problems is arborescent but its most time consuming operation, the Boolean Constraint Propagation (BCP), is a good target for hardware acceleration.
Based on the work of John D. Davis et al., the goal is to design a co-processor that computes only the BCP part of the SAT algorithm. The arborescent part is computed by a CPU that sends BCP problems to the co-processor when needed. The design is written with an Hardware Description Language (HDL) and is tested on a Field-Programmable Gate Array (FPGA).

Network stack in K

K is a simple kernel implementation, in which the user can only play single player games. The goal of this project is to implement a network stack in order to play in multiplayer mode on some games. This implies to implement all the interactions from the network card driver to the user space system calls permitting to send network packets. The stack that is implemented here handles ICMP, IP, ARP, UDP and DHCP requests, as it is necessary in order to discover network, request an IP, and send packets.
This work relies on W. Richard Stevens work and on the implementation he has described in TCP/IP Illustrated book, which explains the feature of BSD-Lite's network stack implementation.

Execution trace analysis

A large part of binary analysis is figuring out what obfuscated executables actually do, and writing tools that can systematically simplify obfuscated code can help a lot. In order to reverse engineer obfuscated binaries, we want to disassemble and lift native code back to LLVM intermediate representation (IR) to be able to run optimization and simplification passes on the "higher-level" IR. The thing is: often in obuscated code, there is much indirection in what is actually going on under the hood (useless operations with side-effects, virtual machines...). To bypass this problem we can concentrate only on the code that is really executed by working on an execution trace rather than the executable directly. From this execution trace, we will build a control flow graph (CFG) of the program execution, made of assembly basic blocks that represents the flow of execution of the actual binary. This CFG can then be used to recover an IR program which has the same behavior as the original program, and that can be decompiled to pseudo-C or recompiled to native code for further analysis.

YML kernel

Most kernel components are really hard to modify on an already functional modern kernel, and developping a new one from scratch requires a lot of work on various parts of the kernel. This can be a real barrier for newcommers in kernel development.
YML is a highly modular kernel designed for components to be swapped. It features most of modern Kernel functionnalities like EFI boot, APIC and ACPI handling. The aim of this kernel is to be used for educational and research purposes.

Secure lazy binding in x86_64

Dynamic relocation is the act of loading all the needed libraries of a binary before executing it, and of writing all the needed symbol addresses inside the loaded binary. Addresses can be written at two different times:
1. Before the execution: bind now.
2. During the execution, when the symbol is used: lazy binding.

The drawback of the lazy binding is that it needs to write to the binary at runtime. To avoid switching between RO and RW permissions every time ld.so writes to memory, OpenBSD created the kbind(2) syscall, which enables ld.so to write to RO memory, and makes sure that only one code location can execute the syscall, making sure that ld.so is the only entity that can use it.
The goal of this project is to implement this syscall under a Linux OS and a common architecture, here x86_64, and use it in the ld.so of the glibc, to enable fully RO lazy binding.

BitTorrent for iPXE

Many enterprise computers have no hard disk, and download their operating system from scratch on each boot. This process uses up a lot of bandwidth, and exceptionnal situations, such as all machines being powered up at the same time, can bring a network to its knees.
However, computers close to each other can share already downloaded parts of the operating system to avoid making more requests to some central server. This project is about adding support for the BitTorrent peer-to-peer protocol into iPXE, a network specialized bootloader.