ABSTRACT

Title of Dissertation: ANALYTICS OF CONFIGURATION MANAGEMENT FOR SYSTEM ADMINISTRATION

Yehuda Aryeh Katz
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Ashok Agrawala
Department of Computer Science

System administrators are usually trusted to be experts on the systems they control, yet security breaches and performance problems can often be traced to improper configuration by these same administrators. These configuration mistakes aren't made deliberately and certainly not maliciously, but are usually due to a lack of information about the consequences and interactions of these settings. This problem becomes more apparent as the complexity of the software being configured grows and as the role of system administrator is taken on by more people with less time to develop a complete understanding of the systems they control. We call this *Uninformed Configuration*. There is a blind spot in existing scientific research when it comes to understanding the effects of configuration changes on system performance and security, which if well understood would allow for informed configuration management.

We present a new way to analyze and understand the effects of configuration management. We define a clear division between the operations of a program that are controlled by configuration and the operations of a program that are affected by the data the program is processing. This allows us to make more accurate inferences about how changing a configuration *knob* will affect the overall security and performance of the system. We build on existing static analysis tools and control flow representations originally designed for compiler optimization to build a clear picture of the effects of configuration changes. We refine the concept of understanding program execution paths with a control plane and data plane by focusing on the effects of configuration changes as a part of the control plane.

We provide a method for communicating the importance of each configuration knob to a system administrator using a standardized ranking and scoring system. We also apply these methods to configuration knobs with known performance and security effects in two commonly used pieces of software.

Finally, we discuss several future avenues of scientific research and practical work which will carry these ideas further to improve the state of configuration management.

ANALYTICS OF CONFIGURATION MANAGEMENT FOR SYSTEM ADMINISTRATION

by

Yehuda Aryeh Katz

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Ashok Agrawala, Chair/Advisor
Professor Rajeev Barua
Professor Abhinav Bhatele
Professor Alan Sussman
Professor David Van Horn

PREFACE

Before I ever enrolled in the University of Maryland as an undergraduate student, I was accused of cheating on my admissions materials by the admissions office. Growing up with a twin brother and sharing most interests and activities, as well as sharing a room for more than 20 years, produces two people who are very different and yet in so many ways, the same. Moshe happens to have gotten ahead of me when it came to dissertation-writing, and I even mention his dissertation on page 22 ([56]), but we made a point of not discussing our research with each other. He is allowed to have a chance to be first once in a while.

Moshe, thank you for everything we have done together (and relevant here, for figuring out how to make the dissertation template work in Pandoc). For the rest of you, no matter how similar our work appears, don't even think of comparing us, and we absolutely do *not* copy from each other.

DEDICATION

לה' הארץ ומלואה

ברוך חונן הדעת

ACKNOWLEDGEMENTS

```
cd CAPSTONE
cd AMTAIL
AMTAIL
```

Three simple lines. I memorized those three commands, without understanding exactly what they did, but I knew they would open **An American Tail: Fievel Goes West**, one of the games we had on our first computer, which we got when I was around five years old. Along with the educational Reader Rabbit, and the brainier games of Fatty Bear's Fun Pack, and of course pounding out gibberish in Word Perfect 5, this game made up many of my earliest interactions with our computer. I watched everything my father did on the computer and I learned about directories, modifier keys on the keyboard, and much more. My father made the *mistake* of editing the `AUTOEXEC.BAT` in front of me to make the computer automatically boot into Windows 3.1, so of course the next time it was game playing time, first we changed the computer to boot directly into a game, then we corrupted the file completely. Everything on the computer was fair game and my father would save his work on 5.25" floppy disks to keep it safe (at least that is how *I* remember it). The children's section of the White Oak library had some introductory computer books and before too long, I had also learned where to find technical books in the adult section of the library. It didn't hurt that by then I knew how to use the library terminal system better than many adults. The teachers and staff in the schools I went to recognized my technical abilities, usually in good ways including by providing information about summer opportunities for additional enrichment, and sometimes less so, like confiding that

they were scared of what I would do to the school's computer systems if they tried to take some negative action against me. I learned to always get permission before poking around into someone else's systems, but I never stopped being curious about any system that crossed my path. My formal introduction to computer programming came through a summer course at Montgomery College for 6th graders. We learned Microsoft Visual Basic and I quickly exhausted all the material the teacher had available. I knew there had to be more. In summer 2005, I had my first experience with Computer Science through the incredible *Passport* program at the University of Maryland. I knew then that I wanted to pursue computer science research.

There are many people who have helped me on my path to where I am today. Of course I have to start with thanking my advisor, Dr. Ashok Agrawala, who took me in during a hard time in my research journey and who put up with the many distractions going on around me. Dr. Agrawala was always able to tell me and my brother apart even though we did so much together, going all the way back to serving as my under-graduate independent study advisor to allow me to get credit for working on *WaterShed*, the University's first place winning entry to the US Department of Energy's 2011 Solar Decathlon. Along with Dr. Agrawala, I also would like to thank the other members of my defense committee, Dr. Bhatele, Dr. Sussman, Dr. Van Horn. Thank you to Dr. Barua for stepping up at the last minute, to Dr. Lawson for agreeing to be on my committee even though it didn't work out, and to all the other faculty members who were interested and willing to be a part. I would also like to thank Dr. Keleher who was a member of my preliminary committee. I have a special thanks for Dr. Sussman who would spend hours chatting with me about topics ranging from Computer Science and my research,

to IT operations, University politics, and so much more. There are so many other faculty members who had a significant impact on my undergraduate and graduate studies and research, including Dr. Gasarch who supervised my undergraduate honors project, and Dr. Purtilo for showing the wide variety of research that can be part of Computer science and for always having interesting things to talk about. A special thank you goes to Nelson Padua-Perez for creating the *Passport* program which besides for starting my formal Computer Science education, also gave me an early connection to many faculty members. Thank you to Fawzi Emad, who taught my first class as an undergraduate student and whose pre-thanksgiving lecture about *sizes of infinity* I still talk about to this day. A special thank you also goes to Dr. Mazurek for supporting my research interests (even if our specific projects didn't work out) and for giving me an opportunity to get into human-subjects research - the concerns I learned about experimenting on humans especially in a technical area, as well as ways to properly understand the resulting data, are skills I put to use in many areas to this day. Thank you to Dr. Foster for taking such an interest in my progress as my advisor and as graduate chair for the department. When I was in high school, I used to digitally sign my email and because of that Dr. Jonathan Katz reached out to me to see if I was interested in cryptography and sent me a chapter of his book. I took his class as an undergraduate student and even though my research interests haven't overlapped, he was still always happy to talk about security. I owe an extra special thank you to Dr. Joe Reddish in the Physics department. His unusual (for the time) interactive teaching style really drew me into the class that I was planning on taking just to meet the *CORE* course requirements, and besides for learning physics in a new way, I also learned about breaking barriers in classroom engagement and how great

teaching can make any course exciting. Thank you to Dr. Tim Koeth who helped me to conduct some cross-discipline research of my own. Thank you to all the other faculty members whose doors were always open for me to stop by to talk, whether about my interests, their research, or anything else - I apologize that I can't mention everyone by name.

Thank you to Gerry Sneeringer and Geoff Ransom who encouraged me to poke around and find issues in the university and department IT systems - in keeping with the theme of my dissertation, it would be charitable to call some of those issues *uninformed*. Gerry once said that when he saw an email from me or heard my voice in the hallway, he would have a heart attack - for that I apologize. Thank you to Jeanine Worden for supporting me through so much of my PhD research as well as giving me so many new systems to learn about and so much material to inspire and inform my research. You probably still got the better side of the deal. Thank you to the great CS department IT staff who I worked with, especially Sergey Ivanov and Steve Ryan, and the Business Office staff, especially, Jodie Grey (the only staff member who has been here longer than me), and Sharron McElroy. Thank you also to the DIT staff and IT staff in other departments who were always happy to work with me, often through backchannel communications, sometimes bending the rules, to just get things done - it's probably better for you if I don't name names.

I had great support from other graduate students outside the department, especially Roozbeh Bakhshi, Annie Rappeport, and Autumn Perkey. I honestly don't look back fondly on my time in the Graduate Student Government (where it is believed that I served the longest tenure as an elected department representative in the history of the organization) but I value the personal connections we made.

Thank you to my parents who didn't know what they were getting into when they gave me access to our first computer. Over the years, the collection of old computer parts I would hold on to grew ever larger, as did the number of computer systems I had access to spanning distant family members, and area schools and businesses. My father just said, "Just make sure that no matter what you are doing, I don't want to see you on the front page of the Washington Post." Thank you to my grandparents for allowing me to always use their computers and for providing so much of the aforementioned large collection of old parts, and especially to my grandparents who let me bring technology into running their business, which let me learn so many new things. Some of my proofreaders have asked why I haven't mentioned Moshe: If you started at the beginning, you would see Moshe gets his own special preface. There aren't enough thank yous in the world to give my wife, Elisheva, for her support during this long and drawn out process. Thank you to my children for trying to bother me less while I was working on my school work. Your attempts to read my dissertation are cute.

Thank you to Nechemia Mond and Mordechai Hyatt for giving me my first (paying) technical job. I worked on so many different systems and I learned about how important it is to always have a single source of truth and always understand the limitations of how your systems are configured. This experience had a significant effect on my philosophy of system administration and had direct impact on my dissertation. Of course, thank you for still being around to talk about my research and everything else that comes up in life.

I am appreciative to **An Annotated Bibliography on the Origins of Digital Computers** for historical reference information [89]. Thank you to the "Mission System Execution Lead" at a military contractor who wishes to remain unnamed, for talking to me about

how configuration management works in the Navy.

Can you write an entire dissertation on the command line? Almost, thanks to the work of Dorothea Brosius at University of Maryland's Institute for Research in Electronics and Applied Physics, who maintains a LaTeXtemplate for the university's format for dissertations. This document is written in Markdown and was compiled into a PDF using John MacFarlane's Pandoc [70]. Citations were managed using Zotero [2], and graphs and diagrams were made with Mermaid [87] or a combination of Microsoft Visio and Visual Basic macros. It is typeset in 12pt Charis [1] from SIL International, with Hebrew in the dedication and acknowledgements typeset in Keter YG, created by Yoram Gnat.

Thank you to everyone who put up with me saying over and over again that I can't talk because I am working on my dissertation. I was as tired of saying it as you were of hearing it.

Last, but definitely not least, thank you to God, who has guided me to reach this moment and who is with me always.

אלו פינו מלא שירה כים ולשוננו רנה כהמון גליו.

אין אנחנו מספיקים להודות לך ה׳ אלקינו

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

> *Being a sysadmin is easy. It's like riding a bike. Except the bike is on fire. You are on Fire. Everything is on fire.*
>
> – Unattributed Saying

A regular user of a computing platform sees a collection of hardware and software resources that should facilitate a secure and performant execution of the user's programs. For example, hardware resources might include processor time and memory usage, and soft resources might include encryption functions and input/output functions. The operating system on the computer manages access to and the allocation of these resources based on the requests made by each program. In other words, the user program executes in an environment consisting of some system resources, which are a subset of all the available system resources, created by the operating system at run-time. A simple program might have very simple requirements for its environment, for example, a small amount of memory and access to the file system. A more complex program might have the requirements for the execution environment specified in configuration which accompanies the program.

The configuration of the hardware of a computer system includes the specific pieces of hardware (e.g. the model of CPU and the amount of memory), as well as the way they are interconnected which cannot be easily changed (e.g. the number of memory controller channels or PCI bus lanes). When an operating system is first installed on a particular com-

puter, it has to be "configured" to work properly for that specific hardware. The operating system has a number of configurable elements that are changed to facilitate this configuration, including *configuration knobs* that allow setting particular operational parameters, and rules about system capabilities based on hardware features (e.g. hardware-based network protocol or encryption offloading). On a personal computer, most users accept the default settings for the configuration parameters, and the operating system is designed to provide reasonable default values for that use. On a multi-user server, managing the system configuration becomes more important and more difficult. The server must be able to supply, on-demand, an appropriate execution environment for each program run by each user, while still ensuring some level of security and performance for all of the users and programs. Both the increase in the number of programs being executed and the increase in size of a server over a personal computer add to the potential number of configuration knobs that are available and now need to be considered as part of creating an appropriate execution environment. On such a system, there may be many different configurations that can produce a suitable execution environment, so one must be selected. Individual users can't be trusted to control the configuration of this kind of system, whether because they will configure the system to give a preferential resource allocation to their own programs over other users' programs, or because they lack the necessary training and expertise and will therefore prevent the system from creating proper execution environments for any program. A multi-user system requires a system administrator with the authority and capability of managing the configuration of the system even as its workload changes.

In this dissertation, we study the challenges faced by a system administrator and pro-

pose some solutions in the area of configuration management through clear identification of what constitutes configuration and what constructive steps can be taken by a system administrator of any experience level to better understand the system configuration.

1.1 System Administration

System administrators generally don't make configuration mistakes on purpose and certainly not maliciously, but the impact of such mistakes when they do occur can be just as devastating to the user and the service provider as improper configuration inserted into a system for malicious purposes. (Protecting against a malicious system administrator is a completely different issue, but it is researched in some specific areas [71].) Modern applications have many configuration knobs and even an experienced administrator may not be able to keep track of the current and/or desired values and the side effects of changing them. Mistakes when setting the knobs could, in the case of security issues, allow theft of personal information, identities, financial information, or trade secrets, or in the case of performance issues, prevent legitimate use. Both security and performance issues could cause significant financial obligations, loss of customers, public scandal, and even force a company to shut down.

We call this the challenge of **Uninformed Configuration**.

As an experienced system administrator, I went looking for a structured way to talk about configuration management and to communicate information about system configuration to new staff. To combat the problem of uninformed configuration, it is necessary to understand the structure and interrelationships of configuration knobs and find a way

to make the effects of these configuration changes easier for system administrators to understand. I was not able to find such a system.

1.2 Configuration Management

There are many papers and books published on the subject of configuration management, but most of them are not talking about runtime configuration of a program by a system administrator. IEEE publishes a standards document [127] that explains why configuration management is important and how to effectively carry out a configuration management plan, but it does not define the technical aspects to identify whether any component qualifies as a configuration knob. Such a definition is also missing from the IEEE Standard Computer Dictionary [128]. These standards intend for the definition of configuration to be specified in a technical document created independently and specifically for every system as part of the initial development and deployment of that system. These and other descriptions of configuration management refer to the purpose of configuration management as being about controlling software development and managing changes to code [28] or introduce configuration management processes to decide what configuration items should be part of a piece of software [106]. None of these provide a general definition of configuration that could be used by a system administrator working alone or in a small group who wants a deeper understanding of the systems being used. There are books that tell an administrator how to configure *specific* systems (e.g. Microsoft Active Directory, MySQL Server), but those are very specific to a particular program and often to a particular version. None of this information would help a system administrator

manage the configuration for a different piece of software that was not included in the publication.

For many programs, software developers decide to put in whatever knobs they want, leaving users and later developers to figure out how to manage those knobs. As programs are developed, the number of configuration knobs tends to increase, so when new versions of the program are released, previously published information is also not as valuable. It is very difficult to take knobs away (as we will discuss further in section 4.1), so published material that is specifically targeted at a version of a program will almost certainly continue to become outdated.

Current best-practices for system administrators, as well as skills taught in IT training courses, do not include looking at code or compiling software. System administrators, especially in enterprise settings, are encouraged to use software packages supplied by a vendor without modification other than through the configuration options that are already provided and documented.

1.2.1 Types of Configuration

The purpose of most (non-trivial) programs is to take some input data, process it, and then output it in some way. It would not be practical to develop custom software for every possible use of a computer, so instead systems and programs provide configuration knobs to allow them to be adjusted and adapted for use in different ways.

Every program executes in an environment with limited resources, possibly shared by other programs. The system administrator is responsible for ensuring that each program

runs in a secure, performant, and of course accurate, way by controlling the configuration parameters that are part of each application and by the entire server itself. We can split this into two categories: Server level configuration refers to settings that determine how multiple programs are allowed to use system resources as part of a shared system. Application level configuration refers to settings that apply only to a single program, but could have affects that impact the overall performance or security of the entire system.

There are other types of configuration. When an administrator needs computing hardware (physical or virtual), configuration refers to how a server is built or how a virtual machine is "built" (e.g. which processor, RAM). Once a server is designed for a particular load, it usually is not modified for the rest of its expected lifetime.

There are some specific areas where configuration is very well understood and there is significant work on managing it in an automated way; we can look specifically at High Performance Computing (we discuss this more in section 3.2). The hardware configuration of an HPC cluster is known to the users, who use that knowledge and knowledge about the job they will be running to specify an execution environment through a configuration file that accompanies the program. HPC clusters also require users to submit a time estimate along with their jobs to be processed and the cluster will kill processes that run past their time allocations. If the job time runs out and the user has not created checkpoints to save the processed work when their program is terminated, it is possible to lose a large amount of processed data when the processing is forced to stop prematurely. The processing work being done with an HPC job has to be well understood so that users can create a rough estimate of the time they expect the job to take on their cluster and can tune the parameters of their jobs for better performance. There are too many pa-

rameters for a user to be able to figure out the most optimal configuration for every job, and users usually can only test their parameter choices on smaller computers but not on the large data processing clusters. There is research into automatic tuning of the job and platform configuration to produce high-performing configurations, especially considering how many possible combinations of parameters there are [73]. For our purposes, we call the configuration supplied by the researcher "job configuration", and when we refer to the system configuration, we refer to how the IT staff configured the scheduling software on the HPC cluster. The major difference between these is what can be known about the job or expected load in advance. Job configuration is accompanied with the specific work tasks to be done and the specific dataset the work is to be done on, meaning it is possible to find a single most-optimal configuration. System configuration needs to be able to handle multiple submissions from multiple users at whatever times the users decide to submit, so it is not as easy to create a set of optimal parameters automatically.

1.3 Configuration has not been studied

Practitioners of system administration (IT managers) configure systems as best they can, based on training and experience. Different server environments are very different, so on-the-job training can't be generalized to new areas. We have not found a place where the details of what constitutes configuration has been clearly identified. We could not find any generalizable methodology or evaluation techniques in the literature.

1.4 Contributions

Our goal is to bring scientific rigor to configuration management in a way that can have direct positive impact on system configuration and performance. To that end, we make three specific contributions:

1.4.1 A Definition of Configuration for System Administration

In order to study configuration management for system administration, we first introduce a definition of system configuration which clearly expresses the hardware and software resource requirements of a system. This configuration is distinct from the user program, which only specifies the resources it wants the system to provide. As part of this, we clearly specify what is involved in determining that something is configuration and not part of the operation or data of the program.

1.4.2 A Common Framework to Understand Configuration Knobs

We introduce a common framework to understand configuration knobs and the affects they have on program execution, in the form of the Configuration Impact Graph. We describe several processes to locate knobs based on our previous definition and then a process to create a configuration impact graph. We don't expect to be able to solve all configuration challenges all at once and this process currently involves manual oversight in addition to automated tools, but this is a reasonable first step to allow information to be shared with

system administrators about how their systems work and allow future work to improve on this foundation.

1.4.3 Evaluation matrix/scoring

Having a method to find which configuration knobs are most important is useless without a method to communicate those results. Experienced system administrators often do not trust automated tools to manage carefully crafted or fragile configurations, so we can't just create a tool that will attempt to modify configuration knob values automatically. Being able to understand the impacts of configuration changes requires a method to communicate the configuration information that we discovered in a structured way so that configuration knob values can be compared across different programs or invocations of the same program.

Because this area is complex, we chose to focus on a program running on a server for a client-server architecture, but with a goal of having it be easy to expand in the future. Client-server architecture is hidden under many programs even where you wouldn't expect it, such as being the setup method for long-running websocket connections, and being the way that programs draw a Graphical User Interface on a display.

1.5 Organization of this Dissertation

The rest of this dissertation is organized into the following chapters. In chapter 2, we go through the details of defining what configuration is so that we know what parts

of a program are important to alert an administrator about. In chapter 3, we describe why this problem is hard to study, as well as other historical and current attempts to address security and performance in system operation and configuration management. In chapter 4, we discuss the difficulties involved in understanding the effects of configuration changes. Chapter 5, chapter 6, and chapter 7 explain how we find and understand the actual impacts of each knob. Chapter 8 describes our approach to communicating the importance of each configuration knob based on the possible impacts and introduces the Uniform Parameter Scoring System. Chapter 9 describes how our work applies to production applications. We finally conclude with a summary of our contributions and future work in chapter 10.

Chapter 2: Defining Configuration

Computers are good at following instructions, but not at reading your mind.

– Donald Knuth

```
Configuration -- noun
a. relative arrangement of parts or elements
b. something that results from a particular arrangement of parts or components
-- Excerpt of the definition of `configuration`
   from Merriam-Webster Online Dictionary
```

What is configuration?

Why do we need to analyze and understand configuration?

Why can't an administrator just turn on a system and expect it to `just work`™?

In order to understand how to manage configuration, we first need to define some parameters of what is considered configuration. Depending on your computing background, the term `configuration` might mean different things. One type of configuration is making hardware changes. Otto Schaeffler patented the first easily-reconfigurable computing machine using plugboards and wires for Hollerith company tabulating machines in 1895 [111]. Physical configuration is probably most famous for being a component of the Enigma machine used by Nazi Germany in WWII, where rotors and a plugboard had "programmed" electrical paths, but would be configured in a particular starting configuration to encrypt a message [23]. Modern hardware configuration concerns the selection of processors, memory, storage, and similar factors. Since this work is about analytics of configuration management for system administration, we will focus on software config-

uration from the perspective of a system administrator of a modern software stack.[1] So, what are the essential components of software configuration?

2.1 Knobs

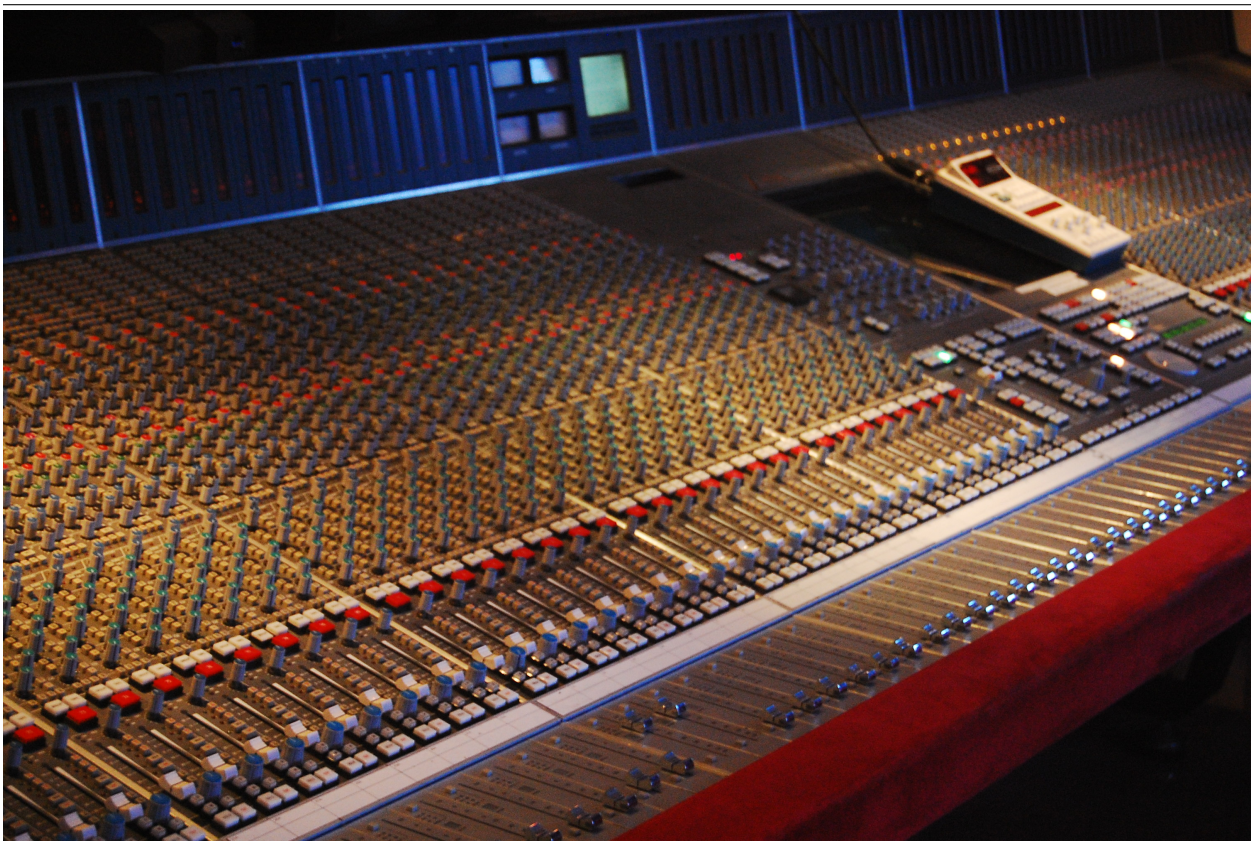
Whether a single hardware server is running a single service or multiple services, each service has its own configuration options that an administrator will need to set to make the program function in a secure and performant way. It might be possible to accidentally stumble upon the correct settings for peak performance and security, but achieving this on purpose, every time, requires that system administrators must be experts on all the settings of all the software they manage.

In the same way that turning a volume control adjusts the loudness of a sound, or a door handle controls the opening of a door, each software setting can also be thought of as a **knob**. Following this metaphor, untrained people look at a professional sound mixing board (e.g. fig. 2.1 [123]) - which may be covered with hundreds of knobs - and understand that proper operation requires a complete understanding of the function of

[1] As we mentioned, hardware has configuration too, but since this is Computer Science and not Computer Engineering, we will assume those values are consistent or not relevant. We are also not considering hardware bugs, or specific temporary conditions that impact proper program execution. For example, the operating system kernel or the storage system driver can get into a state where it believes there is a problem with a storage device and it will put the device into read-only mode to prevent data corruption. The only way to fix this error is to power cycle the system, but in the meantime, many services will appear to have configuration errors because they cannot write to the disk. We also do not include similar management issues such as allowing a storage device to become full, preventing a program from writing data or logs, although it may be possible to configure a program to work despite a full disk.

every part; but people don't necessarily have a similar attitude when it comes to software. We will describe program settings as *knobs*, both as a shorthand term and as a reminder of the complexity hidden underneath.

Figure 2.1 Illustration of knobs



2.2 The Who, What, and Where

Configuration is something which controls how a program executes but is distinct from the `code` or `binary` of the program. A system administrator who is responsible for the configuration of a system is the middleman between a developer who creates a piece of software and an end-user who may not be expected to know anything about the technical aspects of the system they are using. Configuration values are expected to be changed

without any knowledge of the code of the program, and without any expertise or even without any experience at all in the programming languages, libraries, frameworks, etc. used by the program. Configuration values are usually static and do not allow logic, making them distinct from application programming (although they may support `templating` - replacing defined tokens with other values based on the runtime environment or other configuration options). Configuration values may be set in a Graphical User Interface (GUI) or may require editing a configuration file, which may be in a variety of formats. Common formats used for configuration files include `ini`, `yaml`, `json`, and `xml`, although some programs store settings in - often proprietary - binary formats. Some programs support a mix of these and other formats, determining the correct format by looking at the file extension or based on the format of the contents of the file. These configuration files may look like something average computer users might call code if they saw it go by on their screens. All of these methods of configuration require the administrator to understand how to manipulate some sort of code-like file - in the sense that it has a specific structure and the program will not function if the syntax is not valid - but this is not the same as needing to know how to program.

Most software supports one or more of these methods to change the run-time configuration: passing command line arguments, entering values in a configuration file or configuration registry, or environment variables. GUI configuration tools typically modify these settings, but they usually can be changed manually too.

2.3 The When

What parts of software are considered configuration can change over the lifecycle of a system, so we need to be more precise when we determine what exactly we are interested in for the purposes of configuration analytics.

In a simple program, the structure of the program is set at compile time, and operates the same way on each element of the data. Allen describes [7] a control flow analysis that distinguishes the control plane of a system from the data plane of the system and provides a way to ensure the proper control flow is maintained when the compiler optimization *configuration* is changed. This is **compile-time configuration**. Compile-time configuration simply includes any option which is "baked in" and would be difficult or impossible to change once when a program is compiled or otherwise prepared for packaging/"shipping" (e.g. compiler optimizations to remove unreachable code, or byte-code or encrypted interpreted objects such as SourceGuardian [122] or SmartAssembly [121]).

With modern software distribution ecosystems and package managers, it is possible for an experienced system administrator to never need to compile software. (Indeed, when using commercially supported open-source providers such as RedHat Enterprise Linux, self-compiled programs are not included in the support contract and are therefore not encouraged, especially when they directly impact officially supported programs.) Compiled or packaged software of any complexity must provide an administrator with a way to manage the behavior of the program. The configuration options are loaded into memory when the program runs, and they are therefore called **run-time configuration**. This is the type of configuration most commonly changed by system administrators.

We provide an aside about the difference between compile-time and run-time configuration in Appendix B.

2.4 The Why

Why does configuration matter? So what if the configuration values for a program are not optimal, what is the worst thing that can happen? There are a few ways we can categorize the effects of a configuration knob.

2.4.1 Types of Effects

A knob can have the following **types** of effects on the execution of a program:

1. Knobs which only affect a superficial portion of the execution, e.g. a display value:
   If this knob is set incorrectly, it might cause *an administrator or end-user* to say "*Hey, that's weird.*" (e.g. a typo in the contact information shown on an error page)
2. Knobs which affect the output of the program in a significant way, but not specifically the performance or security of the program:
   If this knob is set incorrectly, it might cause *an end-user* to say "*Hey, it's broken!*" (e.g. the user is served the wrong website)
3. Knobs which affect core functionality of the program:
   If this knob is set incorrectly, it might cause *an administrator* to say "*Hey, it's broken!*" (e.g. due to a configuration error, the program crashes instead of running)
4. Knobs which have a critical effect on performance or security:
   If this knob is set incorrectly, it might not be obvious that there is a security issue,

or it might be very obvious that there is a performance problem.

2.4.2 Scope of Effects

A knob can affect different parts of program execution and resource usage/requirements. These can be broken up into subgroups, based on the scope of the effects:

1. Knobs affecting processor usage
2. Knobs affecting memory usage
3. Knobs affecting socket usage (e.g. file descriptor, network access)
4. Knobs affecting input sanitization
5. Knobs affecting other knobs in the same system
6. Knobs affecting other systems

2.4.3 Other Considerations

The value set for a knob may be more critical or require more scrutiny when it is being set, due to external factors:

1. Knobs for which it is difficult to determine the correct setting:

   This could be due to having too many options, options that are too complicated, or not enough information available.
2. Knobs for which it is difficult to change the setting after it was set once:

   This could be due to data being handled based on the value of the knob (e.g. encryption key), or because the knob is only ever used once (e.g. to seed initial values into a database during setup).

3. Knobs which are obscure:

   This could be because the knobs are rarely used, meant for specific purposes such as debugging, being used in a way that was not intended by the original developer, or because they are undocumented.

We will go deeper into the details of the potential effects of configuration later in this work.

Chapter 3: Configuration Challenges and Other Fields

> *Physicists and mathematicians regularly invade other fields but other fields do not invade theirs so we can see which fields are hardest for very talented people.*
>
> – Dominic Cummings

Both physicists and mathematicians come to Computer Scientists for help with really large problems (which puts Computer Science at the very top of the "heap"), but we can still look to other fields to see how they solve similar problems.

Understanding what resources are necessary to complete a computational task has been a field of study since the invention of the computer. With an unlimited budget, it *might* be possible to build a computer powerful enough to complete every possible task asked of it (subject to questions of theory such as "the most important open problem in computer science" [36]: $P = NP$ or $P \neq NP$). Computers that are larger than necessary, besides being a waste of money, are also a waste of other resources - power, cooling, administrator effort, to name a few. Since our major concerns when discussing setting proper system configuration are performance and security, we can look at how these types of issues are addressed in some related areas, and whether those solution can apply to configuration management in other applications.

3.1 DevOps, or "Who needs a System Administrator?"

I think there is a world market for maybe five computers.

– Thomas Watson

The original inventors of computers never imagined that computer use would become as widespread and critical as it is today. This isn't a story about the history of computing, but a brief summary of how the computing landscape got to where it is today will give us an insight into the creation of a critical threat facing modern computing. The well-known quote which leads this section, attributed to Thomas Watson, CEO of IBM, probably paints a picture in your mind of a world with specialized computers, which take up whole rooms and can only be used for specific purposes. Thomas Watson was actually explaining to shareholders that they had expected to lease five units of the IBM 701 Electronic Data Processing Machine on a cross-country sales trip, but they actually leased 18 (at a cost of $216,000 to $325,000 - adjusted for inflation - per month).[1] IBM being blown away by their own expectations was just a start; since then the proliferation of computers and the advancement of the capabilities of those computers has continued to move at a breakneck pace. The "democratization" of computing has led to a need for more people to administer computer systems.

The oldest single-purpose computers, designed for a particular function and taking up an entire room, attended by specially trained technicians in lab coats, have given way

[1]It also turns out that it is impossible to find definitive sources for a lot of oft-quoted lines related to computing and computer science. This is emblematic of the issues that come up when trying to study system administration with scientific rigor. See our discussion of anecdotes in Appendix A for more details.

to commodity server hardware and Virtual Private Server (VPS) hosting which anyone can start running with a single click.[2] The "factory trained" computer operator has gone the way of the vacuum tube, and bookcases that once held thousands of pages of reference materials required to operate a computer are now empty and gathering dust, their books decomposing in a trash heap. System administrators range from career operators with expertise in specific types of systems to people who learned how to run a system from watching YouTube videos. DevOps, the practice of having software developers also serve as administrators for the services they develop, creates a class of administrator not necessarily familiar with the traditional system administrator role. Also falling by the wayside are formerly popular administrative practices of only running a single service on a single server. Indeed, modern computer hardware is so powerful that it can be a waste of resources to run only a single service on a machine (e.g. massive multi-core processors, with optimal RAM installation across multiple channels even using the smallest-available memory modules results in having significantly more RAM than a single service might be expected to use), encouraging administrators to use servers for more than one application. Modern applications are also built out of multiple server components, with discrete services for handling user connections, internal data storage, caching, security, and other functions, requiring a wider breadth of understanding than would be needed for manag-

[2]Interestingly, computer scientists working for IBM in the 1960s claimed that it was wrong to focus on building interconnected or parallel systems and to continue to evaluate the benefits of single processor dedicated systems [9]. This is one of the few examples of something approaching a prediction about the future of computing which we can find in writing outside of science-fiction. It also looks like they were wrong, which may well explain *why* these things are so hard to find in writing.

ing a single service. It may be possible to program the behavior of large and distributed systems as software itself [56], but there still needs to be a knowledgeable system administrator to ensure the services are running in a performant and secure manner.

3.2 Scientific / High Performance Computing

Knight, et.al. define Scientific Computing as "applying computers to scientific problems" [57], particularly the kinds of problems that cannot be solved experimentally. The scientific models used in this type of computing can have billions of independent variables, or large numbers of elements that must all be processed the same way. Large problems in Scientific Computing could include modeling how a vehicle performs when driving instead of building a physical model in an actual wind tunnel, or modeling the performance of a nuclear explosion without destroying a city. The computation is often split into steps, with each step not depending on other completed data processing from the same step, making the step parallelizable. Each step is done by multiple computers, each processing a portion of the data, then the results of the step are combined and processing moves on to the next step.

The calculation of generative artificial intelligence models, such as Large Language Models[3], has recently taken the world by a storm. Generative AI models are both a tool that can help improve scientific computing [41] and a major user of parallelizable computing.

High Performance Computing concerns the efficient execution of parallelizable com-

[3]Hyperactive Auto Correct on Steroids [109]

puting jobs. Allcock, et. al. note [6] that the administration of HPC clusters has the same issues as the system administration of any other group of managed computer systems. HPC systems are designed to be able to handle the types of workloads expected from Scientific Computing and other parallelizable jobs. Users of HPC systems may need to be trained on how to properly use their system [60], but then they can tailor their compute jobs to make use of the specific features of the HPC cluster and can predict the performance characteristics of the job [72, 92]. Users adapt their workloads and estimations of their job time as they learn more about a cluster [93], and it is possible to make very good estimates of resource usage for a job. Indeed, many HPC job scheduling systems require the job submitter provide an estimate of the runtime for billing or resource allocation purposes. If a job takes significantly longer than the estimated time, the job may be stopped, possibly resulting in all progress on the data processing being lost. Users of HPC systems are expected to assemble their entire dataset in advance[4], test small portions of their job to try to understand the expected workload, then scale their expectations for the entire job.

There is work on automatic tuning of job parameters. The Bayesian approach referenced by [73] requires the existence of a possible optimal configuration, and when working in Scientific Computing, job configuration is accompanied with the specific work tasks to be done and the specific dataset the work is to be done on, meaning it is possible to find a single most-optimal configuration. This is significantly different than trying to

[4]For performance reasons, datasets need to be available locally on the HPC cluster. Many HPC systems also do not allow worker nodes to access the network outside the cluster, so worker nodes would not be able to retrieve any other data.

configure a system with a variety of different services or workloads running at once, each with different performance characteristics, and it is also different than trying to model organic use of an application by disparate users, such as web traffic for a public website.

3.3 Translation Layers

Because any given computer hardware has finite processing capacity and data throughput, even tasks on a single computer can be resource-bound. Single-user applications can also still have security considerations. Any computer user familiar with the terminal has at some point had to execute a shell script[5]. Shell scripts are made up of commands to the shell itself or calls to execute other programs. Some shell environments have built in support for high-level programming language constructs including arbitrary loops and conditionals. In other shells, these must be built out of calls to outside executables[6], depending on piped input and output, temporary files, and exit codes. All the commands in a shell script could have been executed manually by the user, but by combining them into a single text file, they can be executed easily as a single command. Shell scripts can also have the benefit of not changing critical environment variables being used by other programs (e.g. path, encoding, file creation mode) and/or not polluting the user's shell environment with unnecessary values.

---

[5]On Linux, often called a Bash script. On Windows, much more commonly known as a BATch or PowerShell script.

[6]In some versions of Bash or similar compatible shells, what appears to be the [operator is actually an executable which will return a zero exit code if the condition passed to it as an argument evaluates to true and a non-zero exit code if the condition evaluates to false.

Shell scripts are often shared and are expected to run the same way across many different systems, meaning it is not practicable to modify a script to take advantage of the specific hardware or environment where it is running. Because shell scripts depend on other executables, which in turn may have varying performance (e.g. speed of decompressing a compressed archive, or downloading a resource from the internet), it may be beneficial to parallelize the execution of the script, but this is not supported by common operating system shells. Shell scripts are also frequently run from an unknown state (e.g. the script can't assume a particular account exists for a service to run as, doesn't know if data is already downloaded or needs to be refreshed, whether a software package is a fresh installation or an upgrade[7]) and have behaviors that depend on environment variables, so it is not possible to statically optimize these operations in advance. The solution to these problems is a translation or proxy layer.

As an example of a performance translation layer, Kallas, et.al. created PaSH[103] to serve as an intermediate layer between a non-interactive script and the shell, allowing for parallelization while maintaining proper execution. This works very well for running a single-user script, but does not help optimize long-running client-server processes.

When using a translation layer with a shell script or program for security purposes, it is usually part of the application development framework (e.g. HTTP request filtering, SQL parameter filtering), or could be built into the operating system (e.g. Windows Virtu-

[7]Shell scripts can even be "abused" as entire software packages. There are a number of large commercial applications (e.g. Atlassian Jira, PaperCut NG) that appear to be distributed as a shell script, but there is actually a compressed archive (e.g. zip or tar.gz) encoded into the file. When the shell script is executed, it unzips the file and compiles it for the system being used.

alStore[8]). Some security-enforcement systems, such as `SELinux` or `iptables`, could also be considered translation layers because they intercept operations by other programs to make sure they meet additional security constraints.

Translation layers provide an important mechanism to control performance and security of applications, but we can't use a translator to manage the configuration for translation layers - these are the exact types systems that we might need an administrator to properly understand and be able to control.

3.4 Anecdotes of System Administration

The trouble with any research connected to the work of system administrators is the prevalence of anecdotes and the lack of data. Entire newspaper columns exist to collect anecdotes of system administrator mistakes and misbehaviors (e.g. the *Who, Me?* and *BOFH* columns in The Register). These tales are almost always told anonymously (or pseudonymously), and while they frequently illustrate a lack of understanding by an administrator who should have a better understanding of the environment and many perfectly describe uninformed configuration, they lack the detail - and certainly reproducibility - necessary to serve as a basis for scientific study.

> An organization was migrating from a proprietary website load balancing solution to HAProxy on several powerful servers that should easily have been able

[8]Windows VirtualStore itself is pretty interesting - it is unfortunate that it didn't become a prominent core security feature, mainly due to objections from programmers and system administrators who didn't understand it. Read more about how it works: [27].

to handle the current load and future growth. The manufacturer of the old load balancer also would not renew the license for the product because it was considered End-of-Life. The plan was to move the websites to the new load balancer individually to make sure the new frontend would not cause any trouble with individual web applications, but the work took longer than expected. The organization realized that they would not finish the move before the load balancer license would expire and the load balancer would stop working, so they moved the rest of their websites during an evening maintenance window the night before the license expired. In the morning, the organization's websites appeared sluggish, but the system resources on the new servers were not exhausted. The organization could not move back to the old load balancers because the license was expired and had to debug and fix the new servers in a production setting. In the end, the entire issue was resolved by changing a single value in the HAProxy settings. This could have been avoided if system administrators were able to see in the manual how critical this setting was.

More discussion of this example, as well as additional examples, can be found in Appendix A.

3.5 Military Systems

The Admiral on a warship at sea gets very upset at seeing a blue screen of death; it truly means death. On a ship, computer system downtime wasn't allowed, ever.

– Project manager for computer systems on a major US Navy vessel [107]

While looking for solutions to system reliability and availability, it is tempting to look at systems where system design and operations can mean the difference between life and death. Along with so many other sectors, warfare is more computerized than ever before. Computers control weapons targeting systems; personnel defense systems; systems for communications and visibility, as well as systems for jamming those systems; all in millions of lines of code running on hundreds of high-performance hardware systems.

Such systems would be expected to be models of ways to avoid issues with uninformed configuration, but aren't generalizable solutions for two reasons. First, many of these solutions are classified or otherwise considered proprietary, and even if they can be used outside of military applications, they would have an associated significant cost. Generic examples of this type of solution could include customized operating systems, customized programming languages, and specialized network equipment. Second, when a commanding officer can throw the system administrator in jail, as well as many other financial, physical, and life-altering penalties (results of NJP or Court Martial), for not following proper procedures when adjusting knobs, it is more likely that procedures put in place to ensure system security and performance will be followed.

3.6 Other Fields

The challenge of choosing the correct value for a knob is not unique to computer systems. Skilled practitioners in other fields are regularly faced with making a decision about a course of action.

3.6.1 Medicine

Good judgment comes from experience; experience comes from bad judgment

– Dr Kerr L White, pioneer in research towards better healthcare services [124]

Maybe if system administrators would slow down and carefully read about every knob several times, they could just figure out the right decision. Moulton et.al. argue in [75] that it is not possible to describe a standard analytical process that an expert uses to make a medical treatment decision. They present a model of medical decision-making and action where slowing down and allowing additional thought produces a better outcome. Further, they point out that less-experienced practitioners often do not know when to slow down and operate less automatically.

The significant difference between medicine and system administration is the ability in the latter field to test decisions and operations and to relatively easily try again. A doctor is not usually given the opportunity to test a medical decision and try again with little or no penalty if it doesn't work. This is a stark contrast to system administration where it is possible - even encouraged - to test configuration changes in a test environment, often without limit. Indeed, in this author's experience, software used for a new service offering may be tested for weeks in advance, with dozens or hundreds of configuration tweaks, restarts, and even complete rebuilds, to make sure it will function properly. Administrators may also believe that they will be able to test changes on a production environment because they can make the changes quickly and no one will notice. Sometimes these temporary or incomplete changes end up staying in production for the life of the service.

While it may benefit a system administrator to slow down and "effortfully" consider the task at hand, our concern is less about taking the time for careful consideration before making an irrevocable decision, and more about having the necessary information to make a decision. Time for careful consideration does not help an administrator choose the value for a setting if the documentation or training are not clear.

3.6.2 Automotive

> *There is tremendous value in physical controls that always do what users understand*
>
> – Article [83] about vehicle control design

The average computer user will likely never be in a situation where their understanding of how to control their computer will put them in a life-threatening situation, but people do put themselves in a dangerous environment every day when they drive a car. Bad user interface design in a car has been responsible for a number of deaths [83]. Alvarez, et.al. suggest a voice interface [8] for vehicles to resolve poorly designed vehicle controls. Bai, et.al. create a model [11] to understand user behavior when faced with a complex dashboard. There does not seem to be significant scientific study of the design of vehicle controls outside China and research papers (or papers from paper-mills) from there are hard or expensive to obtain. People driving a new car for the first time are usually nervous and very cautious until they can get used to the operation of the vehicle. This is a significant enough problem that some airports (e.g. San Diego) have added pull-off areas outside their rental car pickup locations to allow drivers to pull over and figure out how to use the cars they have rented without blocking other traffic.

Outside of a small number of government requirements and ubiquitous standards (e.g. shapes and colors of some dashboard icons, fuel types and fueling methods), and some features that become popular after well publicized accidents (such as automatic parking brakes due to [83]), each auto maker conducts its own industrial research to decide how to make its vehicles safe and easy to use, which in the long run makes cars overall harder to use.

Chapter 4: The Challenge of Configuration Management

> "A computer is like a violin." You can imagine it making beautiful music, but you have to learn how to play it."
>
> – Authorship Unknown[1]

My grandfather calls me regularly for help changing the settings on his oven and refrigerator. Every time, he says the same thing: "You make it look so easy and I don't know how you do it. How can you do it every time and I can't?" It turns out that this doesn't only apply to millennials and senior citizens, and also not only to household appliances, cell phones, and PCs. Despite the commonly held beliefs and presentations in popular culture, being one of the "digital natives" also doesn't necessarily help you with server configuration [95]. With the technical skills that seem inborn to so many these days, the "turn it on and figure it out" approach might work for the cell phones and PCs of today, but "turn it on and figure it out" is not a solution to the complexities of modern system configuration.

As we previously explained, with unlimited resources or with particular mathematic, engineering, and scientific advances (assuming they are practicable, respectively e.g. $P = NP$, availability of electrical power, quantum computing), it might be possible to create a computer system that can handle every possible computation or task instantly. Since that is not possible, due to financial and scientific limitations, complex software applications

[1]This quote is either attributed to Bill Gates, or as a summary of a quote from Marvin Minsky.

need to execute on a platform with limited resources. The knobs set by the system administrator - directly or indirectly - control what resources are requested by the application from the system and which of those resources are eventually allocated. The adjustment of these knobs may have effects that are completely independent from any other knobs or the performance or security impacts may be linked between multiple knobs.

When considering the proper settings for each knob, an administrator must be able to reason about the following criteria:

1. How can we ensure that an application is configured efficiently and securely?
2. How can we manage system resources efficiently to ensure the application will have access to the resources it needs?
3. How can we ensure that the application will remain stable despite changes made to other knobs for the same program or changes made to other programs?

We identified eight specific problem areas when considering a structured scientific solution to understanding the proper settings of knobs and how to evaluate their effects: number of knobs available, process of resource allocation, definition of tasks and system capabilities, connections between control plane and data plane, the relationships of knobs to each other, variability of loads, understanding the actors involved, and a lack of scientific study directed to this area. The following sections describe each of these problem areas and may include an illustration of the issue in a common piece of software or common administrative practice.

4.1 Too Many Knobs

Every time you provide an option, you're asking the user to make a decision.

– Joel Spolsky [97]

Xu et.al. investigate in [108] why software has so many configuration options, or **knobs**[2] and find that in their sample, most configuration options are never used and many options are not used properly. Xu et.al. test natural language processing (NLP) as a way of assisting users in understanding which configuration options are important, but they acknowledge that an NLP-based configuration assistant requires significant engineering as well as developer cooperation in the creation of the user manual for the software. Their proposed solution is for developers to do user studies to understand how users actually configure the software, and then remove unnecessary options, rather than try to make the software as flexible as possible. This proposed solution was not novel for its time - for example, it was suggested by Spolsky [97] 15 years previously - and it clearly was not accepted then as it is not accepted now. As shown in an updated version of the table of configuration knobs for common software, here table 4.1 (Table 2 of the original paper[3]), in the eight years since this paper was released, it is clear that the number of knobs has not gone down.

[2]Incidentally, this is the earliest reference we could find in any scholarly paper to configuration options as "knobs". The paper does not cite any other source for the term.

[3]We were not able to determine exactly how these numbers were arrived at in the original table, but we were able to reach one of the original authors who confirmed that the methods they used were similar to the methods described here.

Table 4.1: Number of knobs in commonly used software packages - based on Xu et.al.

Software	License	Language	Dev. hist.	Version (2015)	Version (2024)	Knobs (2015)	Knobs (2024)
Storage-A	Commercial	–	31 years	–	–	412	[4]
Apache	Open Source	C	30 years	2.2.x	2.4.x	587	707[5]
MySQL	Open Source	C++	30 years	5.x	10.x	461	646[6]
Hadoop	Open Source	Java	18 years	–[7]	3.4.0	312	459[8]

NLP-based configuration assistance has also not taken off. System administrators are able to ask AI assistants such as GitHub Copilot for configuration assistance, but the AI model, at best, "copies" existing published configurations, working or non-working, from other users. A generic LLM (or even an LLM designed for coding like Copilot) doesn't produce guidance on how to properly set each knob for each unique situation. Of the solutions mentioned in the paper, the one recognizable to most administrators is the simplest: ask Google.[9] In my personal experience, searching Google to learn what changes need to be made to software configuration is often an effective way to at least get a pointer[10] to *a* way to make the desired change (although not always the best way). Also in my experience, AI GPT tools are more likely to provide incorrect answers.[11] (As of this

[4]Not available, ostensibly a NetApp proprietary storage system

[5]Number of instances of `<directivesynopsis>` in the XML for the Apache 2.4 documentation

[6]MariaDB (fork of MySQL) 10.11.7 on Windows: Line count from variable output of `mysqld --verbose --help`

[7]Original version not specified

[8]Number of times `<property>` appears in the `hadoop-common/core-default.xml` configuration file included in Hadoop 3.4.0

[9]See more about what can be known about the prevalence of administrators searching for help using Google Search in appendix A.1.

[10]Usually one of `0x3A28213A`, `0x6339392C`, `0x7363682E` [76]

[11]And avoid using Google's new AI search results for anything. As of this writing, Google provides blatantly

writing, AI-based answer engines are too new to have significant research into how much users trust them, but there is ongoing work in this area [51, 102]. This isn't to say that user trust is a good measure of the validity or optimality of a solution, as is evidenced in Acar et.al. [3].)

Why don't developers simplify the available configuration options and restrict the flexibility of the software so that it is easier to configure and manage?

Sometimes it comes down to how the software was originally created and released. Publicly released software can generally fall into two categories. The first kind is software written by a developer for their own use and released publicly just in case someone else might want to use it. Even when such software is open source, developers will frequently reject any request for changes that doesn't match their use case, even when the request is accompanied with functioning code (e.g. a Pull Request in GitHub). This has been this author's experience with a number of software packages commonly used in academia, although none will be named here. The second kind of software package is software originally designed and intended for public use. This kind of software is usually designed to be as flexible as possible, sometimes to the point of completely changing the mission or vision of the original developer or designer. This kind of software is often released altruistically as open source. Obviously, any change that would restrict the flexibility or usability of this kind of software, when its mission is to be useful to the public, is not going to happen, even at the expense of significant technical challenges. Examples of this in the author's experience include maintaining backwards compatibility in Puppet

incorrect information when faced with practical questions - although if you enjoy a pizza with glue on it, go for it [68]. Other AI GPT tools are just as bad [67, 112].

modules, extensions and configuration options in Apache HTTPD, PowerDNS, and many more systems.

Xu et.al. also found that developers include configuration options if they can't decide on a reasonable default value. Setup wizards for many desktop programs are full of choices that the user needs to make - even if they don't understand the significance, or know that they don't care which option is better - because several developers couldn't agree on the best way to do something [97].

4.1.1 Complex Knobs

Sometimes many options are provided for backward compatibility or to allow phased improvements. For very complex settings, it may never be possible to know what the best configuration option is without a significant amount of effort, but it is also not possible to hard-code the best possible settings, due to backwards compatibility requirements or cost reasons.

The most common interaction of the average user with public-key encryption is the use of secure hyper-text transfer protocol connections (HTTPS), wrapping the plain-text HTTP protocol with Transport Layer Security (TLS, but still commonly referred to as its deprecated and insecure predecessor, Secure Sockets Layer (SSL) or sometimes SSL/TLS). Many web server administrators are familiar with the process of obtaining SSL certificates, and will recognize that the security of the connection may depend on the size of the private key. TLS connections have other parameters that are important to make sure the connection is secure, such as the version of the SSL/TLS protocol, the specific Cipher

used, and other connection options such as compression and heartbeat settings. The base SSL/TLS library used by the webserver may support thousands of cipher combinations, but only a small number of those are considered secure. Older client devices may only support older cipher options, so by default, many server programs ship with less-than-completely-secure settings to allow access from older client devices (e.g. very commonly older Android phones[12]). If the system administrator decides they want to change the ciphers being used, they will need to sort through a large "wall of text" to see what ciphers are allowed and they will need to do research to decide on the appropriate cipher list (or copy a list from a Google search and hope it is accurate). Creating a secure HTTPS configuration is hard [15]. Reading an article high on the Google search results for "secure SSL cipher" shows how complicated this can be [48]:

> Disclaimer: I'm updating this post continually in order to represent what I consider the best practice in the moment – there are way too many dangerously outdated articles about TLS-deployment out there already.
>
> Therefore it may be a good idea to check back from time to time because the crypto landscape is changing pretty quickly at the moment. You can follow me on Twitter to get notified about noteworthy changes.

and later

[12]In Android (up until recently) the SSL libraries were part of the operating system and could not be upgraded separately, so when a device would lose operating system support, its SSL functions would also be frozen in time to the last available protocols, ciphers, and root certificates.

> There used to be a bullet point suggesting to use RC4 to avoid BEAST and Lucky Thirteen. And ironically that used to be the original reason for this article: when Lucky Thirteen came out the word in the streets was: “use RC4 to mitigate” and everyone was like “how!?”.
>
> Unfortunately shortly thereafter RC4 was found broken in a way that makes deploying TLS with it nowadays a risk. While BEAST et al require an active attack on the browser of the victim, passive attacks on RC4 ciphertext are getting stronger every day. In other words: it’s possible that it will become feasible to decrypt intercepted RC4 traffic eventually and the NSA probably already is. Microsoft even issued a security advisory that recommends to disable RC4. As of 2015, there’s also an RFC.

In short, known bugs and design flaws, changing security requirements, and the ever increasing processing power of modern computers make the proper configuration for SSL ciphers a moving target.

Let’s say that an administrator knows what each knob does - how do those knobs interact with the system itself?

4.2 Resource Allocation

What kinds of resources do programs use? When people talk about the resources required to run a program, they are usually referring to processing power, traditionally, the CPU clock speed which indicates the number of operations a processor can complete in a second (modern CPUs are usually measured in GHz - billions of operations per second).

Modern scientific and gaming workloads are also concerned with GPU processing power (modern GPUs are usually measured in teraFLOPS - trillions of floating-point operations per second). A system may have multiple CPUs and GPUs, as discretely installed chips or cards, or as multiple "cores" sharing a single piece of silicon. Modern GPUs have multiple (often hundreds) of parallel compute units on one card, but they can only handle certain types of simple operations. When a system uses multiple processors to achieve a higher raw processing capacity, it actually creates additional performance considerations as information has to be transferred between the processors. Other limited resources that need to be considered include the amount of memory: processor caches (at multiple levels), physical RAM, and the amount of interconnection bandwidth available between them, including direct traces between the processor and RAM and how many controller channels there are which could allow parallel access. Multiple processors on a single piece of silicon may share processor caches, allowing all the cores to benefit from loading shared instructions or data a single time. Specialty links between GPU cards (e.g. Nvidia SLI, AMD CrossFire) exist to allow this type of faster sharing between discrete processors, but they were mostly designed and tested for reduced lag when playing video games; they do not typically provide performance improvements for non-game workloads[13], and they are no longer developed. Another complicated resource is virtual RAM (also called `swap`) and whether a program is even allowed to use that resource (e.g. some security software restricts the kernel from moving sensitive information into swap because it could allow that information to be recovered later off the disk). The media type and speed of long-term

[13](and often performance could be worse!)

storage (e.g. hard disk drives[14], solid state drives, network-attached storage) affects the time to load a program (the first time if there is enough RAM to hold the entire program, continuously otherwise) and the time to access and write data. As soon as a network of multiple computers is involved, the available resources as a total amount increases, but the amount of network bandwidth and the increased latency of those connections add additional limited resource considerations.

All the components of computers themselves can also break down over time, resulting in worse performance and fewer options for resource availability [21].

- Each application requires some resources, which are allocated by the operating system and the network.

[14]When it comes to magnetic hard drives, the speed of accessing data can actually change based on how long the drive has been in use for - another factor that makes modeling this type of performance in the real world very difficult. There are hardware reasons that magnetic hard drives are slow: spinning platters inside a hard drive can only be accessed when the read/write head is in the correct place and that can only happen at specific times as the platter spins. The read/write heads inside the hard drive need to seek to the correct location too and can be affected by environmental factors such as vibration. Hard drives are also vulnerable to fragmentation - data is written to the drive in chunks, but individual chunks of a single file do not need to be stored sequentially. Instead there is a table of contents that tells the drive where to locate the chunks of any particular file. When a disk is new, it is more likely that there is space to save all the chunks of a file sequentially, but as the disk is used and files are deleted, empty areas may not be large enough and a single file will have to be split into multiple locations, or fragmented. This slows file access significantly because multiple seek operations are required to access a single file. While Solid State Drives have the same issue with fragmentation, because they have no hardware seek time, it is less noticeable.

- The operating system has a scheduler that prioritizes the resources requested by each program from the kernel.
- The CPU has a scheduler that prioritizes the requests from the kernel against hardware interrupts and other internal functions.
- The knobs set by the system administrator - directly or indirectly - control what resources are requested by the application from the system and which of those resources are eventually allocated.
- The adjustment of these knobs may have effects that are completely independent from any other knobs or the performance or security impacts may be linked between multiple knobs.

Some settings can have no effect on overall system performance and some can cause the entire system to stop working due to resource exhaustion/oversubscription.

Before we can know how resource allocation will work, can we even know what an application really needs to complete its tasks?

4.3 Software Tasks

We can't understand what resources will be allocated for a task by the kernel and CPU without knowing as much as possible about the tasks themselves. Some tasks have fairly obvious resource limitations associated - for example, only one process can write to a file at a time to prevent file corruption, so if there is a chance that multiple tasks will need access at the same time (whether in the same part of the program for parallel users or two different parts of the same program for the same user), there will need to be some

kind of locking system which will impact performance. At a deeper level, each CPU core must have the necessary instructions and data to operate and that could be impacted by limited bandwidth between the CPU and various level of data storage (Lx caches, RAM, durable storage, network). CPU schedulers can potentially anticipate needing to wait for data for a particular operation and run other operations in the meantime.

We will represent a lot of these relationships by graphing the execution paths of the program, but first we need to understand what a task is. Do we stop our analysis any time a call is made to a kernel function, any `stdlib` call, a call to any linked library, some other high-level operation? Depending on how deep into the execution we take our analysis, we might be able to label parts of the program by whether the CPU will be required to wait for resource. Before we can decide what is considered a task and do any of that labeling, we first need to know: How can you separate the programming of a task from the data used by that task?

4.4 Control Plane and Data Plane

There are very few programs that take absolutely no data input, whether from the user, from the environment, or from other systems. Because of this, even if a program is running on an isolated system with no need to share resources with other programs, it is not possible to look at a static analysis of the program and know exactly what code paths will be taken because the actual execution will be impacted by the data specific to the particular instance. This is especially true if we already know that the code paths will be changed by the request or response data.

To work around this, we separately study the control plane of the program and the data plane of the program. The control plane decides how data is handled and the data plane does the actual handling. Network switches generally have this split baked into the hardware architecture, with data plane hardware designed to quickly process network packets using a limited set of rules, and control plane hardware setting up those rules.

Even though most software isn't built with this terminology, we can look at a "virtual" control plane and data plane for many kinds of programs to try to understand their operation. A task that must be completed by the program is created in the code of the program, subject to the configuration parameters set by the system administrator. This can be considered the control plane. The data used for the task, flowing from the user or environment input, through the task and back to the correct output, is part of the data plane. It can be difficult to distinguish the control plane from the data plane in many programs, so we need a method to clearly separate what parts of the program operation are part of the control plane - managed by the program configuration - as opposed to part of the data plane - completely connected to the user's request. We can then label different parts of the program based on what data they are using.

Once we have this method to identify and label the parts of the program, how can you model the control and data flow for a variety of uses to actually understand the performance and security of the program?

4.5 Relationships between Knobs

Some program settings are entirely standalone knobs, with no impact on any other part of the program operation. Some settings theoretically should never affect other operation of the program, but have unforeseen interactions or trigger obscure bugs. Some settings by design will have obvious impacts on other settings.

An example at one extreme of knob relationships is a setting that changes a simple display option. A setting which changes the text displayed in a particular place in the program without affecting the program control flow may have a miniscule effect on performance or security and is not expected to affect any other knobs. A setting like this should have no connection to the values of any other knob, but it still cannot be determined to have absolutely no effect on program execution, because for even something as simple as a text option which is displayed with no modification, the text needs to be copied between different buffers in memory or written to the output destination (i.e. file, network, display). If this utilizes a variable length buffer in memory, performance may be impacted by the time it takes to read or write the buffer may change significantly in CPU cycles even though it does not change in algorithmic complexity. If this utilizes a fixed length buffer space where the entire buffer is accessed every time, security may be impacted (depending on the programming language) because it can lead to buffer underrun or overrun errors, performance will always be as bad as the worst case scenario, and administrator choices for the value of the option will be constrained to the maximum size chosen by the developer.

An example at the opposite extreme of knob relationships is a setting that serves as a

condition controlling a loop. A setting which causes the program to execute in a gated section of the code or a loop, besides for the potential of significant performance impacts due to the potential to change the algorithmic complexity exponentially ($O(n^x)$), can cause the program to read the values of settings which are otherwise not used. For settings that explicitly control requests for resource allocation or process management (e.g. forking, threads), values of these settings may be multiplied together when allocating system resources, causing significant changes to available processing power and memory usage.

A setting which controls whether the program will read other settings can affect the security of the entire program or server. For example, the Apache HTTPD server includes the `<IfModule>` directive in its configuration which allows certain parts of the configuration to be entirely skipped if a shared library (`.so/.dll`) is not loaded. When the server will not start because there is a module that is supposed to be loaded that can't be found, an administrator might try to make the simplest fix: remove the directive to load the module. If there are important parts of the configuration that depend on that module being loaded, it could lead to sensitive information being released or other significant impacts to the program operation. Interacting knobs can also have affects that are completely accidental due to the specific configuration chosen by the administrator. Using the same `<IfModule>` example, the HTTPD server doesn't care whether the settings inside the conditional part of the configuration are actually related to the module; any settings can be inside any `<IfModule>` directive. If an administrator neglects to end the conditional section of the configuration (using the corresponding `</IfModule>` directive), it is possible for the entirety of the rest of the server configuration to be skipped - even across multiple included files - if the specified module is not loaded.

Administrators are expected to have some level of understanding of their workload and technical ability when it comes to setting configuration knobs. What else can we understand about the people and the workloads?

4.6 Variability of Loads

> *Of course you've probably heard that healthcare.gov hasn't worked as smoothly as it was supposed to work. The number of people who visited the site has been overwhelming, which has aggravated some of these underlining problems. Nobody's madder than me about the fact that the website isn't working as well as it should, which means it's gonna get fixed.*
>
> – President Barak Obama, talking about the botched [99] rollout of Healthcare.gov

Because computer systems have limited resources, there is a limit to the number of concurrent users who can use any particular system at the same time. Many systems are tested for functionality first, with performance testing often as an afterthought or never even happening at all [19]. When a website opens for sales for a popular concert or for enrollment in a new government assistance program, the load can be expected to be very high, but just how bad will performance be and what can a system administrator do about it?

The Federal Emergency Management Agency provides emergency relief services to people and communities impacted by natural disasters. They need to collect information about people affected by the disaster and process it in a timely manner, often without

significant advanced notice. For most of the time - when there is no new major natural disaster - there is almost no load on the web application, but as soon as people regain internet access, the first stop for many of them will be FEMA's website. In orders of magnitude, a web application could be expected to handle anywhere from one user per month up to thousands of users per second. The ability to design software that is capable of this level of performance falls on software designers and developers rather than system administrators, but an administrator has to know how to configure the software properly to minimize unnecessary resource use and handle the expected load. There are two ways to understand how a program is expected to perform under load: instrumenting and modeling in advance - likely the province of software developers rather than system administrators, or live testing and tuning.

4.6.1 Instrumenting and Modeling

Famed German physicist Werner Heisenberg is quoted in popular culture as the person responsible for the concept that observing something makes it change.[15] Physicists continue to argue [37] over whether the observer effect applies to quantum mechanics, but it does without question apply to software performance and security testing directly on applications. When there are a limited number of available CPU cycles to execute instructions, additional instructions asking the processor to also record instrumentation information have to impact performance in some way. Many processors include a small number of

[15]The *observer effect* is actually a more general effect which can even been seen in simple every-day operations like checking the pressure in a tire which lets a bit of air out and so is not an accurate representation of the pressure before the measurement.

registers that can be used to store performance counters directly on the CPU to make the performance impact as small as possible, but due to this limit multiple measurements may be required to get all the desired testing done. There are also tracing methods (e.g. `perf`, `dtrace`, `eBPF`) that hook into the operating system kernel to provide low-impact modeling and tracing features, but using these effectively requires specific training and is usually a job for programmers, not system administrators. There are regular improvements in the performance of program-modifying instrumentation techniques for general programs [22] and for performance in scientific computing [12]. Modifying a program to instrument its performance can affect security monitoring software, especially systems that monitor the "dirtiness" of user-supplied information or that look at the behavior of a program using heuristics.

4.6.2 Load Testing

Sometimes there is no substitute for testing environments or live tuning, especially for a system administrator who does not have access to the code or appropriate instrumentation tools. Creating a load testing setup separate from the live environment presents its own challenges, mainly due to difficulties creating an environment that is similar enough and generating a load that will be similar to the live load [39]. (Sometimes this type of testing also reveals that administrators need to take a step back and rethink basic assumptions or design decisions about the system [53], but that kind of change it outside the scope of understanding configuration knob changes.)

With a modeling and testing framework and enough hardware can test every position

of every knob?[16]

4.6.2.1 Extreme Dummy Loads (or "Bees with Machine Guns"/"Chaos Monkey")

One way to test the performance of a system is to test it "to destruction". A single client might not be able to create enough traffic to test the system or the system configuration may cause all traffic from a single client to only hit certain parts of the program (e.g. in a load-balanced setup). The phrase Bees with Machine Guns was coined by the Chicago Tribune for a script that would create a self-targeted Distributed Denial-of-Service attack, with many small workers attempting to access their service all at the same time. If their service passed the test, they would assume it could handle any legitimate load. This testing approach is usually limited to testing a specific endpoint and does not test an entire workflow or path that a user might take through the site.

Another way to test a live system is to cripple the system internally and make sure everything still works as expected. Chaos Monkey [25] was a project by Netflix that would pick random AWS EC2 instances to power off and then make sure that Netflix services were still healthy. Once some systems were powered off and all services continued to work, you can assume that the entire system has a tendency to stay stable even with some failures.

Both of these types of testing require some tolerance for disrupting production services in a controlled way.

[16] In keeping with `Betteridge's law of headlines`, the answer is "No".

4.6.2.2 Cloud Provisioning

Another way to deal with load testing is to never let the load on the application get over a predetermined metric. This strategy requires monitoring the performance of the program and scaling the available resources to meet the current load. For example, the Marmoset project submission testing software used in this Computer Science Department is set to start up additional servers for submission testing if the time any submission has been waiting goes over a particular preset value, then shut down those servers after they have been idle for a set amount of time.

There are two different strategies that a system administrator can use to be able to serve loads that are larger than a single server can handle: horizontal scaling and vertical scaling. Vertical scaling means adding additional resources to a system to be able to handle a greater load. Vertical scaling usually does not require making changes to an application or configuration, unless there are specific performance-related configuration options, however for many applications which use separate processes for database servers, application servers, caching, and other functions, the administrator will need to understand where their performance bottlenecks are and decide how to allocate resources to the different server programs on a single machine. Vertical scaling can also become more expensive as the costs to purchase higher-performance hardware grows. Vertical scaling does not provide any other benefits besides performance improvements and configuration simplicity; for example, it does not help with high availability and there is a limit to how much hardware can be used as part of a single machine. Horizontal scaling means splitting a load across multiple machines. This could include each machine running the same

services in a replicated or eventually consistent [104] arrangement. This could also include separating different types of loads (e.g. load balancing, application server, database, caching) across multiple servers, each with its own high-availability, replication, or eventual consistency configuration. Farming out individual components to hosted services (e.g. using Amazon S3 for static file hosting) can also be considered horizontal scaling, both because it is spreading the load and because cloud Platform-as-a-Service offerings are themselves horizontally scaled, often also using an eventual consistency model [58].

Both of these types of scaling can be accomplished through Everything-as-a-Service (XaaS) providers, more commonly known as "The Cloud". Vertical scaling in the cloud can be accomplished by renting a larger Virtual Machine. Depending on the specific technologies used, it may be possible to dynamically add and remove processing power, memory, and storage resources from a VM while it is running, but usually a restart is required for VM size changes to take effect, so this is not a good solution for handling variable loads.

If a workload is really known to be variable and is horizontally scalable, it may be a good candidate for renting time on someone else's computers. The specifics of horizontal scaling and eventual consistency vary across applications, cloud providers, and even different offerings by the same provider, so the specific needs of each situation and tech stack must be evaluated. "The Cloud" is not a magic solution to variations in load [46]; each use must be evaluated to determine whether the cloud is appropriate, and specifically what strategy to use to get the most out of the available resources [52][17]. A

[17]There is even an ongoing conference workshop which has a focus on evaluating the applicability of cloud services to traditionally-local workloads [64].

system that would allow an administrator to fully understand the available knobs and tune the settings would allow more informed decision making and cloud service selection or infrastructure design.

4.7 Actors in Configuring and Running Systems

> *If debugging is the process of removing software bugs, then programming must be the process of putting them in.*
>
> -– Edsger Dijkstra

> There are two hard problems in programming: 0. cache invalidation, 1. naming things, and 2. off-by-one errors.
>
> – Based on a quote from Phil Karlton

There are three different roles for human actors when it comes to the final execution of a program. Software developers, or programmers, are responsible for creating the software that everyone else is using. End users of a program are not expected to know anything about how the program works. On personal computers or mobile devices, the end user is likely going to be responsible for setting all the configuration values for the device. Sometimes the user knows how to do this properly, or the system is designed to walk the user through to help make the right choices, but other times, the user just blindly guesses or leaves all the settings at the default values. System administrators are expected to be experts on the systems they administer and serve as intermediaries between the developer and the end-user; configuring the program to serve the end user in the best possible way.

It is tempting in a system research context to consider a program in a “clean room” situation - dissecting the program and its configuration knobs without considering the humans behind it, but even without a Human-Computer Interaction focus, there may be something we can learn by making sure we understand the human component of configuration.

There is a lot of research into how to make programmers code better for performance and security. For example, there are safeguards in compilers to prevent dangerous behavior, and there are recommendations for programmers to use only memory-safe programming languages [30][18]. Acar et.al. measure [3] how programmers get insecure (or non-performant) programming advice from the internet and continue propagating it through new programs. For our discussion of configuration, we have to assume that the programmers have not made significant mistakes and that every part of the program works as documented (or as the programmer intended).

There is also a lot of research into end-user behavior, primarily related to security and usability, but often dismissed as less of a science due to the influence of psychology and the study of humans rather than machines.

It is not a coincidence that system administrators are also referred to as **operators**. We are familiar with an operator of a piece of heavy machinery, an operator of a piece of public infrastructure such as a water treatment plant or a telephone switchboard, and an operator of a business such as a tour operator. Although that term seems archaic now in relation to computing, the operator of a system is the person responsible for safe and

[18]and rebuttals from language creators [59]. Additionally, the White House report has been challenged on other grounds [84].

efficient operation. Is there any better description of the role of a system administrator?

Configuration changes that impact performance and security aren't necessarily bugs. Developers design software to be flexible, and the ability of an administrator to choose a less-performant or less-secure configuration is part of the design. For example:

- an administrator may choose to configure a less-secure option to be compatible with older software.
- an administrator may choose a setting which disables caching - and is therefore less performant - to allow for more flexibility.
- an administrator may choose a setting which prefers using computational power rather than additional storage (or vice versa) in order to save on cost of operation.

4.7.1 What Drives Changes

When a system is working, why would a system administrator touch it? Sometimes there is a change in operational requirements. Sometimes a software update. Sometimes an administrator just finds that the initial configuration wasn't optimal.

Uninformed configuration can happen in two ways: due to external factors driving an administrator to make a change, or due to an incomplete understand about the system (which we will call *internal factors*). Externally, administrators may be forced to make untested emergency changes or workarounds with unintended consequences for business-process reasons or to meet a mandate issued by management. Some companies attempt to prevent this from causing an issue by not allowing untested changes, but strict rules can cause their own issues when they get in the way of fixing issues with production

systems. Internally, sometimes the default configuration values appear to be sufficient with a casual inspection, but issues may be hidden in large walls of text in documentation or configuration example files, or may require some unexpected domain-specific knowledge to recognize. Complex situations can also result in settings changes that impact performance in unforeseen ways. Some settings may cause unexpected load on a system or break compatibility with older components. Either of these types of situations can cascade quickly and make it difficult to understand what change actually caused the problem.

4.7.2 Software Complexity and User Expectations

Workloads are different between systems administered by a system administrator and designed for commodity (i.e. every-day, unremarkable) use by end-users, compared to distributed computing, where the user is expected to know a lot more about their workload and the appropriate way to process it. End-users do not understand the complexities of system administration or programming. Users see something “shiny” and want to be able to have it themselves, but don’t understand how it fits in to the overall puzzle of their existing systems and applications. Users search for information on the internet and expect their system administrators to be able to implement it. Managing systems to serve a few users is different than managing systems to server a few hundred users, a few thousand users, etc.

4.7.3 Teaching System Administration

In my own educational experience, system administration tasks are routinely bypassed in Computer Science courses. Students are encouraged to debug their code on the same system or cluster the instructor used to create and test the project. If the student's code does not work due to a system difference, the instructional staff will often not look at the problem unless it is reproducible on the "official" system, so students do not learn about the effects of configuration differences between different systems. If students will need to work with a system that is configured in a particular way but that cannot be done on a shared system (such as tasks that requires disabling security software or kernel modifications), instructors will distribute a virtual machine that has already been configured appropriately. In classes that have been taught many times, that virtual machine has likely been configured by an instructor or grad student many years before and has been passed on - but never changed or updated - as new instructors teach the course. When courses or research do cover configuration management, they are often targeted at IT students/practices rather than at Computer Science students/practices, and a survey of such courses and research (e.g. [20, 66]) also shows that they mainly focus on the methods for managing configuration changes, but not how to know what knobs to actually change or analyzing the effects of such changes.

4.8 Basic Concepts, Communication, and Basic Research

For many people, as soon as they tell someone that they are studying Computer Science, they are immediately asked, "Well then, can you fix my computer?" There is a difference

between an education in Computer Science which allows us to understand the designs and mechanisms of computing, versus the IT technician who operates or repairs computers systems, or the person who learned to code from a quick bootcamp, but doesn't understand the concepts. The divide between scientist and technician is evident in the practice of Computer Science research. Even Computer Science researchers who work on theoretical problems have likely learned to code at some point (writing in LaTeX is its own kind of challenge), and they have almost certainly been the end-user of a computer or smartphone. Because these are areas that are familiar to researchers, there is a lot of research on improving the security and performance for software developers, and there is a lot of research on the experience and behaviors of end users. Most Computer Science researchers have never been in the position of a system administrator of a large network in a production environment, so there is a corresponding lack of published scientific research from that perspective.

There is a point in the study of any field when people think there isn't anything basic or game changing left to be discovered. In observed (real) sciences[19], there are still earth-shaking discoveries every so often, such as a previously-unrecognized entirely new organ in the body [14, 91] in 2018. There are no hard and fast rules for when a field has reached a level of maturity where people believe that there are no foundational concepts to discover or understand, or put another way, no more low-hanging fruit.

The Department of Computer Science at the University of Maryland recently celebrated its 50th birthday and I have had some connection with the department for almost

[19] as opposed to engineering-based "sciences"

20 of those years. In all of that time, I have seen a very clear delimiting line between researchers and computer operators. There are always faculty members who are interested in managing their own systems, sometimes with the approval of the network and datacenter operators and sometimes explicitly against their wishes. There are not many people involved in basic research who are also experienced system administrators, and especially not of large production networks. For many researchers, IT staff are there to support the hardware that supports the research and education missions of the university, but do not have a role directly in the research. For other researchers, the connection between the role of the operator and the technology is to be left for those who are interested in Human-Computer Interaction. Other than basic helpdesk tasks, computer operators are usually not expected to be able to teach how systems work or communicate scientific ideas connected to their theory of operation. Some faculty members even view the system administrators adversarially and assume the system administrators are getting in the way of their legitimate research work for no reason. While entire books could be written about the relationship between computing research and centralized IT departments (as well as lawyers interested in protecting the university from liability), I think it is fair to say that there is a lot of space to discover and develop novel concepts and completely new ways of viewing the science at the intersection of Computer Science and Configuration Management. More than once I have started explaining research in this area to someone else in this field and been told that it doesn't seem to be a legitimate line of inquiry because there are very few existing published papers that are related and therefore there must not be anything worth pursuing. While this specific work may not be the earth-shaking discovery of a new organ, we are looking at a new angle to study

configuration management.

4.9 Problem Summary

We have described why there are so many configuration options in software programs, why it is difficult to understand which configuration options are important, and why there is so little research into how to better solve this problem.

We will break this down into a handle-able problem by considering a webserver.

4.9.1 Why a webserver?

A prerequisite to any type of communication is an initial agreement between the communicating parties of the standards they will use to communicate. In the physical world, this could be an understanding of what language to use and whether the communication will be verbal, written, or visual. We may make certain starting assumptions, such as speaking the native (or most common) language of a country/region, or that all parties are physically capable of using the chosen communication mechanism. Modern high-level programming will allow a developer to use asynchronous or long-running methods of communication, such as multicast traffic, websockets, server-sent events, or message queuing, but at their core, all of these methods rely on a synchronous (in-band or out-of-band) agreement between the parties. We therefore consider all of these technologies to be extensions of a standard client-server communication. One of the simplest forms of client-server communication is the webserver. Traditionally, a client will send a plain-text request for a single web resource and the server will send back a plain-text response with

the requested data. A web request will include additional metadata about how the plain text request should be interpreted, and a response will include plain text headers that indicate if the response should be handled differently, for example if the data streamed will be a document, image, or video. Because all types of inter-system communication can be simplified to a client-server interaction, and because web server client-server interactions are familiar to most technical readers, almost all of the practical examples we will discuss here are taken from different types of webservers.

Chapter 5: Inside the Knobs: A Configuration Impact Graph

If configuration knobs are so critical to performance and security, and administrators don't know how to properly set them, and none of the existing solutions really resolve the issue, what else can we consider? Software developers can write specific documentation for configuration entries that they feel are important or have far-reaching effects, but this is not a scalable solution and is also prone to human error. We need to have a system that can identify and then a way to communicate how these knobs work and how the knobs interact with each other, and this system must be able to work without a significant amount of manual intervention.

5.1 Step 0: Program Design Patterns

By our definition of configuration from chapter 2, configuration options are set when a program execution begins and are not changed by user input during the program execution[1]. (We will discuss situations where a program has multiple sets of configuration options which may be selected for a particular user interaction based on user input as

[1]Most server programs only read their configuration when they are started and the running configuration will not be updated unless the program is restarted. Some programs may provide a mechanism to reload the configuration without restarting the program, such as an internal command or a predetermined signal from the operating system/service manager. A program could technically load configuration data every time the main loop is executed, but that would be very inefficient and could lead to unexpected behavior (based on accepted design patterns).

part of our look at Apache HTTPD in section 9.1.)

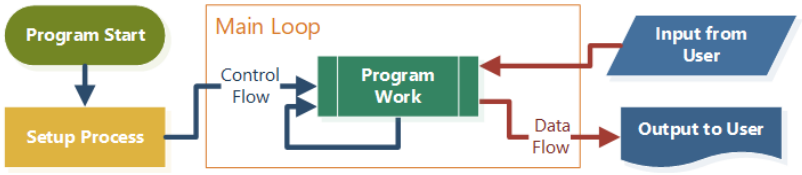
The lifecycle of the execution of a client-server program has three phases: the service phase, when client requests are accepted and responses are served; preceded by a setup phase and possibly followed by a tear-down phase. An example of a program lifecycle can be seen in figure 5.1a. The service phase of a program is written with a `main loop` - an infinite loop in the program that blocks until it receives input and then executes on that input. In this diagram, the blue arrows represent control flow and the red arrow represent data flow. The program has an entry point (green oval), some setup processes (yellow rectangle), then the service phase's main loop. There may be a "tear down" phase once the main loop exits, but this is similar enough in structure to the setup phase that it does not require its own diagram. Unless a drawing is labeled otherwise, we try to always use the same shapes for the same operations. More information about what each shape means is available in Appendix C.

Even systems or frameworks that claim to be event-driven still have a main loop which calls the functions defined as "event handlers" in the program code (event-driven systems can also be implemented using hardware interrupts - where hardware can signal the processor to interrupt the regular program execution and do something else, but these are not used in commodity computers for application programming). Modern operating systems support thread sleeping, allowing a process to not use significant resources while blocking for input, but the structure of the program is still written with a main loop.[2]

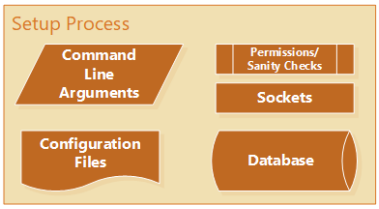
[2]Even programs that present the user with a Graphical User Interface technically operate with a main loop and a client-server model. A program instructs the graphics card to draw a shape, but then waits to be signaled that data or additional instructions are ready to be processed. This architecture is even more

When the server starts up, it runs setup operations before it starts the main loop and then is ready to serve requests. The operations that run as part of the setup phase often include reading command line arguments, loading data from configuration files or a database, checking permissions, and opening sockets, as in figure 5.1b. During the tear down phase, the program may close files handles cleanly, finish database transactions, delete temporary files, or similar tasks. Once the main loop starts, the program operation can still perform many of the same operations, including read from (and write to) files or a database and access external resources, as seen in figure 5.1c.

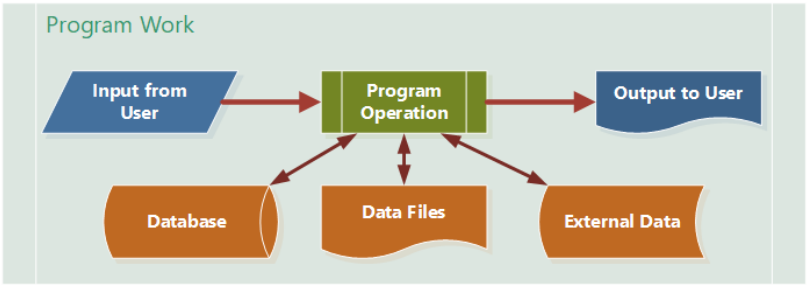
Figure 5.1 Example operations of a simple client-server program



(a) Simple Setup and Main Loop



(b) Setup Process Detail



(c) Main Loop Detail

We will use a sample webserver program called `maitre_d` as a simplified example client-server application. The `maitre_d` server can operate as a forked process or a threaded application. It can operate as a very fast static resource server (content is cached

clear in Linux as the GUI provider is referred to as the `display server` (e.g. `X`, `Wayland`) - the display server runs in a loop and any program that wants to write to the display connects to it as a client.

in memory at start-up) or in an application server mode where it is able to run dynamic programs. A code listing for `maitre_d` with cross-references to its Configuration Flow Graph is in Appendix D.

5.2 Step 1: Identifying Configuration Knobs

In the example above, both the setup phase and the service phase of the program execution include file operations, user input (whether from the command line or `standard in`), database access, and potentially other types of operations which appear similar in their behavior. Because the types of operations in the setup process and main loop overlap, we can't make a clear distinction between the types of operations done as part of the setup phase and service phase of the program and use static analysis to find every time a file is read as a way of flagging what parts of the program operation are configuration, but we can potentially start with looking at any operations that occur before the main loop is started.

There are usually a limited number of ways configuration can be entered:

- Configuration file(s) or registry - On start-up, the program reads a file from a hard-coded location or a location specified as a command line argument.
  - A configuration registry, such as the Microsoft Windows Registry, is effectively a special-purpose file system: it has a tree structure (just like most standard file systems) containing keys (filenames) with different types of values (e.g. integer, hexadecimal, string).[3]

[3]In some languages and frameworks, such as in PowerShell, the Windows registry can be accessed (with

- Command line arguments - On start-up, the program processes input supplied on the command line (a special case of `standard in`), possibly consisting of:
  - flags - boolean on or off values, possibly with abbreviations (e.g. `--help`, `-h`, `/?`)
  - options - key/value pairs (e.g. `--port=80`, `-p 80`, `device eth0`)
  - bareword arguments - often a single main action a program should execute (e.g. `config-test`) or a list of other options in order by position
- Additional Locations - Once the initial configuration is read, it may specify other configuration sources. For example, configuration may be loaded from a database or a high-availability cluster, but accessing these configuration stores requires values from the initial configuration (e.g. database password, cluster discovery mechanism)

5.2.1 Read the (Friendly) Manual?

The traditional way of understanding program configuration is to read the manual. Before World Wide Web access became prevalent (or fast), the manual for a program would have been printed in large books. It was not uncommon to find wall-to-wall bookshelves in the office of a system administrator, filled with bookmarks or dog-eared pages. These days, most manuals are available electronically, either as electronic documentation distributed

---

the appropriate permissions) with standard filesystem paths, using `HKLM:\` or `HKCU:\` as a prefix instead of a drive letter (e.g. `C:\`). This is also similar to the Linux kernel virtual filesystems that provide `/dev`, `/proc`, and `/sys`.

with the program, or on the internet. Even though the manuals are more easily accessible, there is no standardized way to organize a software manual - manuals may be organized by topic, by expected usage, by order of addition of new features, alphabetically, or may not have any obvious organization method. Many electronic manuals have a table of contents, index, and/or search function, but not all do, and not all that do have one that is complete or fully functional. Some programs have a getting started document which may say something like “The primary server is fairly resource intensive, and must be installed on a robust, dedicated server.” [115], but no specific notes about performance, or might have a “Performance Tuning” document [116] buried in alphabetical order with the rest of the settings.

Many programs also come with sample configuration files, whether created by the original developer or by a packager. There may be a single file with every possible option, or there may be multiple files, each based on a particular scenario for using the program. Some programs can generate a list of every possible configuration option by themselves (e.g. MySQL has a command to show all server variables and almost all of those can be configured at runtime), but a configuration file with all of those options would be overwhelming, and would not be useful because most deployments of the software do not need to change every single setting. Sample configuration files and manual contents are often outdated, which may make them difficult to use or just plain broken, but they can still provide insight into a starting point to understand the available configuration options. Unfortunately, many software packages only provide simplistic configuration samples, leaving administrators to try to piece together more-complex applications from separate - and often partially conflicting - examples.

5.2.2 Structured Representations of a Manual

A complete manual for a complex piece of software is necessarily complex, and is therefore difficult for a system administrator to completely read and retain. A structured, computer-readable, version of the manual can help an administrator find and understand what parts of the manual are the most important or relevant.

A software manual may be written as a separate documentation project, it may be generated automatically from comments or attributes in the source code[4], or some combination of the two. Modern manuals are often authored using a standardized data format, commonly Markdown or XML, and then built using a static website generator or groff (`man` page) converter into a final rich text (human-readable) format. Because the configuration information is available in a machine-readable intermediate format, it is possible to use that data for analysis purposes which can help us understand the program's knobs.

5.2.2.1 Reverse Search

Given a structured representation of an existing manual, we can extract a list of all the configuration knob names and search for them in the program's code. Depending on the structure of the program's code, this may produce obvious useful results or manual additional review may be required.

For example, a simple server may have the configuration options listed in the documentation listed in listing 5.1.

[4]Documentation can also be generated from static analysis of the source code (e.g. [69]), but this is intended for use for API documentation for programmers to use, not for documentation for system administrators.

Listing 5.1 Simplified excerpt of configuration based on httpd documentation

```
<directivesynopsis>
    <name>ServerName</name>
    <description>Hostname the server uses to identify itself</description>
</directivesynopsis>
<directivesynopsis>
    <name>Listen</name>
    <description>IP address and port to bind to</description>
    <syntax>Listen [<var>IP-address</var>:]<var>portnumber</var></syntax>
</directivesynopsis>
<directivesynopsis>
    <name>Security</name>
    <description>Should the server use SSL or plain text</description>
    <syntax>Security On|Off</syntax>
    <default>Security Off</default>
</directivesynopsis>
```

If we search in a theoretical example program source code for the configuration directives in listing 5.1, we find these results:

```
1. $socket_to_open = $CONFIG["Listen"];
2. if ($CONFIG["Security"]) {
3. print("This is :servername", [':servername' => $CONFIG["ServerName"],]);
```

This tells us that the `Listen` knob sets the contents of a variable, and when we investigate further, that value is used in a system call to open a socket. The `Security` knob is a condition on a gate, which may have significant effects on the program operation. The `ServerName` knob is only ever used when printing output.

The other thing we learn from here is that configuration is accessed through the `$CONFIG` array variable. The `$CONFIG` array variable itself is set this way:

```
$CONFIG = parse_config_file($ARGV["config"] || "/etc/myserver/server.conf")
    || throw new Exception("Invalid Configuration");
```

This tells us that the configuration is read from a filename specified on the command

line, and if that is not specified, from a file at a default location. (If the configuration is not able to be loaded, the program will exit with an exception.) We will use this information to evaluate the significance of the effects of changing each knob.

Aside: Code parsing is actually hard The Apache HTTPD documentation is a good example of why parsing the manual to find configuration knobs is not necessarily simple: The syntax for the `Listen` directive on line 8 of listing 5.1 is a combination of 1. the structure of the manual (`<syntax>` means this will be rendered as an example of how to use the directive), 2. the options for the directive (`[]` means that the content inside the brackets is optional), and 3. display values (`<var>` is rendered in a particular visual style). Additionally, this line does not include any information about the format expected of the value for `IP-address` (which will be even more confusing if dealing with IPv6 IP addresses because they may be wrapped in square brackets and they contain colons in the address, not only separating the address and port). Additionally, the Apache HTTPD server itself is very complicated because it is built for many different platforms and it uses the Apache Portable Runtime, so it uses a lot of C macros to make sure it will build the correct executable for each platform; and because it has many years of backward compatibility requirements and technical debt. The actual way these directives are processed is with hooks and several layers of indirection, as can be seen in code listing 5.2.

The `LISTEN_COMMANDS` macro declares that the `Listen` directive will be processed by a function called `ap_set_listener`. `ap_set_listener` itself is defined with a macro that determines how to actually define the function based on whether there are statically- or dynamically-linked libraries. The values specified in the (one or more) `Listen` directives

in the configuration are parsed by a function from the Apache Portable Runtime - not the HTTPD server - and are written back to variables in memory by passed pointers. This is among the simplest examples of macros in the HTTPD code - for example, some macros change the names of the functions being declared, making tracing executions even harder.

Listing 5.2 Excerpt from httpd configuration processing for `Listen` directive

```
#define LISTEN_COMMANDS \
AP_INIT_TAKE_ARGV("Listen", ap_set_listener, NULL, RSRC_CONF, \
"A port number or a numeric IP address and a port number, and an optional protocol")

AP_DECLARE_NONSTD(const char *) ap_set_listener(cmd_parms *cmd, void *dummy,
                                                int argc, char *const argv[])
{
    rv = apr_parse_addr_port(&host, &scope_id, &port, argv[0], cmd->pool);
    if (rv != APR_SUCCESS) {
        return "Invalid address or port";
    }
    return alloc_listener(cmd->server->process, host, port, proto, NULL);
}
```

5.2.2.2 Forward Search

Many software development projects use “hooks” in the version control system that will run a linter on a commit to check for new methods in the code and make sure they are internally documented. This is not universal and may be harder in some programming languages and therefore more or less common in different areas. This type of automated checking is almost never done on configuration options; there is no common system or practice for automated flagging of undocumented or changed knobs. (In some programming languages and with a program written following good programming practices, this may be much easier [86], but these methods still require significant manual intervention.)

As a side effect of being able to complete a reverse search, matching known configuration knobs to their associated code, we can use the recovered pattern to find configuration entries which are not present in the manual. We can search our theoretical example code from the previous section for any use of the `$CONFIG[*]` array with a key that we do not already know about.

```
1. print("Admin Contact :serveradmin", [':serveradmin' =>
       ($CONFIG["ServerAdmin"] || "Unknown"),]);
2. for ($i = 0; $i < ($CONFIG['BotherUserMaxTimes'] || 1), $i++) {
```

We have discovered two undocumented knobs. The first one, `ServerAdmin` is just a display value, similar to the `ServerName` directive that we previously saw in the documentation. The second knob is a gate condition on a loop - something that has the potential to cause significant performance impacts and that we didn't even know existed.

This type of searching requires a manual review to make sure the knobs that were found are properly extracted from the code. An additional manual review is necessary to understand the impacts of the knobs.

5.2.3 Static Analysis

Now that we know what patterns or behaviors we expect configuration knobs to look like in a program, we can try to find knobs with static analysis. We know that configuration values come from direct program input (stdin or command line), files on the filesystem, and possibly other areas such as databases, which are also places where program data can come from. We can distinguish loading configuration from request data based on whether they are accessed before the main loop of the program starts or inside the main

loop. Static analysis tools can create a list of all the places in the program where outside data is loaded and what variables are loaded with that data.[5]

Whichever way we find the configuration knobs, we now need to know whether they actually have a significant impact on the execution of the program.

5.3 Step 2: Identifying Critical Knobs and Interactions

How do we know which of the knobs are important? In our simplified examples, we take for granted that we can manually identify what the function is of each of these lines of code, but that quickly becomes unmanageable when dealing with a program of nontrivial size. Instead we can create a graph of the program operation which will help us find the critical areas.

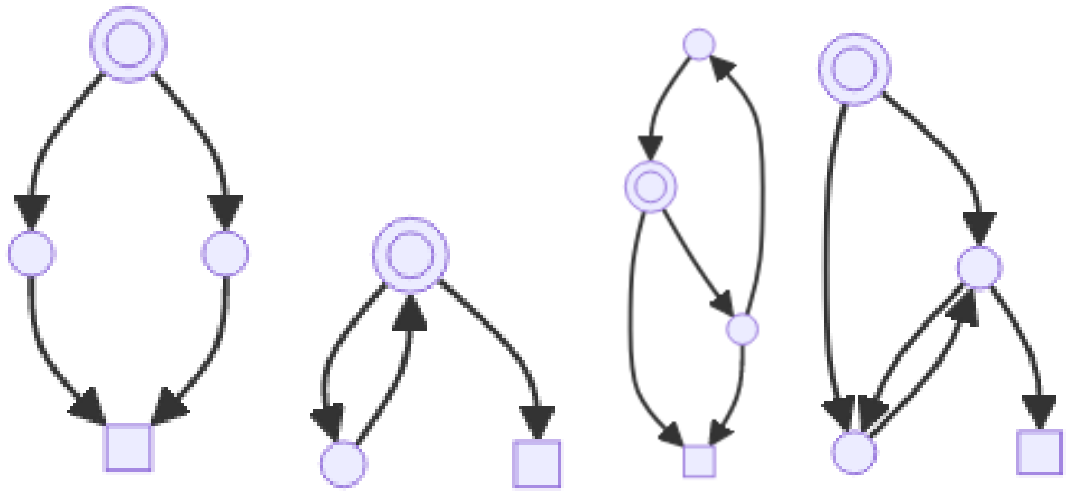
5.3.1 Control Flow Graphs

A Control Flow Graph [7] is a minimal directed graph representing the potential execution paths of a program. The original purpose for Control Flow Graphs was to allow reasoning about the correctness of compiler optimizations, but they are useful for other kinds of reasoning about program execution. Each node in the graph represents a section of the code that is always executed sequentially with no possibility of branching or jumping.

[5]Some programming languages like Perl keep track of that kind of information during runtime for regular user-supplied data, marking data as “dirty” and not allowing it to be used in sensitive operations until it has been validated/sanitized which will automatically mark it as clean. While this is similar to our static analysis, I am not aware of any programming language or framework that does this for configuration values.

Figure 5.2 shows several examples of graphs that represent execution of a program.[6] (Technically, figure 5.2c, although it correctly shows program execution, does not show a true Configuration Flow Graph because it can be further reduced by removing the node that only has a single entering edge and a single exiting edge.)

Figure 5.2 Classical examples of Configuration Flow Graphs



(a) if/then/else **(b)** while **(c)** loop with two exits **(d)** loop with two entrances

We aren't interested in every possible change in the control flow, but we can use a Control Flow Graph as a starting point to understand how knobs affect program execution.

5.3.2 Graph Labeling and Minimization

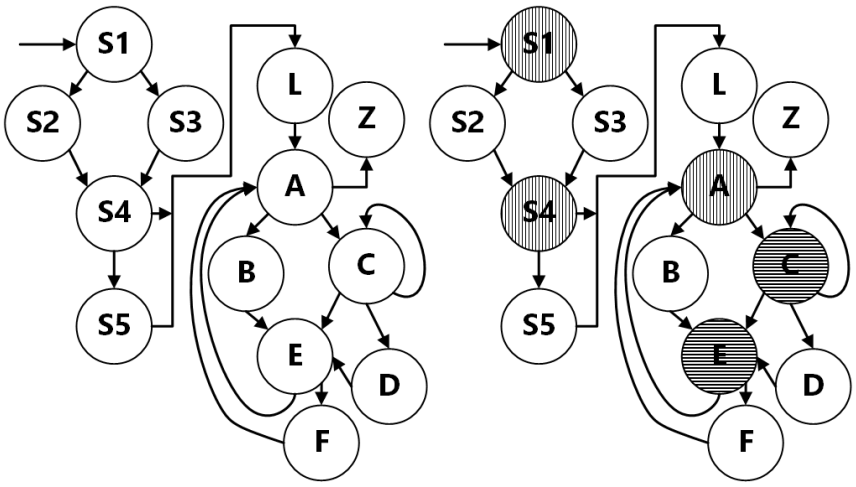
Branching decisions in a program can be made based on the values of configuration knobs or based on data specific to the request sent from the client to the server.[7] A standard Control Flow Graph shows all branching points in the execution of a program, but no

[6] See notes in Appendix C for information about graph entry and exit points.

[7] Even environmental data (e.g. current time, timezone, locale, network availability, uptime) which could be used as a branching condition is either associated with the current request, in which case we consider

points which are not branching points (i.e. there are no nodes in the graph with only a single exiting edge). Figure 5.3a shows the Control Flow Graph for `maitre_d`, with a configuration phase `S1-S5`,[8] service phase main loop `A-E`, and cleanup phase `Z`. Where there are multiple options for exit paths from a node, we can label the node based on whether the choice of path is controlled by a configuration option or by request-specific data. Because we are just concerned about configuration, we can minimize the graph to only include nodes that are controlled by configuration, but not those controlled by other data. Figure 5.3b shows a graph labelled by whether the direction of the program execution is controlled by configuration or other data. Nodes where the exit path is determined by a configuration value are shaded with vertical lines. Nodes where the exit path is determined by data flow are shaded with horizontal lines.

Figure 5.3 `maitre_d` Control Flow Graph



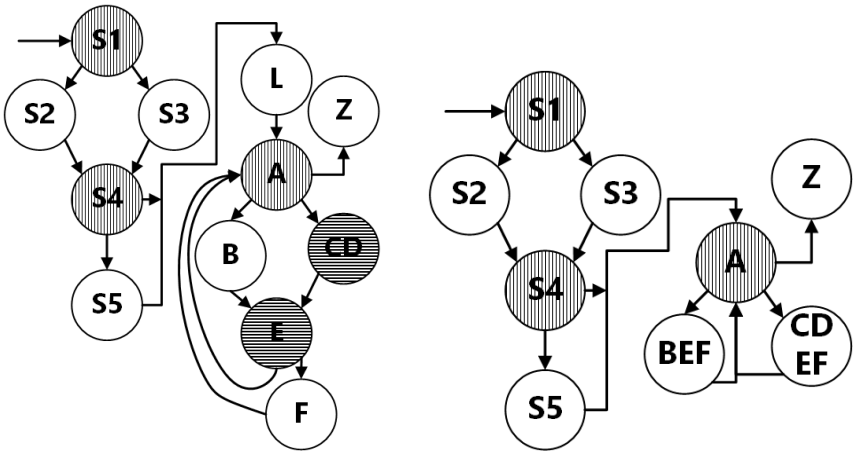
(a) Control Flow Graph **(b)** Control Flow Graph labeled

it to be part of the request-specific data even though it was not explicitly supplied by the client, or is fixed when the program starts and is the same for every client requests and therefore can potentially be considered part of the configuration data.

[8]See notes in Appendix D for an explanation of the extra node `L`.

We can use this information to reduce the Control Flow Graph to a simplified graph showing only the nodes which are affected by configuration changes. We traverse the graph to find any subgraphs that do not contain branching decisions that use configuration values and we collapse those subgraphs into a single node, as shown in figure 5.4a. In our graph for `maitre_d`, we could collapse nodes `C` and `D` into a combined node `CD`, but that still leaves us with a branching decision at node `E` that is not determined by configuration. We can also duplicate a node `E` and that will allow us to create new nodes `BEF` and `CDEF`, each representing all the possible nodes for the two branches chosen in node `A` based on the configuration.[9] Our simplified graph is shown in figure 5.4b. We call this a Configuration Impact Graph.

Figure 5.4 `maitre_d` Graph reduction steps



(a) Control Flow Graph reduced **(b)** Configuration Impact Graph

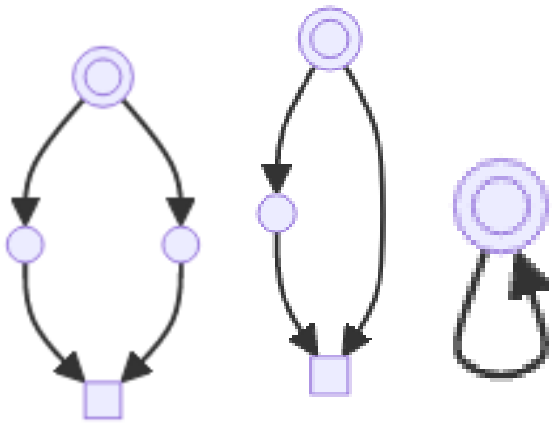
The Configuration Impact Graph gets us half way to really understanding the impact of configuration on our program. What are the actual *impacts* of any particular knob?

[9]It is not obvious that there is an algorithm to complete this step efficiently, especially because this graph is not guaranteed to be acyclic, and therefore not necessarily solvable with currently available technology.

5.3.3 Configuration Impacts

To make the Configuration Impact Graph useful, we also need to label each node with its expected performance or security implications. We can start with some simple observations about the structure of the Control Flow Graph. Figure 5.5a shows an `if` statement with an `else` option, which has the same number of nodes on each execution path and therefore appears to have the same performance no matter which path is followed. Figure 5.5b shows an `if` statement without an `else` option, which has a longer execution path if the guard statement evaluates to true. Figure 5.5c shows a loop which will be executed an unknown number of times, which therefore has an undetermined, but possibly very significant effect on performance. Any of these nodes could have some impact on the security of the program.

Figure 5.5 Examples of costs apparent in Configuration Flow Graphs



(a) if/then/else **(b)** if/then **(c)** loop

How we label the impacts of each node depends on whether we are interested in security or performance.

5.4 Step 3: Performance and Security of Each Knob

The considerations for properly setting a knob may be significantly different depending on whether you are interested in performance or security. This varies because of the actual effects of any particular knob and the mechanisms by which the knob changes the behaviors of the program. Similar to the considerations that come up when labeling the nodes of the configuration impact graph, we can't create a single method of analyzing the impact of a knob on performance and on security. While security and performance are almost certainly not independent of each other, these considerations are still so different, and indeed sometimes in direct opposition to each other, that each deserves its own attention (and its own chapter). We will discuss analyzing the effects of knobs on security in chapter 6 and performance in chapter 7.

5.4.1 Other Measures of Importance

In our example programs, the potential magnitude of the impacts of each knob are readily apparent, but as a program gets larger, other strategies are needed to know what parts of the graph to look at.

One approach is to list all the nodes of the graph in order of total cost to performance or by some measure of security severity. One downside to this strategy is the potential for the top of the list be made of knobs which are not used very often, making the list itself less useful. For example, knobs that control debugging/tracing functions are very likely to have a significant performance impact, but are not likely to be used very often.

Another approach is to look at the centrality of each node of the Configuration Impact

Graph. Network centrality measures are commonly applied to graphs of users in social networks to determine how central any individual person is to the entire network. Elsaka et.al. showed in [32] that network centrality can be applied to networks created from Graphical User Interface interaction options and that network centrality is a reasonable measure of how important each part of the GUI is for the operation of the program. Network centrality measures are an established method of understanding the importance of nodes in a graph and may be used to locate the most important entries in a configuration impact graph. A knob with is more central will have an impact on more of the operation of the program.

5.4.2 Further Understanding Knob Dependencies[10]

Knobs may have a hierarchical or other dependent relationship with each other. Sometimes this relationship is intrinsic to the design of the program itself, in which case we may be able to see this relationship in the Configuration Impact Graph as one branching node which controls access to another branching node. An example of this would be where one knob controls whether SSL is enabled for a program and the value of a different knob only changes the behavior of the program based on that. Listing D.1 shows an example configuration file (based on Apache HTTPD's syntax) for an application that only considers user-based authentication if the credentials will be passed over a secure connection, so disabling the connection security will result in all content on the server being available for public access. To allow representing this as a graph, listing D.3 shows the

[10]Code for this section has been moved to appendix D to improve readability.

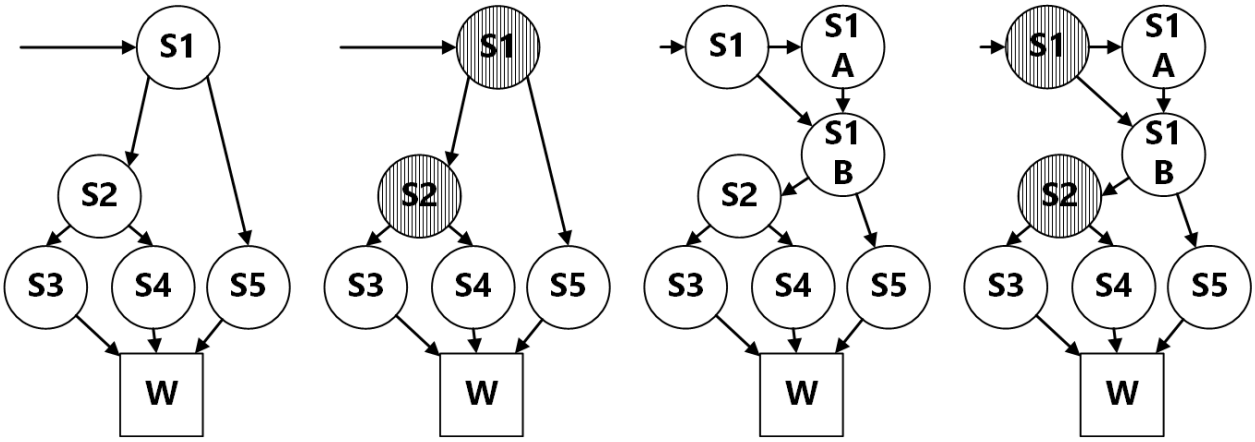
behavior implemented in pseudocode. The corresponding Control Flow Graph is shown in figure 5.6a and the labeled Configuration Impact Graph in figure 5.6b. In these graphs, node `S1` branches based on whether `SSL` is enabled. Node `S2` branches based on whether the authenticated user is allowed to access the resource. It is clear that `S2` will only be reached based on the outcome of the guarding condition in `S1`.

If this knob relationship was represented slightly differently (which we will refer to as *"messy"* for short), it would be much more difficult to detect in our Configuration Impact Graph. Listing D.2 shows an similar example configuration file, except that the security configuration has been separated logically from the authentication configuration, although the final outcome is the same.[11] Listing D.4 shows the behavior implemented in pseudocode. The corresponding Control Flow Graph is shown in figure 5.6c and the labeled Configuration Impact Graph in figure 5.6d. Node `S1` branches based on whether `SSL` is enabled, and the SSL module is loaded in node `S1A`. Node `S1B` branches based on whether the SSL module is loaded in the server. It is not clear that `S2` will only be reached based on the outcome of the guarding condition in `S1`.

Sometimes this relationship can't be evident in the graph because it is external to the program. An example of this would be a knob that controls how many threads or forked process are created which may end up in contention for processor time or shared

[11] We assume for this example that there are no other conditions that would cause the SSL module to not be loaded and still allow the program to continue (e.g. if the module is missing, the program will fail to start rather than proceed without it), making these examples functionally the same. There may be other methods (e.g. compiler optimization) that might be able to simplify this graph, but there is no way to be sure for this generic pseudocode example.

Figure 5.6 Example graphs showing knob dependence



(a) Configuration Flow Graph for Dependent Knob Example

(b) Configuration Impact Graph for Dependent Knob Example

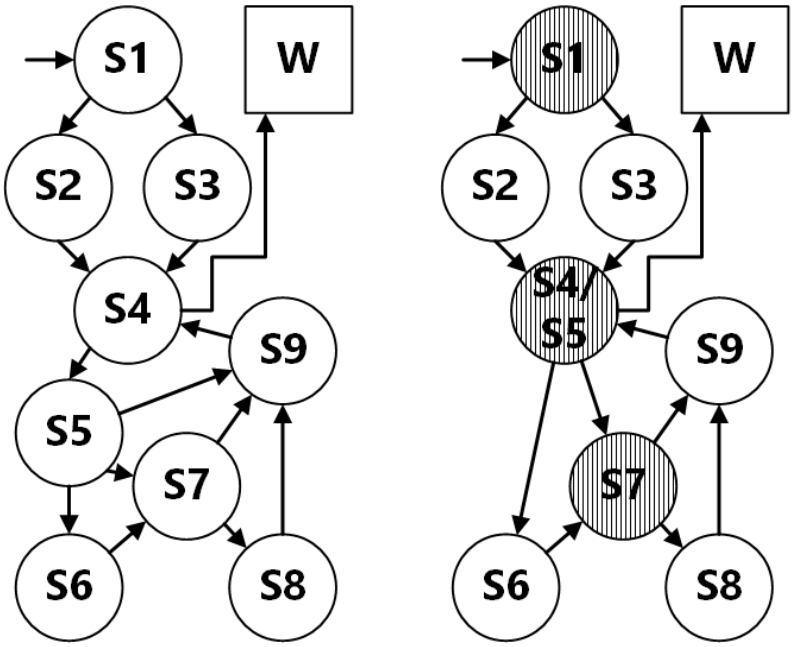
(c) Configuration Flow Graph for Messy Knob Example

(d) Configuration Impact Graph for Messy Knob Example

resources. Listing D.5 shows a (deliberately poorly written) program which will perform significantly differently depending on whether the configuration is set to have the program create workers by forking child processes or by creating sibling threads. This is a contrived example because in all other examples in this work, we assume that proper programming standards were followed, not that memory is shared unintentionally or incorrectly due to poor programming (e.g. the programmer might have intended to share the memory between threads and did not understand that this particular example is not thread safe). The corresponding Control Flow Graph is shown in figure 5.7a and the labeled Configuration Impact Graph in figure 5.7b. In these graphs, node `S1` branches based on whether the program is running with forked workers or threaded workers. This will impact other parts of the code which are not thread-safe and this impact will not be shown in the configuration impact graph. Node `S7` is supposed to create a sampling log file - recording operations only when the counter is a modulo of another number given

in the configuration, but this will likely not produce the expected results under threaded execution because the counter can be set by a different thread.

Figure 5.7 Example graphs showing knobs can't always be shown to be dependent



(a) Configuration Flow Graph for Bad Code Example

(b) Configuration Impact Graph for Bad Code Example

5.5 Step 4: Communicating Knob Importance

The goal of analytics of configuration management is to allow a system administrator to plan a system to allow each piece of executing software to have all the resources it needs to execute safely and efficiently, whether it is executing as a standalone program or as a component of a larger system, and whether each operation of the program is independent of any others or whether they build on each other or conflict with each other. Even if we can determine the impacts of a configuration change in a practical way, publishing it in an abstract scientific paper that no one will ever read again won't help

improve the global field of configuration management. Instead, we also need a way to communicate the importance of each configuration knob, preferably in a hierarchical way so that overworked and overwhelmed system administrators can take note of the most critical knobs.

In our definition of configuration in chapter 2, we listed many different ways that configuration knobs could impact program and system operations. Once we have located the knobs and established their potential impacts, we can use that information to create a basic score sheet for each knob, similar to the way scores are calculated for security vulnerabilities (e.g. the Combined Vulnerability Scoring System). We discuss the scoring system and communication more in chapter 8.

5.6 Additional Step: Detecting Unsuitable Values

Understanding all the of the available knobs and their impacts is just a start; maybe a way to turn a competent system administrator (who understands the system, but needs some pointers to the hidden factors that can have significant impacts) into an excellent system administrator. To take the analytics of configuration management up to the next level, we should also be able to make specific recommendations for the proper usage of each knob. As we explained in our discussion of why software has so many knobs to begin with, because we are dealing with software that is designed to be used in many different ways, with different loads and with a variety of different hardware resources available, there is no simple way to know how every knob should be set (if there was a simple way to know the value a knob should always be set to, it probably wouldn't need to be a knob).

We suggest several approaches to evaluate the suitable or unsuitable values for a knob, but the actual implementation of each of these (and other) strategies is an entire project in its own right and is left for further study.

5.6.1 Developer Specification

One way to find unsuitable values for knobs would be for the software developer to literally add a checker into the program. This has all the same problems of any solution to knob impact analytics that relies on an action by a developer: keeping the program checks up to date with other changes to the program, and knowing what values are important in the first place. For many systems, this type of recommendation system will only work once the system is configured and is running in production because it will only show results under high organic (non-artificially generated) loads.

This is implemented in to varying extents in some software packages. Some network switches will warn the user on the console when debugging or traffic monitoring options are enabled because these can cause high CPU load and crash the switch. The Apache HTTPD server will log errors[12] that recommend settings changes to possibly improve performance:

```
AH00161: server reached MaxRequestWorkers setting, consider raising the
    MaxRequestWorkers setting

AH00162: server seems busy, (you may need to increase StartServers, or
    Min/MaxSpareServers), spawning %d children, there are %d idle, and
```

[12]Apache HTTPD developers insert a unique number for each place in the code where an error is logged. This is designed to make it easier to help people who post on mailing lists or forums about errors they encounter, leading the developers or other users to the exact reason the error has been encountered.

```
    %d total children

AH00326: Server ran out of threads to serve requests. Consider raising
    the ThreadsPerChild setting

AH01144: No protocol handler was valid for the URL %s. If you are using
    a DSO version of mod_proxy, make sure the proxy submodules are included
    in the configuration using LoadModule.
```

These recommendations can actually be misleading because the server hardware still has to be able to support the number of concurrent processes or threads; increasing the number of workers the application tries to create without having the appropriate system resources available will still result in poor performance, but won't log these particular messages.

Similarly, the operating system kernel can log performance-related messages although these are more likely to indicate a hardware or network issue than a configuration problem.

5.6.2 Automated Testing and Fuzzing

> *A software tester walks into a bar and orders a beer. Orders 2 beers. Orders 99999 beers. Orders 0 beers. Orders -1 beers. Orders qwerty beers. Orders a lizard in a beer glass. The bar opens for business and the first customer walks in and asks where to find the bathroom. The bar explodes.*
>
> – Joke of unknown origin about software testing

Effective software testing, whether automated or manual, is a major field in its own right, with its own dedicated practitioners, its own focus in scientific research, and its own

practical implementation problems [16]. Some large software projects have significant automated testing suites. Many open source projects will not accept new code submissions without associated tests to show that the changes work correctly and do not introduce undefined or unexpected behavior. When tests are available, we may be able to use them to evaluate the impacts of changes to different knobs. If a program has a comprehensive test suite, it gives an insight into the types of sets of configuration that the developer expects to be valid and can allow flagging configurations that are significantly different than the test suite.

Fuzzing is the process of feeding a lot of different (generally random) values to a program to see how its behavior changes [74]. This is useful for testing user input to a program that expects a very large variety of input values where it would be hard for a tester to write tests that would cover every possible input, and especially useful for finding bugs due to nonsensical values that a real user would never think of using. We can test applications to discover significant changes to the configuration impact graph with modified fuzzing that will only produce valid configuration files. (Most configurations produced by standard fuzzing techniques would likely not be valid, and this includes generation by current LLMs [98] - since we are interested in how the configuration impact graph changes, we need the program to be able to run with the generated configuration.) By creating many new configurations with fuzzing and looking at how the path through the configuration impact graph changes, we can find knobs that produce particularly significant changes and flag them for additional study.

5.6.3 Evaluation of Internet-Posted Configuration

Francis Galton famously published an article [38] in the journal Nature in 1907 declaring that the democratic "voice of the people" - the median guess of those guessing the weight of an ox at a fair - was more accurate than any of the individual guesses. Galton's conclusion that the wisdom of the crowd is better than any single guess is commonly quoted and accepted as a fact (with many scientific and popular works building on this assumption, e.g. [101]), but it has been challenged on the grounds of consistency and statistical methodology [105]. This conclusion has been tested (scientifically [10, 54], and in practical demonstrations [113]) and holds up under certain conditions, including: that the guessers have some basic understanding of the concepts, the knowledge is not too domain-specific, the guessers don't need to expend significantly different effort to answer the question, the guessers don't have confounding experiences, and the real result is not random.

Is there any value in the wisdom of the crowd when it comes to setting configuration knobs? There are many forums and mailing lists for technical assistance with configuring different programs; for example, questions about configuring the Apache HTTPD server can commonly be found on ServerFault (the server configuration version of StackOverflow) and on the HTTPD users' mailing list. Both of these locations have many examples of working and non-working (and partially working) configuration.

While it is commonly the questions which have non-working configuration and the answers that have working configuration, this is not always the case. Instead we can try to perform natural language processing and sentiment analysis on each post that contains a

configuration snippet to see whether it is likely a working configuration or a non-working configuration. For any knob that appears in a post associated with a functioning configuration, we record it with a positive score. For any knob that appears with a non-functioning configuration, we record it with a negative score. In addition to the presence of the knob, we can also record the values for each knob to be able to look for bounds of "good" values (assuming the knob takes a numeric value). This will create a list of knobs that are most common in working configurations compared to non-working configurations.

There are a number of challenges with this approach. Configuration snippets posted may be anonymized or munged in a way the hides values that would be important or make the entire snippet incomplete. Sentiment analysis for software engineering may require a significant amount of custom tuning if it is even possible at all [49, 63]. It is considered best practice when posting mailing list or forum answers to include context for configuration files, but people often post answers with just the specific value that needs to be changed, which could leave the knobs present in the question with an unexpectedly low score because they are not present in the answer.

Chapter 6: Knob Security

Sir, […] Turn Your Key!

– War Games (1983) - 1st Lieutenant Steve Phelps

In the previous chapter, we mentioned several times that security and performance need to be addressed separately from each other. The reasons may not be readily apparent, so we will take a short detour to explain. As an undergraduate student, I took a one credit winter course on computer forensics, taught by an investigator from the US Department of Justice Child Exploitation and Obscenity Section. At the beginning of the class, he went around the room and asked what people were interested in and why we were taking the course. This was in January 2011, as the `cybersecurity` buzzword was becoming very popular[1] and almost everyone said they were taking the course because they were interested in security. The instructor was very disappointed that we wanted to make his job harder; as he explained it, good security is the enemy of effective forensic operations because it makes it harder to perform comprehensive (especially adversarial[2]) forensic

[1]The word cyber was invented in the 1940s and extensive use of the prefix `cyber-` can be tracked to the 1980s, although the term `cybersecurity` did not enjoy huge popularity until much later [44]. A search on Google Books Ngram Viewer [65] for `cybersecurity` [125] shows a significant jump from 2008 to 2010, then plateauing for a year. Listening to the local Washington, DC radio or reading the newspapers in those years, every advertisement targeting government technology buyers could be counted on to include a phrase about “meeting your agency’s cybersecurity needs”.

[2]For example, a failed storage device using full-disk encryption will be difficult to recover even though

analysis. Configuration settings which are otherwise-good for security can be the enemy of many other things: usability, availability/reliability, and performance.

```
Your password has expired. Passwords must be changed every 90
days. Your new password must by at least 8 characters long and
contain uppercase letters, lowercase letters, numbers, and symbols
from this list: !#$%^*){}[]\|
```

This is a familiar sight to internet users, but does the list of allowed symbols include your favorite one? Does it include all the symbols that were generated by the random password generator in your password manager? This is a classic example of configuration determined by (incorrect[3], outdated[4], or poorly programmed[5]) security considerations,

---

the user may be able to provide the encryption key. A criminal protecting information with full-disk encryption is a lot less likely to hand over the encryption key to the investigators.

[3]Many studies of password complexity don't have a "ground truth" [34] and are likely to have other validity concerns. Outside of that, many password requirements are made up or implemented based on what a developer or manager happened to read recently, without any scientific studies or engineering evaluations [61]. Both of these would lead a developer to set password requirements based on security considerations that are "incorrect".

[4]The National Institutes of Standards and Technology publishes a standard for secure authentication systems [43]. Even many governmental organizations don't follow these standards (due to auditors' own ideas of what is considered secure).

[5]Some websites restrict the use of special characters in passwords because some characters have special meanings in certain database systems. This can also indicate that the website does not properly implement user input sanitization or parameterized queries, meaning the website may also be vulnerable to other vulnerabilities such as SQL injection. Website policies or poor or unnecessary javascript programming can

any of which significantly impacts usability[6].

In the quote that begins this chapter, the controls of a nuclear missile silo are configured to require two people to simultaneously turn control keys to launch a missile. This configuration was created to prevent a single person from unilaterally performing a deadly action, but the functionality of the system breaks down because one of the operators can't bring himself to actually turn the key. You could call this a security-induced lack-of-availability or Denial-of-Service situation. The solution in the movie of computerizing the missile launch process - and not configuring the system to require strong passwords and multi-factor authentication - leads to (an entertaining movie and) actual technology regulation in the form of the 1984 Computer Fraud and Abuse Act [94]. This is a good example of how bad configuration can cause significant security problems.

Other examples of security-related configuration impacting availability or reliability include settings that control program memory access leading to a kernel panic (e.g. the well-known Blue Screen of Death), or a firewall with a security "trip wire" that blocks all access to a system to allow for investigation into a potential security breach. Any additional processing work that has to be done by a CPU will result in worse overall performance, sometimes very noticeably worse.

Security means different things in different contexts. Each application and use case will have its own threat model which guides the security-related decision process. Devel-

---

conflict with password managers that automatically generate a secure password; standards to address this have not been widely adopted (and probably wouldn't be implemented properly anyway) [40]. Password strength meters can also be unreliable [42].

[6]On the subject of passwords, the reader is strongly encouraged to try The Password Game [5, 85].

oping a threat model is outside the scope of this discussion, but here are a few examples of security concerns that an administrator might be able to address by changing a configuration knob:

- If a user requests a directory that does not have an index page, should the server show a directory listing (which may have useful information or may expose sensitive information) or show an error page.
- Will the server accept a cipher for secure-channel communication that is known to be possible to compromise (e.g. a block cipher in ECB mode[7] or a stream cipher using RC4[8]).

We are also only looking at security from the perspective of the program running on the server, or things that will affect all users of the service. User interacting with the program may expose their own personal security considerations (e.g. leaking the domain name of the website they are connecting to in the SNI headers to their office or govern-

[7]ECB ciphers are not practical for large amounts of data, mainly because 1. they reuse the same key multiple times, so an adversary might be able to see general information about a message or conduct statistical analysis on the ciphertext (the most common illustration of this is a picture encrypted with ECB will likely still be recognizable), and 2. because they don't provide any security against an adversary replaying or modifying the ciphertext. (This isn't a paper about cryptography implementations, so we leave further research on block cipher encryption modes to the reader.)

[8]Rivest Cipher 4 was very commonly used starting in 2011 because it was not vulnerable to the BEAST attack against other popular ciphers, but it has multiple known weaknesses and is not recommended for use anymore under any circumstances. Also refer back to our discussion of cipher configuration challenges in section 4.1.1.

ment monitoring system), but these may not be the concern of the system administrator.[9] (When system administrators do implement settings that protect individual users, it is often to get higher scores on automated configuration testing tools[10], rather than because they understand why they are making these changes.)

6.1 Step 3: Security of Each Knob

There is a wide variety of impacts a knob can have on security of a program. Some knobs explicitly control security related features, for example, enabling or changing parameters for secure-channel communication. Some knobs have security implications even though they are not explicitly changing security-related properties of the program execution, for

[9]System administrators may be able to implement extra systems or services to assist with these types of user concerns, but that is often outside the scope of the configuration of the server program. This issue gets at the heart of the limitations of currently-available cryptography methods: in short, how do you get the encryption key for a server without talking to that server first in plain text? If you are worried about a specific adversary watching your network traffic, you could use ToR [29]. A research group in our Computer Science Department was looking into internet censorship [18] and assumed that their target users would be using ToR. They approached the helpdesk because users on ToR were not able to connect to the website. UMD's edge firewalls block incoming ToR traffic as a proxy for stopping attackers from reaching the network [96] (if the only tool you have is a firewall, everything looks like a network problem?). We reserved two specific IP addresses that the campus agreed to open to ToR traffic. Another protocol that is supposed to address this concern is `Encrypted Client Hello` [17, 90] which is becoming the preferred solution to resolve this issue for the average user.

[10]The most popular SSL testing tool is the Qualys Lab SSL Tester. Unless the administrator requests that the results of their test be hidden, the names and scores of websites tested are shown publicly. It is possible to observe administrators changing settings and retesting in their pursuit of an `A+` rating.

example, allowing a remote user to upload arbitrary content which could include untrusted programs or code.

For the first category, it is relatively easy to find which sections of the code interact with security functions or libraries. There is a well-known software developer maxim that you should never "roll your own" cryptographic functions; Instead, you should always use a well-known and trusted library, developed by and reviewed over and over again by people who are smarter than you.[11] This gives us an easy target to label in our Control Flow Graph before converting it to our configuration flow graph. Unlike performance impacts where we will primarily look for program execution changes caused by knob settings in the service phase (as we will discuss in the next chapter), security impacts can appear in the setup phase (e.g. setting up secure channels) and in the service phase (e.g. encrypting data being stored in a database).

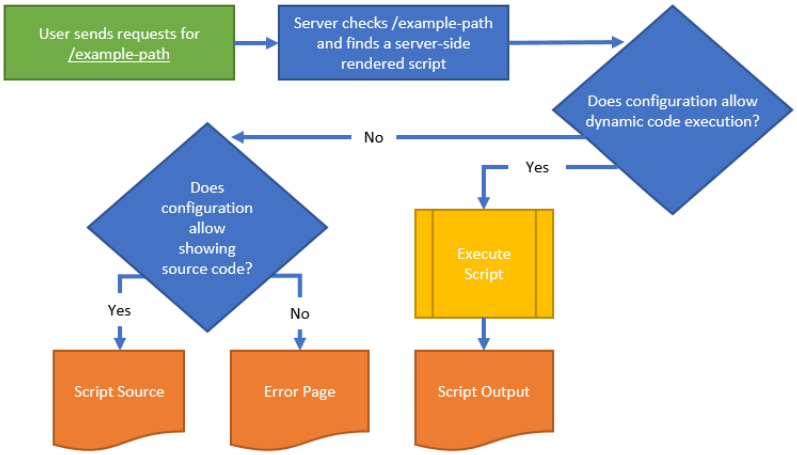
Performance is sometimes connected to security; for example, enabling a security feature may require more processing which will slow down the application (e.g. secure channel communication has encryption/decryption overhead), but that does not make the security concern into a performance concern.

Changing a setting of a knob can have unexpected security impacts due to the value of another knob. An administrator usually does not want their server to expose the code behind the application (of course unless the purpose of an application is to show people how to code). Source code or configuration exposed this way may contain sensitive information such as passwords, contain proprietary information or algorithms, or just be

[11] *Anyone can invent an encryption algorithm they themselves can't break; it's much harder to invent one that no one else can break.* – Bruce Schneier

considered the property of the developer (or whoever paid the developer) and the owner doesn't want to give it away for free. The request flow for the sample application shown in figure 6.1a has two branching conditions that are based on configuration knob values. The first knob controls whether the application executes scripts or considers them to be plain text. The second knob controls whether plaintext copies of the source code are served or whether the response will be an error. As long as the first knob controlling script execution is not changed, the application code will not be exposed, no matter how the second knob is set. If an administrator decides that the way a server is put into maintenance mode is by turning off script processing, this could have the unintended consequence of exposing the application code.

Figure 6.1 Example request flow of a webserver request/response



(a) Simplified Configuration Impact

6.1.1 Labeling the Configuration Impact Graph

Before we can label the nodes of a Control Flow Graph with their impact on security, we need a way to understand what parts of the program actually impact security. We can

split these into several categories:

- Knobs that control access to cryptography or other security libraries: These knobs will change the way the client is allowed to communicate with the server (e.g. turning security on or off completely, or choosing allowed ciphers).
- Knobs that control privileges: These knobs will change how the program asks the kernel to treat it (e.g. dropping root user privileges after startup).
- Knobs that control input or output: These knobs change where the program tries to read or write from in a filesystem or network (e.g. make sure user supplied data can't be written to a location where code can be executed to prevent malicious activity), or they control how data is stored across different invocations of/requests to the program (e.g. to make sure one user can't read another user's data).
- Knobs that control communication with other programs: These knobs will change what other programs the server tries to interact with (e.g. the knob controlling script execution in the source code exposure example we discussed above).
- Knobs that don't fit into any of these groups: The knobs are things we know impact security, but might be hard to find automatically because they are not clearly security related (e.g. the knob controlling error message display in the source code exposure example we discussed above).

`maitre_d` is a simple program, so it doesn't include many of these concerns, but we can still label a few things on the Control Flow Graph. Based on the original Control Flow Graph for `maitre_d`, in figure 6.2a, we have highlighted nodes with security implications. We labeled a node with a black dot if there is a security-sensitive operation that directly

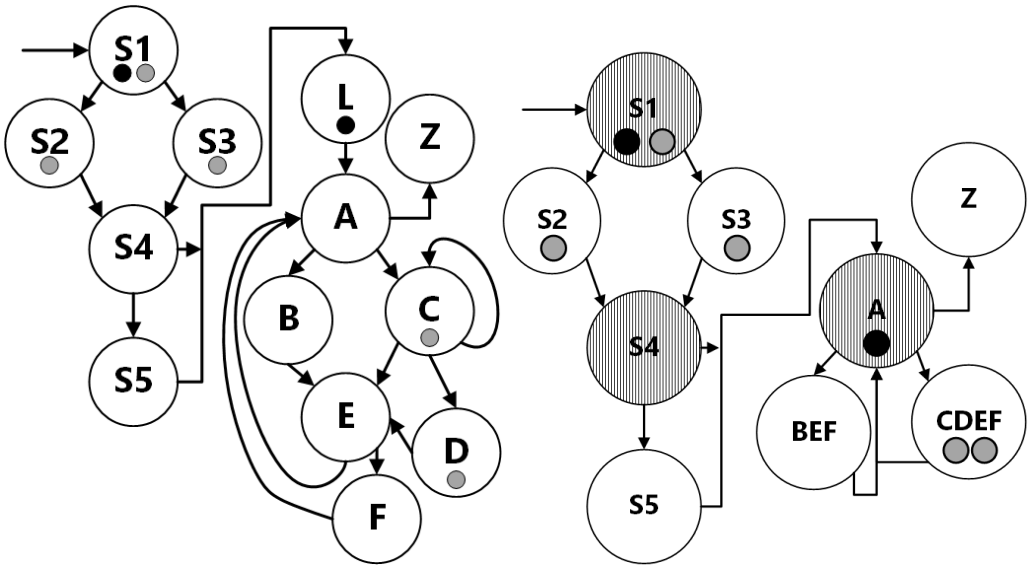
uses values from the configuration. We labeled a node with a grey dot if there is a security-sensitive operation that does not directly use values from the configuration. (The colors don't indicate the severity of the potential security impact, just whether the configuration is used directly as part of the sensitive operation.)

- Node `S1` has a call to open a socket using a value from the configuration, and a call to listen for interrupt signals which is not based on configuration.
- Nodes `S2` and `S3` open files or sockets, and are selected based on a value from the configuration but don't directly use those values themselves.
- Node `L` forks the process or creates threads (which we assume the kernel or PID 1 can handle in a single function call) - combining thread-safe and non thread-safe code usually leads to program crashes, but this could be security-sensitive in just the right (wrong?) circumstances, creating unintended memory sharing or race conditions. In our example program this is likely to be an issue only if the program is running `threaded` and in `dynamic` mode and using a linked library that is not thread-safe in node `D`.
- (Node `C` loops through directories in the filesystem and has no protections against symlinks to arbitrary filesystem locations. This threat is likely out of scope for the threat model relevant to this discussion, but is worth acknowledging. It is not an issue with many modern servers which either internally implement permission checks such as SELinux, or which as a rule refuse to follow symlinks.)
- Node `D` actually executes, either using a linked library or a socket to another program, code which may be changed at any point in the server's lifetime - if there

is a way for users to upload any content, they may be able to execute arbitrary instructions on the server. (There could also be issues with thread-safety as noted above.)

Looking at the labeled Configuration Impact Graph in figure 6.2b, we can see that there are only two nodes - `S1` and `A`, that branch the program operation based on a configuration knob value where that branch decision also changes whether a security-sensitive operation will take place. This graph doesn't show the relative security impacts of the two different operations in `S2` and `S3`, but does call our attention to the fact that there may be two different operations with potentially different impacts on security. Branching at node `S4` is controlled by configuration, but does not affect the paths through any security-sensitive operations. Node `A` is the only node that has a clear security impact; the first path has no security-sensitive operations, but the second path potentially has two.

Figure 6.2 Graphing `maitre_d` Security



(a) Control Flow Graph with security **(b)** Configuration Impact Graph with security

This information allows the system administrator to decide if the operations around

the security-sensitive path are truly needed for this instance of the program, or whether the entire execution of the program can be made safer by changing the value of the knob that controls whether the program executes that branch. In the case of our very simple sample application `maitre_d`, the branching condition in question switches between two completely different modes of the program operation, so the administrator is unlikely to want to change the mode, but this information still allows the administrator to be aware of the potential security impacts of the program.

Chapter 7: Knob Performance

Marty: …and most of these amps go up to ten….

Nigel: Exactly.

Marty: Does that mean it's…louder? Is it any louder?

Nigel: Well, it's one louder, isn't it? It's not ten.

Marty: Why don't you just make ten louder and make ten be the top number?

Nigel: ….. These go to eleven.

– This is Spinal Tap (1984)[1]

One of the first computers I ever used had a *turbo* button. As a kid, no one ever explained to me what the button actually did, but I knew that when I had a game open and I (or a brother or friend) pushed the turbo button, a display on the front of the chassis would change to show a different number and the characters in the game would suddenly move faster (and this would make *Word Rescue*[129] harder). Most early games were built without an independent timing system and would rely on the CPU clock speed to set the running speed of the game. The turbo button was created to solve the problem

[1]Spinal Tap was not the first to use the idea of a knob going up to eleven to indicate that something is better than its competition (usually without any basis) or is more powerful than the user might be used to or expect (e.g. the 1947 Chesapeake and Ohio class M-1 steam turbine locomotive had a throttle that went up to eleven [88]). I have never seen the movie, but the phrase "it goes up to eleven" is strongly associated with the cult-classic film. It was also not the first *mockumentary* although it did popularize the genre.

of new CPUs being too fast for some games. Counterintuitively for those used to the turbo ratings of modern CPUs which indicate that the CPU will run faster than the base clock speed under certain conditions (while using more power and generating more heat), the computer would operate at the normal CPU speed when turbo was *on* and would artificially slow down when turbo was *off* (and there was no downside to running with turbo on).[2][100] Thankfully almost all software no longer relies on the processor to slow down its operation in order for the software to be useable[3].

In the quote that begins this chapter, a band member is explaining that his guitar amplifier is louder than other amplifiers because it has more gradations on the knobs. It is supposed to be clear to the viewer that this is ridiculous, but the message relevant for this work is that it is hard to compare performance without some objective measures and without knowing what performance is expected. Computer users are generally familiar with the performance degradation of a computer over time. This may be due to increased resource demands from newer programs (or newer versions of programs), due to physical component degradation, or due to overhead that is inherent to systems that have been

---

[2]Also interestingly, the display values on the turbo screen were not actually showing the CPU clock speed. The speed shown was fixed using jumpers on the back of the display or on the motherboard and would just alternate between the preset values for turbo *on* and *off*.

[3]There are cryptographic/authentication functions which rely on an upper limit of processor speed. These functions are intended to be slow to make it too computationally expensive to use a brute-force attack against them, but this is not the case for most program operations which use their own timers if they need to ensure an operation runs at a particular speed. Computer users can still run old programs through an emulator (e.g. DOSBox) that will present the program with an artificial CPU clock to control the overall speed.

in use for a long time. It is not unexpected that a computer that has been in use for a year will be slower than the same model computer which has been reformatted and has a brand new system installation. It is also not unexpected that a multi-user computer will often be slower than a single-user computer[4], but the actual impact will depend on how many people use it and it is difficult to predict exactly how any particular computer and set of software will perform. This is even harder to figure out for a server designed for public access and will be there area we will focus on for the rest of this discussion of performance.

In modern computers, processor speed and other architecture limitations are viewed as a constraint on the high end of the performance range rather than a constraint on the low end. Developments in processor technology for many decades have been targeted at running larger programs faster than ever before, all without creating enough heat to damage the computer. Advances in miniaturization have allowed more components and more parallel data lines (wires) to be crammed onto a single circuit board, all with the goal of bringing more data closer to the CPU faster. To use these available hardware resources efficiently, a system administrator needs to understand the impact of configuration knobs on performance.

---

[4]For example, Windows has a feature called *Fast User Switching* which allows multiple users to be logged in and have programs running, each one locking the computer when they get up from the console which allows another user to log in. When I was in high school, our IT department would usually disable the Windows *Fast User Switching* option because users would never log out and that would cause the system to appear to run slowly.

7.1 Step 3: Performance of Each Knob

All [instructions' execution times] are equal, but some are more equal than others.

– George Orwell [81]

We introduced this chapter with a discussion of hardware - even though we previously noted that hardware configuration is out of the scope of this work - because almost any (in-person) conversation about computer performance invariably starts with a discussion of the hardware specs. There are many reasons computers operate on binary representations of information and instructions. Many of the binary operations performed by a modern processor would be recognizable to a computer technician from the early days of room-sized computers although they would be surprised by the number of different operations baked into the processor and by the sheer number of instructions a modern processor can handle in a very short period of time. For the purposes of this work, we are not considering hardware configuration differences even though they can change the execution time of each instruction because they are (usually[5]) not subject to the configuration knobs controlled by a system administrator. It is very unlikely that any knob in software doesn't have *some* impact on system performance. No matter what operations

[5]Many new processors have different types of cores for processing loads with higher performance using high-speed multi-threading cores or better efficiency lower-power lower-speed single-threaded cores, referred to as `p-cores` and `e-cores`. They are ultimately just two different spec'ed CPU cores connected to the same CPU socket. An administrator could manually set the affinity for a process to an `e-core` that should be processed by a `p-core`. This is a more likely to be an issue on consumer-grade hardware rather than server-grade CPUs which do not ship with mixed cores (as of August 2024). We will discuss this in our consideration of HAProxy in section 9.2.

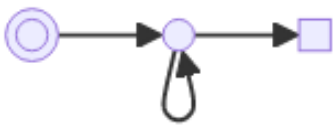
are being done, some number of bits must be moved between different registers in the CPU and transferred in and out of different memory locations, long-term storage locations, or other devices, and the value set for a knob will change those bits or the paths used to process them. For most types of software operations, a user would never notice the difference in the processing time of very lightly different bits, so we look at the impact of higher-level computer operations.

Not all operations are created equal when it comes to processing time, some intrinsically and some because they have to wait for other resources. For example, consider programs which take an integer from a user, double it, and return the value; one program works with the local user and the other with a remote user over the network. Listing 7.1 and listing 7.2 might look very similar and might both produce the same Control Flow Graph as seen in figure 7.1, but it is clear that the communication with a remote user will be slower and require more system resources than with a local user.

Listing 7.1 Get an integer value from a local user, double it, and send it back

```
define("IN", open_fd(stdin));
define("OUT", open_fd(stdout));
while (some_condition_which_is_usually_true()) {
    $value = get_integer_from_in(IN);
    print_value_to_out(OUT, $value * 2);
}
close_fd(IN, OUT);
```

Figure 7.1 CFG for both example programs



Listing 7.2 Get an integer value from a remote user, double it, and send it back

```
$socket = open_socket();
define("IN", open_fd($socket->in));
define("OUT", open_fd($socket->out));
while (some_condition_which_is_usually_true()) {
    $value = get_integer_from_in(IN);
    print_value_to_out(OUT, $value * 2);
}
close_fd(IN, OUT);
```

Here are some of the different operations that could execute as part of a program, along with some explanation of their potential performance-related issues:

- Locks/Mutexes/Semaphores[6]: This is usually a mechanism to ensure data integrity (whether in a single system or a cluster), or limit concurrent expensive operations. The locking operation itself may be cheap (for example, in some languages opening a file handle is used as a lock - because it can be enforced by the kernel, it is low cost), but its actual performance impact will depend on the number of concurrent requests that require the resource (and whether they wait for the lock or if they fail because they can't lock).
- Sockets/Network Access: Cross-process or cross-device communication is slower than operations internal to a single program. This difference may be measurable,

[6]Locks, Mutexes, and Semaphores are very similar. Definitions of the terms vary depending on the programming language or framework. Locks are usually contained in a single process and prevent more than one thread from entering the same part of the program at a time. Mutexes are usually shared by all processes on a single system and only allow one process to access them at a time. Semaphores are meant for communication between multiple processes and they can allow multiple processes to access them at the same time up to a certain limit.

but also may change based on load (e.g. overall network congestion).

- Cryptographic: Some cryptographic operations, mainly for hashing passwords and cryptocurrency, are designed to be slow to make it more expensive to conduct a brute force attack against them. [26]
- File access: A single file can usually only be written by a single process at a time to prevent corruption. Some programming languages or frameworks will allow multiple processes to read from a file at the same time and others will not. If the file is located on media with a measurable seek time, the program might need to wait for the requested file to become available (e.g. a modern spinning hard drive might have a seek time as short as 20 milliseconds down to single-digit milliseconds, while magnetic tape could have a seek time measured in minutes).
- Database Access: Just like any other client/server system, a database engine can only process a limited number of requests at a time. Inadequate database design can cause a program to run slowly. Database engines are usually responsible for their own data integrity constraints, potentially including locking at a database-, table-, row-, or cell-level, and any of those could cause performance slowdowns.

We looked at identifying different types of performance impacts in our discussion of program task definition in section 4.3.

7.1.1 Labeling the Configuration Impact Graph

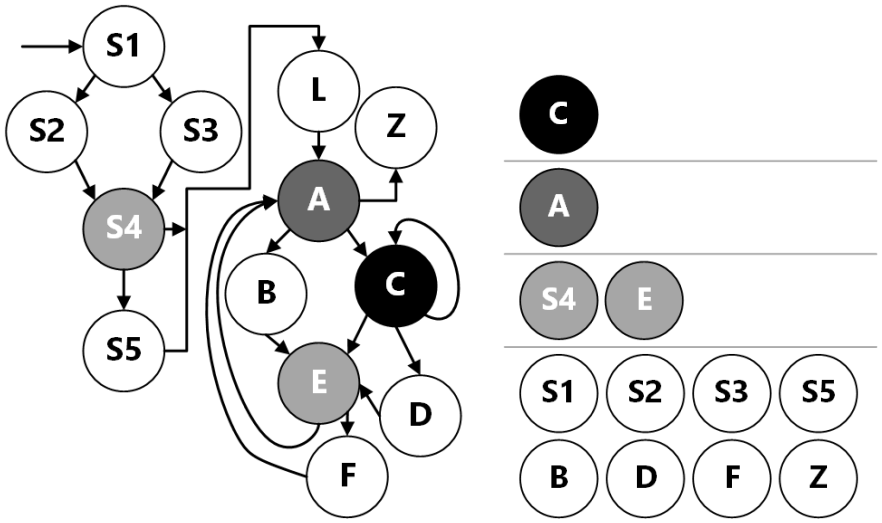
Performance implications might appear to be obvious from the structure of a Control Flow Graph, but that can be misleading. Because Configuration Flow Graphs only show

branching operations and do not include any information about the operations running inside a single node, we can't rely only on the cost-related features visible in the graph. Instead, we can label each node with its algorithmic complexity (i.e. big O notation), or less formally, with any other kind of structured or unstructured labelling system. For example, we can label nodes that include any of the operations we just mentioned: calls to open files, make database queries, load external resources, or require locks. Similar to security considerations, we can also label nodes that are using data that is dependent on other executions of the same program and therefore may have performance penalties due to locking.

There is no hidden complexity inside each node of the Control Flow Graph for `maitre_d`, so we can look at the nodes and edges in the graph to understand the program execution. Based on the original Control Flow Graph for `maitre_d`, in figure 7.2a, we have highlighted some nodes that could have obvious performance implications; the darker the filled node, the more severe the expected performance change. (Figure 7.2b shows the nodes separated by fill color.) We can look at the potential edge traversal options to see how significant the impact will be. The regular nodes either have no branching decisions or the length of the path for each decision is the same. The light-grey nodes `S4` and `E` have a single branching decision and the length of the path for each branch is slightly different. The dark-grey node `A` has a branching decision which will potentially have a more significant impact because the path lengths are potentially significantly different. The black node `C` has a branching decision which controls a loop; a potentially infinite edge traversal path.

We labeled this graph manually, but what would we do for a more complicated pro-

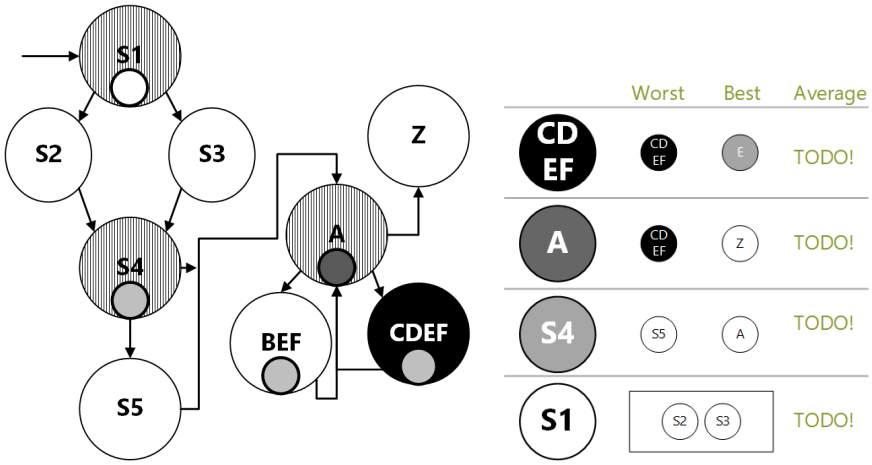
Figure 7.2 Graphing `maitre_d` Costs



(a) Control Flow Graph with costs **(b)** Nodes sorted by cost

gram? Finding the maximum edge traversal of a potentially cyclic graph, or *the longest path problem* is known to be NP-Hard. (Note that when we refer to a potentially cyclic graph, we are only looking at the subgraph inside the main loop, not the whole program: we *know* there is a main loop, but we can't guarantee that the service code doesn't include loops, such as node `C` in our example.) To make this solution generalizable, we can't impose any restrictions on the type of program, so there isn't a simple algorithm that we can use to determine the cost of path traversal for every case. No matter what algorithm is used to label the cost of each node, we can use this information when we convert a Control Flow Graph into a Configuration Impact Graph. Figure 7.3a shows the Configuration Impact Graph for `maitre_d` labeled with the cost of each configuration knob setting. Figure 7.3b also shows the minimum, maximum, and average costs of the nodes in the graph traversal path.

Figure 7.3 `maitre_d` Configuration Impact Graph with costs



(a) Configuration Impact Graph with costs

(b) CIG cost details

7.2 Modeling Performance

Most security considerations are not dependent on the number of concurrent users of a system; the configuration of a system usually enables a feature for all users or no users[7]. Security is sometimes connected to performance; for example a vulnerability may only be exploitable when there are other concurrent users (e.g. a race condition allows one user to access data belonging to another user), but that does not make the performance concern into a security concern. It is easier to find many security considerations directly in the program code, but even with the Configuration Impact Graph, performance considerations can be much harder to find.

[7]Users can be granted different permissions if the program has an *authorization* component, e.g. filesystem permissions limit access to a file, but anyone with access could change a file access timestamp if that feature is enabled.

7.2.1 Extrapolating from Current Load

A Configuration Impact Graph can tell us which nodes are expected to cause performance changes based on particular configuration values. One way we can verify is by measuring the number system calls for each node. Once there is a list of the expected number of system calls based on a small load, it might be possible for some programs to multiply starting point by additional load factors for each additional client and produce a worst-case estimate of the load on the entire server, but this also doesn't take into account the limits of each resource (e.g. file system, socket to database server, etc.) required to complete the system call. This may be best illustrated by the examples of the impact on actual services in chapter 9.

Chapter 8: Scoring and Communicating the Impact of a Knob

One of the best ways to communicate the importance of knobs is to assign each one a numeric score. This allows an administrator to sort the knobs and focus on the ones which are most important. Because there are many different types of effects a knob can have and there are different considerations for the importance of each knob, a single number isn't enough.

The Common Vulnerability Scoring System (CVSS) is the most well-known standard for evaluating the severity of security vulnerabilities. CVSS assigns metric scores - effectively a set of scores based on different facets of the vulnerability - which are then combined into a final score to allow administrators to understand both the overall severity of the vulnerability as well as understand more specifics about the method of attack, required skill of an attacker trying to exploit the vulnerability, and the worst possible outcome of a successful exploit. CVSS has received a lot of criticism for its accuracy and effectiveness and other scoring systems exist, but performance of CVSS has been studied [50] and CVSS continues to be used as the main scoring system for vulnerabilities in the Common Vulnerabilities and Exposures (CVE) database. The initial CVSS standard was created under the stewardship of the National Infrastructure Advisory Council, an advisory group under the United States Secretary of Homeland Security. The initial CVSS version 1 was released in 2005 with no peer review, but has since been managed by the Forum of Incident Response and Security Teams and has undergone regular updates.

CVSS in its own words [114]

The Common Vulnerability Scoring System (CVSS) captures the principal technical characteristics of software, hardware and firmware vulnerabilities. Its outputs include numerical scores indicating the severity of a vulnerability relative to other vulnerabilities.

Modeled on CVSS, the Uniform Parameter Scoring System (UPSS) captures the effects of changes to software configuration parameters. The long-term goal of its design is to output a numerical score indicating the significance of the consequences of changing the parameter, most useful when compared to the scores of other parameters of the same program.

We propose these metrics as a starting place and plan to adapt and revise as there is more real-world use, similar to the eventual development path taken by CVSS. The development of these standards was inspired by CVSS version 3.1 although there are significant changes to the way CVSS version 4.0 is calculated which are supposed to make it simpler [31]. (Each new CVSS scoring method relies on expert opinions collected from previous scores and thus is not possible with a newly created system.)

8.1 Initial Metrics

Metrics can be broken down into two categories: the changes intrinsic to the setting itself, and "external" considerations for the administrator choosing the correct value. They may reference the categories of effects which we discussed in section 2.4.1.

8.2 Effect Categories

Table 8.1: Effect Categories for Knob Scoring

Category	Description
Settings which only affect a Display value (GUI)	Someone says “Hey, that’s weird.”
Settings which affect the output of the program, but not performance or security (OUT)	The end-user (or client program) says “Hey, it’s broken!”
Settings which affect core functionality (FNC)	The administrator says “Hey, it’s broken!”
Performance Critical (PRF)	Settings which have a critical effect on performance
Security Critical (SEC)	Settings which have a critical effect on security

8.2.1 Intrinsic Characteristics

8.2.1.1 1. Affects Processor Usage (PU)

This metric measures the changes to the CPU load expected from changing a setting.

Table 8.2: Metrics for Affects Processor Usage (PU)

Impact	Description
Critical (C)	Changing this setting is expected to have an impact on CPU usage to the point of likely significantly (positively or negatively) impacting overall performance.
High (H)	Changing this setting is expected to have a measurable impact on CPU usage to the point of potentially significantly (positively or negatively) impacting overall performance.
Low (L)	Changing this setting is expected to produce a noticeable, but not critical, change in performance.
None (N)	Changing this setting is not expected to produce any (measurable) changes to performance.

Scoring Guidance: `Affects Processor Usage` should include settings which change

the expected processor cycles as well as the thread or process count.

Points of interest for Static Analysis:

- A `Low` score might be appropriate if a setting changes a significant number of lines of code/system calls executed.
- A `High` score might be appropriate if a setting directly changes the number of processes, threads, or arbitrary usage controls (i.e. `nice` parameters)
- A `Critical` score might only be able to be appropriately decided by an experienced developer or administrator.

8.2.1.2 2. Affects Memory Usage (MU)

This metric measures the changes to the RAM/SWAP usage expected from changing a setting.

Table 8.3: Metrics for Affects Memory Usage (MU)

Impact	Description
Critical (C)	Changing this setting will allow unlimited/uncontrolled memory usage.
High (H)	Changing this setting is expected to cause a calculated, "human" measurable (i.e. `free -h`, gigabyte level) change in memory usage.
Low (L)	Changing this setting is expected to cause a change in memory usage, but not at a scale significant enough to notice.
None (N)	Changing this setting is not expected to produce any (measurable) changes to memory usage.

Scoring Guidance: `Affects Memory Usage` is `Critical` if a setting can be changed that would allow run-away memory usage to the point of crashing the entire system and other reasonable values could be set which would prevent such a situation.

8.2.1.3 3. Affects Socket Usage (SU)

This metric measures how the program opens sockets/file descriptors, how many it can open, and in what locations.

Table 8.4: Metrics for Affects Socket Usage (SU)

Impact	Description
Yes (Y)	Changing this setting will change the number of open files, types of open sockets, or locations of open files.
No (N)	Changing this setting will not affect file/socket handling.

Points of interest for Static Analysis:

- Network sockets
- Any file handling operations
- Calls to change `ulimit`

8.2.1.4 4. Affects Input Sanitizing (IS)

This metric measures changes to how the program sanitizes user input.

Table 8.5: Metrics for Affects Input Sanitizing (IS)

Impact	Description
Yes (Y)	Changing this setting will change whether user input is trusted and/or how it is sanitized for dangerous values.
No (N)	Changing this setting will not affect input handling.

8.2.1.5 5. Does changing this setting cause the scores of other settings to change? (CO)

This metric measures whether this setting doesn't fall into any of the above categories itself, but changing it will cause changes to the other metrics.

Table 8.6: Metrics for Changes Others

Impact	Description
Yes (Y)	Changing this setting will likely directly change other metrics.
No (N)	Changing this setting will not directly affect other metrics.

Scoring Guidance: A good example of this is the Apache HTTPD MPM model and `.htaccess` processing. Besides that, this is open-ended.

Points of interest for Static Analysis:

- Loading DLLs/extensions

8.2.2 External Characteristics

8.2.2.1 1. Difficulty of Determining the Correct Value (CV)

This metric measures how hard it will be for an administrator who has never used this software to choose the correct value for this setting.

Table 8.7: Metrics for Correct Value

Impact	Description
Critical (C)	An administrator experienced in other areas is still likely to have difficulty choosing the correct value for this setting.
High (H)	An administrator experienced in other areas is likely to choose the correct value for this setting, but a less experienced administrator will have difficulty.

Impact	Description
Low (L)	Any administrator will be able to choose the correct value with minimal difficulty.
None (N)	The value of the setting has no practical changes to the program operation.

Scoring Guidance: Settings which only change a displayed value are likely to fall into the `None` value of the metric. Settings with a fixed range of values which are easy to test and change (see the next metric) may also have a `None` value.

Points of interest for Static Analysis:

- Consider other possible connections (or lack of connections) between difficulty of setting a value and its `effect category`.

8.2.2.2 2. Difficulty of Changing the Value Later (CL)

This metric measures the difficulty of changing a setting once it has been set and the program has run.

Table 8.8: Metrics for Difficulty Changing Later

Impact	Description
Critical (C)	Once this setting has been set, it may not be possible to change it.
High (H)	Once this setting has been set, it will be a complicated or lengthy process to change it, or it may require downtime to change.
Low (L)	Once this setting has been set, the complexity of changing it may vary, for example, depending on the length of the program has been run (e.g. database will need to be migrated)
None (N)	This setting can be changed easily and at any time.

Scoring Guidance: Consider whether it will be difficult to change this setting later, either because the steps required to change it are complex or not documented, or because

it will require significant effort, or because there is a high likelihood of the change going wrong and causing damage to the system or data because it will directly invalidate or render unreadable previously stored data.

This could also apply to software licensing tied to a particular hardware/software "thumbprint" (e.g. Windows activation or Proxmox activation) where it will not be possible to make changes without invalidating the license.

Points of interest for Static Analysis:

- Encryption settings - data encrypted at rest may be difficult or impossible to convert to a new encryption method.

8.2.2.3 3. Frequency of Change (FQ)

This metric measures whether a setting is often changed. (Obscure settings are often more dangerous and harder to get support for.)

Table 8.9: Metrics for Frequency of Change

Impact	Description
Never (N)	This setting is only used for specific uses (i.e. debugging during development) and is not intended for regular use.
Low (L)	This setting is usually only changed by experts on this program.
High (H)	This setting is commonly changed, but may require expertise to choose the correct value.
Always (A)	Anyone who uses this program is expected to change this setting.

Scoring Guidance: This is like the box on your tax form which says "this will not apply to most people".

8.3 Communicating the Results

8.3.1 Vector String

As inspired by CVSS, the values of each metric can be quickly communicated with a Vector String.

The UPSS v0.9 vector string begins with the label "UPSS:" and a numeric representation of the current version, "0.9". This is followed by `/EC:` and the abbreviation of the most severe `Effect Category` of the setting. Metric information follows in the form of a set of metrics, each preceded by a forward slash, "/", acting as a delimiter. Each metric is a metric name in abbreviated form, a colon, ":", and its associated metric value in abbreviated form. The abbreviated forms are defined above (in parentheses after each metric name and metric value).

For example, a setting which affects only a displayed value, has no effect concerns, is easy to change, and is commonly changed would produce the vector: `UPSS:0.9/EC:GUI/PU:N/MU:N/SU:N/IS:N/CO:N/CV:N/CL:N/FQ:A`.

A setting which turns on debug logging to disk with multiple separate files concurrently which will cause a significant impact to performance might produce the vector: `UPSS:0.9/EC:PRF/PU:C/MU:L/SU:Y/IS:N/CO:N/CV:N/CL:N/FQ:N`

8.3.2 Metric Values and Scoring Equations

For this rating to be the most useful to a system administrator, there needs to be a clear hierarchy of the importance of each knob. This can be done by converting each vector score to a single numeric score. Inspired by CVSS, the formula for calculating the final

numerical score is created by taking multiple examples of settings from programs we are very familiar with, deciding what qualitative rating we would expect them to receive, and assigning weights to each factor so as to achieve the expected results. The scoring algorithm for UPSS version 1 is still being developed. An excel spreadsheet is available with more calculation details [118].

CVSS v3.1 assigned values to each of their metrics using metrics based on real world vulnerability and usage data which is not (yet) available for UPSS. CVSS v4.0 makes the math even more complicated using multiple lookup tables which are also produced with the hindsight of almost 20 years of data.

Once a single numeric score is calculated, it can be useful to also map it to a textual representation. For the purposes of this initial version, we will adopt the scale and use policy used by CVSS [117], shown in table 8.10.

Table 8.10: Converting numeric scores to ratings

UPSS Score	Rating
0.0	None
0.1 - 3.9	Low Importance
4.0 - 6.9	Medium Importance
7.0 - 8.9	High Importance
9.0 - 10.0	Critical Importance

As an example, an UPSS Base Score of 5.0 has an associated severity rating of Medium. The use of these qualitative severity ratings is optional and there is no requirement to include them when publishing UPSS scores. They are intended to help administrators properly assess the consequences of a value change.

Completing the scoring tables for UPSS v1 is left for future work.

Chapter 9: Application to Real Servers

> *If you build a better mousetrap, the world will beat a path to your door.*
>
> – A quote attributed to, but not actually said by, Ralph Waldo Emerson[1]
>
> *Emerson was wrong: If you build a better mousetrap, the world will build a better mouse.*
>
> – Unknown[2]

Do these methods we presented actually improve on the status quo of configuration management and actually help us understand real-world applications? This isn't just a better mousetrap, this is a generalizable solution to understanding configuration knobs that can be applied to a variety of systems.

---

[1]Emerson actually said "If a man has good corn or wood, or boards, or pigs, to sell, or can make better chairs or knives, crucibles or church organs, than anybody else, you will find a broad hard-beaten road to his house, though it be in the woods" [33]. After his death, the mouse-trap quote started appearing attributed to him.

[2]This is probably a spin-off of another famous quote of unknown origin, "If you make something idiot-proof, someone will just make a better idiot". The first quote is also wrong about the popularity of the better mouse trap. It has been suggested based on statistics from the US Patent and Trademark office that mouse-traps are the "most frequently invented device in U.S. history"[47, 55], so there is no need to be beating a path to any one inventor's door. Similarly, Douglas Adams wrote "a common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools" [4]. This quote is in a footnote because, unlike almost every other quote, we actually know who wrote it.

Let us look at existing software packages and see how we can analyze and improve their configuration.

9.1 Apache `HTTPD`[3]

The Apache HTTPD server is one of the most popular web servers on the modern internet, at one point serving more than 70% of all internet traffic [77]. It is very flexible, allowing an administrator to load dynamically-linked extensions that significantly change the behavior of the server. These extensions can proxy requests to other applications, create internal sub-requests for other server resources, invoke other programs through their own dynamically-linked libraries, or consume and generate content in many other ways.

9.1.1 Reconfiguration based on Request Data

As of this writing, there are 707 configuration directives that can be used in an Apache HTTPD configuration. Many of these directives can be used multiple times, each instance applied to the internal directory path for the requested resource or based on a specific URL pattern. HTTPD allows many configuration items related to request/response handling to be specified either globally for all requests to the server, inside `<Directory>` blocks which will apply to on-disk paths, or inside `<Location>` blocks will which apply to the requested URL. Unlike many programs which read the configuration once and require a restart or `signal` to reload the configuration, HTTPD can also be configured to read a

[3]All references to HTTPD code are from the `2.4.x` branch - which is currently the actively developed version, available on GitHub [119], and are up to date as of 2024-02-29. HTTPD is licensed under the Apache License 2.0

configuration file - named `.htaccess` - in every directory, allowing on-the-fly changes to the configuration of the server. The main server configuration cannot be changed this way and there are limits, some hard coded and some set in the configuration itself, on what parts of the configuration can be changed, but it is possible for a user without access to the main configuration to significantly alter the behavior of the server.

9.1.2 Understanding the Configuration

As of this writing, the most recently released version of Apache HTTPD is 2.4.58, part of the 2.4 branch. The last version of Apache 2.2 was released in July 2017 and that version was declared end-of-life in January 2018. The last version of Apache 2.0 was released in 2013 and the last version of 1.3 was released in 2010. Despite the age of these end-of-life announcements, a significant amount of the developer documentation contains warnings that it has not been updated for the newest versions; some even from version 1.3 to 2.0 even though that was a significant change to the entire theory of operation of the program! Importantly, there is almost no documentation for the request processing flow for the currently supported version of the software [120]. An administrator who wants to understand more about how the server works cannot read the documentation and expect to understand the request flow and the effects of a configuration change.

For a large program such as HTTPD, a generated Control Flow Graph or Abstract Syntax Tree will be very large, so even a knowledgeable administrator won't be able to review the entire graph and it is critical to be able to find important knobs in a systematic way. Analyzing and graphing HTTPD also proves to be complicated due to its use of the Apache

Portable Runtime (APR). APR provides a standardized set of APIs that map to underlying operating system calls so that the HTTPD developers do not need to worry about different behaviors in different operating systems, but this has the effect of masking the actual system calls from the control flow graph. APR also implements a hook system which makes the graph more complicated. The hook system allows the incredible flexibility of the HTTPD module system as it allows modules to insert themselves almost anywhere in the request/response processing pipeline, but a lot of the HTTPD core is also implemented using hooks, just in case a module may want to use these functions too. In the case of `.htaccess` files, the function that actually parses them, `ap_parse_htaccess`, calls another function `ap_run_open_htaccess`. You won't find `ap_run_open_htaccess` by searching the source code - it is declared with a macro, `AP_IMPLEMENT_HOOK_RUN_FIRST(apr_status_t, open_htaccess,...`, which resolves to a loop running every function registered with the name `open_htaccess` until one of them succeeds. HTTPD uses the macro `APR_DECLARE_EXTERNAL_HOOK` to declare the implementation of the hook, which expands to the function `ap_hook_open_htaccess`. Because the function names created by the macros match and we are in the core of the program, not in a separate module, we may also assume that there is only one implementation of the hook and replace all the calls to the hook in the AST with calls to the underlying function.

By passing the `-save-temps` parameter to `gcc`, we are able to see preprocessed source `.i` files with expanded APR macros. We trace the `open_htaccess` function to `ap_pcfg_openfile`, which in turn calls the operating system dependent `apr_file_open` function and then confirms that a file was actually opened. The `apr_file_open` function handles OS-dependent functions relating to setting the file mode, handling locks for thread-safety,

opening the file, and setting hooks for cleaning up the memory when the file is closed.

As an aside, if we run these files through the `clang -ast-dump` function and filter for any functions which are called that are `extern[al]`, indicating that the function is supplied by the kernel or stdlib. These functions usually are related to program IO or process or memory control (shared memory, forks, threads, etc.) and are therefore more likely to be critical to the performance or security of the program, as we discussed in earlier sections on security and performance.

9.1.3 Performance of Dynamically Changeable Configuration

Because the configuration can be changed significantly at runtime, it is difficult to model the behavior of the webserver the way it is generally used. We will start by modeling the additional overhead of just reading the `.htaccess` files. Many administrators use `.htaccess` files even if they have access to the main server configuration because it is faster and it allows configuration values to be bundled with individual applications. If system administrators were aware of the potential performance penalties of having `.htaccess` enabled, they might decide that it is better to put all the configuration changes in the main configuration files.

During every request, the server refers to an in-memory cache of the entire configuration, checking each component of the URL and the on-disk path of the requested resource against the global configuration and all of the `<Directory>` and `<Location>` sections. As part of this process, when configuration overrides are enabled - using the `AllowOverride` or `AllowOverrideList` directive - as the server traverses the in-memory directory config-

uration from the root down to the directory of the requested resource, it tries to open the `.htaccess` file in each directory and merge its contents with the in-memory configuration for the current request. To improve performance, the directives found are cached for a single request and any internal subrequests in HTTPD, but they are not cached from one request to another (`request.c`, `prep_walk_cache`). This means the performance of a single request is expected to change based on the depth of the path where the requested resource is located.

For this test, we use a short URL, http://10.96.6.12/1.txt, pointing to a text file with the content "1\n", and a long URL, http://10.96.6.12/1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19/20/21.txt, pointing to a text file with the content "21\n". For the initial tests, there are no `.htaccess` files present anywhere in the directory structure. For the subsequent tests with `.htaccess` files, they alternate down the directory tree having the contents `Require all granted` and `Require all denied`.

The code listing below provides a small window into how this processing works.

```
1067 /* If .htaccess files are enabled, check for one, provided we
1068  * have reached a real path.
1069  */
1070 do {  /* Not really a loop, just a break'able code block */
1071
1072     ap_conf_vector_t *htaccess_conf = NULL;
1073
1074     /* No htaccess in an incomplete root path,
1075      * nor if it's disabled
1076      */
1077     if (seg < startseg || (!opts.override
1078         && apr_is_empty_table(opts.override_list)
1079         )) {
1080         break;
1081     }
1082
1083
```

```
1084        res = ap_parse_htaccess(&htaccess_conf, r, opts.override,
1085                                opts.override_opts, opts.override_list,
1086                                r->filename, sconf->access_name);
1087        if (res) {
1088            return res;
1089        }
1090
1091        if (!htaccess_conf) {
1092            break;
1093        }
1094
1095        /* If we are still here, we found our htaccess.
1096         *
1097         * Calculate our full-context core opts & override.
1098         */
1099        core_opts_merge(htaccess_conf, &opts);
1100
1101        /* If we merged this same htaccess last time, reuse it...
1102         * this wouldn't work except that we cache the htaccess
1103         * sections for the lifetime of the request, so we match
1104         * the same conf.  Good planning (no, pure luck ;)
1105         */
1106        if (matches) {
1107            if (last_walk->matched == htaccess_conf) {
1108                now_merged = last_walk->merged;
1109                ++last_walk;
1110                --matches;
1111                break;
1112            }
1113
1114            /* We fell out of sync.  This is our own copy of walked,
1115             * so truncate the remaining matches and reset
1116             * remaining.
1117             */
1118            cache->walked->nelts -= matches;
1119            matches = 0;
1120            cached = 0;
1121        }
1122
1123        if (now_merged) {
1124            now_merged = ap_merge_per_dir_configs(r->pool,
1125                                                  now_merged,
1126                                                  htaccess_conf);
```

```
1127      }
1128      else {
1129          now_merged = htaccess_conf;
1130      }
1131
1132      last_walk = (walk_walked_t*)apr_array_push(cache->walked);
1133      last_walk->matched = htaccess_conf;
1134      last_walk->merged = now_merged;
1135
1136  } while (0); /* Only one htaccess, not a real loop */
```

We run performance monitoring on HTTPD by building with symbols included and compiler optimizations mostly disabled (`CFLAGS="-Og -pg -fno-omit-frame-pointer -g" ./configure --enable-mpms-shared=all`).[4] We use only the default-enabled extensions and the prefork MPM. We then use the Apache Benchmark tool (`ab`) to make requests for a text file to the server on the local IP address, using a short path (directly in the document root) and a long path (20 directories deep), both with `AllowOverride off` and with `AllowOverride on`. Each test was run once 10,000 times and twice 100,000 times to ensure there was no variability between tests. All tests were run on a single Dell PowerEdge R330 with and Intel Xeon E5-2667 v3 running a fresh installation of Rocky Linux 9. SELinux is enabled.[5]

---

[4] We also save the intermediate files with macros expanded and dump the control graph, so the full command is `CFLAGS="-Og -pg -fno-omit-frame-pointer -g -fdump-tree-all-graph -save-temps" ./configure --prefix /srv/httpd --enable-mpms-shared=all`

[5] Many modern Linux distributions ship with SELinux enabled in enforcing mode by default. However, even if SELinux is in permissive mode, HTTPD itself enforces SELinux policies internally, so this doesn't have a significant performance impact.

Table 9.1: Runtimes of `.htaccess` processing (10k/100k/100k)

URL	AllowOverride None	AllowOverride All
Long URL	1.367 seconds	2.002 seconds
	/ 13.637 seconds	/ 20.015 seconds
	/ 13.607 seconds	/ 20.032 seconds
Short URL	1.283 seconds	1.370 seconds
	/ 12.981 seconds	/ 13.581 seconds
	/ 12.989 seconds	/ 13.590 seconds
Long URL with `.htaccess` files present	1.359 seconds	3.062 seconds
	/ 13.572 seconds	/ 31.042 seconds
	/ 13.640 seconds	/ 31.122 seconds
Short URL with `.htaccess` files present	1.287 seconds	1.431 seconds
	/ 12.894 seconds	/ 14.487 seconds
	/ 12.899 seconds	/ 14.461 seconds

Table 9.2: `perf report` Event Count

URL	AllowOverride None	AllowOverride All	AllowOverride All with `.htaccess`
Long URL	117,315,746,065	160,914,969,105	235,685,047,836
Short URL	113,810,262,509	117,411,874,763	124,024,410,562

As seen in the results in table 9.1, there is a 5%-6% performance penalty between serving a file in a very deep path, compared to a shallow path, even with `.htaccess` processing disabled. This entire difference can be accounted for in processing the in-memory configuration and looping through the configuration for each directory, and the slightly larger payload due to the longer filename. There is also a 5%-6% performance penalty between serving a file in a shallow path with `.htaccess` processing enabled, compared to `.htaccess` processing disabled. However, when we look at the performance penalty for a long path with `.htaccess` processing enabled, even with no `.htaccess` files present, we see a more than 30% increase. Similarly, while there is a 5% penalty for serving a

deep path file over a shallow path file without `.htaccess` processing, there is again more than a 30% penalty in those two cases when `.htaccess` files are processed. Because the difference is almost the same between the two cases, it is clear that the major contributing factor is the attempt to check for and load the `.htaccess` files in all the intermediate directories.

Analyzing the `perf` sampling data collected from the shallow path requests with `.htaccess` processing enabled, we were not able to capture `ap_directory_walk` calling `ap_parse_htaccess` for these requests, even after running the test many more times, showing how insignificant the parse call is (because it isn't using a significant amount of processing time, it is less likely to show up in `perf` sampling). Summing the execution percentages for the `ap_directory_walk` functions given by the sampler, we see those function calls being responsible for more than 2% of the program execution, with the rest of the difference made up of the kernel checking if it can open the requested file.

Analyzing the data collected from the deep path requests, `perf` sampling with `.htaccess` processing enabled shows that the `ap_directory_walk` function and the `htaccess` parsing functions it calls are responsible for 3% of the total execution time of the program and the rest of the time difference being made up of kernel IO calls. Syscalls related to checking for and loading `.htaccess` files made up the largest share of all function calls in the entire `perf` recorded dataset for these requests - the total number of IO system calls in the `perf` sample effectively doubled, from 436 to 858, when compared to the shallow path sampling. For all three other types of requests, almost all of the IO calls were related to sending the HTTP response over the network socket.

The addition of `.htaccess` files to be parsed and added to the configuration makes an

even larger impact, in these tests literally doubling the amount of wall-clock time and the number of events counted by `perf` over the case of having `.htaccess` files disabled.

9.1.4 Locating critical sections in HTTPD

Now that we have shown that a single configuration change, allowing `.htaccess` files to be used, can make such a significant difference in the performance of the application, specifically due to increased kernel IO calls in a loop, we can create a configuration impact graph and use it to search the code for other places where there is a similar structure to the graph, indicating potential for a similar issue. We then flag the configuration entries that control these loops as being more important for the performance of the application.

9.1.5 Scoring

Since we have located `AllowOverride` as a significant configuration knob, we can look at how it might be rated with UPSS to explain its importance to an administrator.

Table 9.3: UPSS Metrics for HTTPD's `AllowOverride` directive

Metric	Score	Reason
Effect Category	SEC	`.htaccess` can affect all the different categories. `Security Critical` is the most severe.
Processor Usage (PU)	H	`.htaccess` processing is commonly turned on, but we have shown that it can have a significant impact on processing speed.
Memory Usage (MU)	L	`.htaccess` files are usually very small, so loading them doesn't consume significantly more memory.
Socket Usage (SU)	N	Listen settings cannot be changed in `.htaccess`.
Input Sanitizing (IS)	Y	Input and Output filters can be modified in `.htaccess`.

Metric	Score	Reason
Other Settings (CO)	Y	The entire point of `.htaccess` is to change other settings.
Correct Value (CV)	H	This setting is usually left enabled even if it is not needed. Many administrators will not understand how or will not be willing to combine the `.htaccess` supplied by an application into the main server configuration.
Change Later (CL)	L	Settings from an existing `.htaccess` file may need to be migrated into the server configuration.
Frequency of Change (FQ)	A	The choices for this setting are straightforward, are frequently changed, and are easy to get support for.

The resulting vector score for `AllowOverride` is

`UPSS:0.9/EC:SEC/PU:H/MU:L/SU:N/IS:Y/CO:Y/CV:H/CL:L/FQ:A`.

Based on a non-scientific survey of experienced administrators provided with this information, we would expect this to result in a UPSS overall rating of `High Importance`.

9.2 `HAProxy`[6]

HAProxy is a high-performance frontend web proxy, commonly used as a load balancer in front of multiple webservers. It supports advanced request handling features to split requests between different backend servers based on almost any metadata (or data) supplied in the request.

I was responsible for setting up a set of highly-available HAProxy servers to replace

[6]All references to HAProxy code are from the `2.0` branch - which was the actively developed version when I was first configuring this service for the CS department, available on GitHub [126]. HAProxy has developed very quickly since then and some of the specific configuration knobs discussed here have changed. HAProxy is licensed under the GPL and LGPL licenses.

the department's proprietary load balancers that were no longer supported by the manufacturer. We started with new server hardware that should have been able to handle hundreds of thousands of connections at once, but when we moved all of our web traffic to these systems, traffic slowed to a crawl. We were able to see that the servers were not overloaded, but the HAProxy processes had a `ulimit` - in this case a maximum number of open file handles - enforced by the kernel which was limiting the number of requests that could be handled. After reading the manual - which is organized several separate parts, each basically in alphabetical order - at length, we found the parameter `maxconn`, controlling the maximum number of connections the server would accept, was set by default to be very low (this is no longer the case as of HAProxy 2.0) and this would cause the program to request a lower `ulimit` when it started up. It took a while to find this important information - it isn't in the "Starter Guide" for the program and all the examples in the "Configuration Guide" use relatively low `maxconn` values - and once we changed that value, our performance problems instantly went away.

9.2.1 Understanding the Configuration

HAProxy is much simpler than HTTPD. In this particular case, HAProxy changes the resource limits using the `setrlimit` call, as seen in listing 9.1. Tracing the source for the parameters to this function, we can see it comes from the value of `global.rlimit_nofile`, which is itself set either by parsing it directly from the configuration file (Listing 9.2) or calculated from other configuration values (`global.maxsock`).

This is a relatively straightforward example of a single setting that an administrator

Listing 9.1 HAProxy ulimit settings

```
/* ulimits */
if (!global.rlimit_nofile)
    global.rlimit_nofile = global.maxsock;

if (global.rlimit_nofile) {
    limit.rlim_cur = global.rlimit_nofile;
    limit.rlim_max = MAX(rlim_fd_max_at_boot, limit.rlim_cur);

    if (setrlimit(RLIMIT_NOFILE, &limit) == -1) {
        /* try to set it to the max possible at least */
        getrlimit(RLIMIT_NOFILE, &limit);
        limit.rlim_cur = limit.rlim_max;
        if (setrlimit(RLIMIT_NOFILE, &limit) != -1)
            getrlimit(RLIMIT_NOFILE, &limit);

        ha_warning("[%s.main()] Cannot raise FD limit to %d, limit is %d.\n",
            argv[0], global.rlimit_nofile, (int)limit.rlim_cur);
        global.rlimit_nofile = limit.rlim_cur;
    }
}
```

would need to know to change in order to significantly improve performance.

A more complicated setting is `cpu-map` - this allows the administrator to manually set how many processes or threads there should be and set the affinity for each one to a specific processor core. It seems like an administrator would want to set up and directly "attach" one process to each core, but depending on the actual CPU in use, this could result in less-than-ideal performance for some users. Some modern CPUs mix cores with multiple speeds - referred to as Performance Cores (p-cores) and Efficiency Cores (e-cores). A process with affinity to an e-core will probably not perform as well as a process with affinity to a p-core. The code for this operation is long and complex, so we will not include it here, but you can find it in a very similar way to the `maxconn/ulimit` code we

Listing 9.2 HAProxy configuration parsing

```
else if (!strcmp(args[0], "ulimit-n")) {
    if (alertif_too_many_args(1, file, linenum, args, &err_code))
        goto out;
    if (global.rlimit_nofile != 0) {
        ha_alert("parsing [%s:%d] : '%s' already specified. Continuing.\n",
           file, linenum, args[0]);
        err_code |= ERR_ALERT;
        goto out;
    }
    if (*(args[1]) == 0) {
        ha_alert("parsing [%s:%d] : '%s' expects an integer argument.\n",
            file, linenum, args[0]);
        err_code |= ERR_ALERT | ERR_FATAL;
        goto out;
    }
    global.rlimit_nofile = atol(args[1]);
}
```

talked about first. HAProxy uses the `sched_setaffinity` function to tell the kernel how to schedule these threads/processes. The call to this function is inside an if statement in listing 9.3 which is guarded by several variables set by configuration entries: `nbproc` and `cpu_map`.

Listing 9.3 Code listing showing how processor affinity is guarded

```
if (proc < global.nbproc &&  /* child */
proc < MAX_PROCS &&       /* only the first 32/64 processes may be pinned */
global.cpu_map.proc[proc])    /* only do this if the process has a CPU map */
```

9.2.2 Scoring

How would these two directives for HAProxy score?

9.2.2.1 `maxconn`

Table 9.4: UPSS Metrics for HAProxy's `maxconn` directive

Metric	Score	Reason
Effect Category	PRF	Limiting the number of available sockets significantly impacts performance.
Processor Usage (PU)	L	This setting itself does not affect performance, but allows more concurrent requests which could cause higher load.
Memory Usage (MU)	N	This setting does not change the amount of memory HAProxy uses as all buffers are the same size.
Socket Usage (SU)	Y	This allows more files to be opened.
Input Sanitizing (IS)	N	
Other Settings (CO)	N	
Correct Value (CV)	H	Administrators might need to experiment with this setting to find a reasonable value.
Change Later (CL)	N	
Frequency of Change (FQ)	A	You wouldn't know it from the manual, but changing this setting is required to good performance.

The resulting vector score for `maxconn` is

`UPSS:0.9/EC:PRF/PU:L/MU:N/SU:Y/IS:N/CO:N/CV:H/CL:N/FQ:A`.

Based on a non-scientific survey of experienced administrators provided with this information, we would expect this to result in a UPSS overall rating of `Medium Importance`.

9.2.2.2 `cpu-map`

Table 9.5: UPSS Metrics for HAProxy's `cpu-map` directive

Metric	Score	Reason
Effect Category	FNC	The administrator will not see full CPU utilization, but performance may be fine depending on load.

Metric	Score	Reason
Processor Usage (PU)	L	This setting itself does not affect total load, but may allow requests to be served faster.
Memory Usage (MU)	N	This setting does not change the amount of memory HAProxy uses as all buffers are the same size.
Socket Usage (SU)	N	
Input Sanitizing (IS)	N	
Other Settings (CO)	N	
Correct Value (CV)	L	One process per processor is usually a good choice (for a dedicated server).
Change Later (CL)	N	
Frequency of Change (FQ)	H	`auto` should be the default value, but isn't.

The resulting vector score for `cpu-map` is

`UPSS:0.9/EC:FNC/PU:L/MU:N/SU:N/IS:N/CO:N/CV:L/CL:N/FQ:H`.

Based on a non-scientific survey of experienced administrators provided with this information, we would expect this to result in a UPSS overall rating of `Low Importance`. (We got feedback that many administrators consider this a "micro optimization" that is only necessary on the busiest servers.)

Chapter 10: Conclusion

In this dissertation, we have presented the complexity of understanding and analyzing configuration management for system administration, and explained why there is limited scientific study in this area, and showed the complexities of beginning this study.

We introduced a framework for studying the impacts of configuration management; Configuration Impact Graphs, an novel evolution of Control Flow Graphs that allows us to focus on only the effects of configuration knobs on the execution of a program. Configuration Impact Graphs allow us to reason about the execution of a program by helping separate the control an administrator has over the execution of a program through configuration from the code written by a developer and the data supplied by an end user. We can do all of that just based on the way each particular piece of information affects the control flow of the program.

We also introduced a system for determining a hierarchy of configuration knob importance using a standardized scoring system. Scores create as part of the Universal Parameter Scoring System can be created automatically by static analysis process, manually flagged by developers or knowledgeable administrators, or a combination of both. Both of these provide an entirely new level of insight into program execution built around the system administrator as the actor. We have shown how we can apply both of these innovations on sample programs and real-world production systems.

Now that we have a framework to measure the effects of configuration knob settings,

we expect to build on these analysis techniques to be able to detect the effects of configuration knobs with decreasing human intervention. This will allow more reliable predictions about system performance and security and give system administrators more insight into the systems they operate. We expect to generalize these concepts in ways that can be applied not just to single applications, but to entire systems running multiple applications. As the operations of each program can be better understood, this will allow us to provide a system administrator with recommendations for balancing resource allocations between multiple programs. We will also discover more about how to detect unsuitable configuration knob settings to provide further insight on how to manage systems securely and performantly.

Attempts to study system administration in a scientific setting are often rebuffed with the objection that it clearly isn't a scientific problem worthy of study because so little attention has been paid to the area up to this point (and there must not be any basic research left to be done in such an area). This type of study is also difficult because system administrators themselves are hard to study and there are not many people who cross over between the worlds of research and system administration. Experienced system administrators don't trust automated tools to do a system administrator's job, so any system which wants to provide advice to an administrator about how a program or service should be configured needs to be transparent, reliable, and must be configurable itself. Software only gets more complex as time goes on, as evidenced by the ever-increasing number of knobs and there is an entire field of configuration analytics waiting to be studied to help improve the security and performance of the systems we rely on every day.

Appendix A: Anecdotes of System Administration

The trouble with any research connected to the work of system administrators is the prevalence of anecdotes and the lack of data. Entire newspaper columns exist to collect anecdotes of system administrator mistakes and misbehaviors (e.g. the *Who, Me?* and *BOFH* columns in The Register).

A.1 Everybody does it!

It is taken as a given in many studies [e.g. 35] that Google Search is the starting point for programmers and administrators when they have questions about how to do something, although data on the prevalence of this is hard to come by. From my own experience, system administrators (and many other tech-savvy people) are more likely than an average user to disable tracking/telemetry features, so it is hard to measure user behavior directly. (This is potentially why it seems that large software companies remove power-user-friendly features - they can't see data showing that the features are actually being used.) Based on the prevalence of IT support discussions, presentations, and memes which all say that Google Search is the techie's go-to resource to resolve all problems, this seems to be a valid conjecture.

A.2 Anecdotes of Uninformed Configuration

These tales are almost always told anonymously, and while they frequently illustrate a lack of understanding by an administrator who should have a better understanding of the environment and many perfectly describe uninformed configuration, they lack the detail - and certainly reproducibility - necessary to serve as a basis for scientific study.

Unless otherwise cited, each of these examples was directly experienced by the author or told to the author by a participant:

A.2.1 Software Configuration and Physical Danger

It seems tenuous to link uninformed configuration by a system administrator to a risk to life or limb, but there are many examples. The first software-controlled radiation treatment machine, the Therac-25, killed or significantly injured at least 6 patients. [62] It turns out that this was because operators became very familiar with the interface and were able to enter configuration values faster than the programmers expected and would use the arrow keys to go back and correct mistakes, sometime triggering a race condition that would set the physical controls on the machine incorrectly - for our purposes, that error itself is end-user error. The administrators fixed it by removing keys on the keyboard to prevent users from going back. Zelkowitz described [110] this and other medical and technology related disasters as failures of good programming practice and failure to learn from the past, but the same applies to system operators setting configuration values. Zelkowitz, Neumann [78] and others also bemoan the apparent failures of new generations of software engineers and operators to learn from past mistakes and disasters,

even though the risks can be just as significant.

Configuration values can also include user authorization policies: > A large police department installed Computer-Aided Dispatch systems in their patrol cars. > Dispatchers used this system to send information about calls to the officers and officers were only supposed to be able to update information about calls they were assigned to. > A police officer discovered that he could activate a particular mode that would make the system think he was a dispatcher even from the computer in his patrol car and would make creepy pictures pop up and sounds play in other officer's cars. > This led to some near crashes and also caused officers to distrust the system that was supposed to help them respond more quickly to incidents. > The system should have been configured to only allow supervisors to send these types of messages, but administrators were not familiar enough with the system and believed it would be too complicated, or believed the feature could not be abused and changes to the authorization settings were not necessary.

When faced with the challenge of a user not having sufficient permissions to complete a task, many administrators will react by granting very broad permissions to the user, often to parts of the system which the user should never need to access. The same phenomenon can also be seen in physical access control systems - for example, a user will need access to a building outside of the usual schedule, so they are granted unlimited access to the entire building because more granular permissions are more difficult to set.

A.2.2 Rushed Changes

Many programming languages have safeguards in place to prevent programmers from making simple mistakes when working quickly or with minimal understanding of the language or framework. Type-safe languages force the developer to be clear about what type of data they think they are operating on. Many languages that are not strictly type-safe have added warnings when an object of an unexpected or unusual type is encountered. Even where the program or framework doesn't enforce "good behavior" by the operator or software developer, many programming and operations manuals have a *Getting Started* section detailing the minimal steps required to start using a piece of software. The *Getting Started* sections rarely have way to communicate the consequences of a single change on the operation of the entire program.

> An organization was migrating from a proprietary website load balancing solution to HAProxy on several powerful servers that should easily have been able to handle the current load and future growth. The manufacturer of the old load balancer also would not renew the license for the product because it was considered End-of-Life. The plan was to move the websites to the new load balancer individually to make sure the new frontend would not cause any trouble with individual web applications, but the work took longer than expected. The organization realized that they would not finish the move before the load balancer license would expire and the load balancer would stop working, so they moved the rest of their websites during an evening maintenance window the night before the license expired. In the morning, the organization's web-

sites appeared sluggish, but the system resources on the new servers were not exhausted. The organization could not move back to the old load balancers because the license was expired and had to debug and fix the new servers in a production setting. In the end, the entire issue was resolved by changing a single value in the HAProxy settings. This could have been avoided if system administrators were able to see how critical this setting was.

Programmers and system administrators often have to focus on a problem and see it to completion due to what Ortega describes as the desire to make “Progress” [80], but the line between self-sacrifice and “crunch time” becoming “plan continuation bias” or “get-there-itis” is very thin. Harford describes [45] “get-there-itis” as the desire to finish the current attempt without being able to objectively see and analyze the challenges involved. The administrator or programmer will lose any pretense of discipline, ignoring spelling and grammar [79], ignoring standardized procedures, ignoring other possible solutions to the current problem, ignoring any possible problems that will be caused by the current fix, and completely skipping testing the proposed solution completely.

Parnas concludes [82]:

- Until customers demand evidence that the designers were qualified and disciplined, [customers] will continue to get sloppy software.
- As long as there is no better software, [customers] will buy sloppy software.
- As long as [customers] buy sloppy software, developers will continue to use undisciplined development methods.

- As long as [customers] fail to demand that developers use disciplined methods, [customers] run the risk—nay, certainty—that [customers] will continue to encounter software full of bugs.

The same is true of configuring systems.

A.2.3 Inadequate Visibility

It is easy to say that software design is to blame for uninformed configuration decisions. Sometimes an administrator will run into issues of uninformed configuration because they don't know everything about the environment, rather than not knowing everything they need to know about the software. Sometimes being able to know more information about the environment is not possible, but would be more important for ensuring the best performance or security outcomes.

> Enterprise organizations may delegate IT operations to individual departments. At this university, the Computer Science Department operates its own network and the university's Division of IT acts as an upstream internet service provider. The only network interaction between the two groups is with Layer 3 routing using the Border Gateway Protocol (BGP). Improper BGP configuration has the ability to bring down the internet on a global scale [13, 24], so DIT and the department both implement routing best practices by filtering advertised and accepted routes. None of this considers the CPU load of the router. Processing routing changes is known to cause high CPU load on routers and manufacturers may make recommendations of an appropriate

load cushion to ensure the router continues to function. Each side of the layer 3 route does not have the ability to see the CPU load of the router on the other side of the connection, so it is possible for a simple routing change on one side to crash the router on the other side, and it isn't always clear what settings changes would cause this to happen.

Appendix B: Compile-time Configuration vs. Run-time Configuration

We established previously that modern system administrators are unlikely to change compile-time configuration options. Why would a developer choose compile-time configuration over run-time configuration? Here is a non-exhaustive summary of some major considerations.

B.1 Open Source

The source of the program must be available in order to compile it. This clearly would not apply to most commercial software packages.

B.2 Libraries

There is a well-known maxim among software developers: “don’t ever roll your own encryption”, and developers who ignore this warning often end up with significant security flaws in their software. This is because doing encryption properly is **hard**. In other areas, it could be complicated or annoying to implement software to properly handle complex standards. From my own experience, this might include email address validation, MIME email parsing or creation, or URL parsing and routing, to name a few examples. Instead of creating these parts of a program from scratch, developers are encouraged to make use of libraries available for this purpose. (There is such a thing as too many libraries and using libraries can also create a supply-chain security risk, but those are both outside the

scope of this discussion.) Some of these libraries may only be needed when packaging a program and not when it is running, so it makes sense to configure and use the library as part of the compilation and packaging process rather than include the library as a dependency to run the program.

For example, a web application might use `SASS` to define display styles, but it must be compiled to `CSS` for the browser to be able to use it. The SASS library would be configured at compile-time and the compiled CSS would be shipped with the software rather than shipping both the SASS source and the SASS compiler.

B.3 Overhead (or over-worked?)

Many system administrators can go an entire career without needing to compile custom software and it may not be considered part of the administrator's job, or it may not be a high priority to figure out if pre-compiled software is available. Compiling software may require commercial licensing which is not required to just execute the program (i.e. commercial compilers). Even if the toolchains themselves are available, they may be difficult to set up or may not operate on every architecture or operating system (i.e. the software is intended to be cross-compiled).

B.4 Too Many Options

Many options would just not be useful if they had to be compiled into a software package rather than being set at runtime. Other options which are never modified in the regular use of the program may be more efficient to set at compile time. Even if it would be more

efficient to set some options in compile-time configuration, the increased complexity of distributing many versions of the compiled program may outweigh those benefits.

B.5 Program Bloat

If a program has many options which can never be used together, compile-time configuration can allow sections of the code that can never be used to be removed, resulting in a smaller binary/object size.

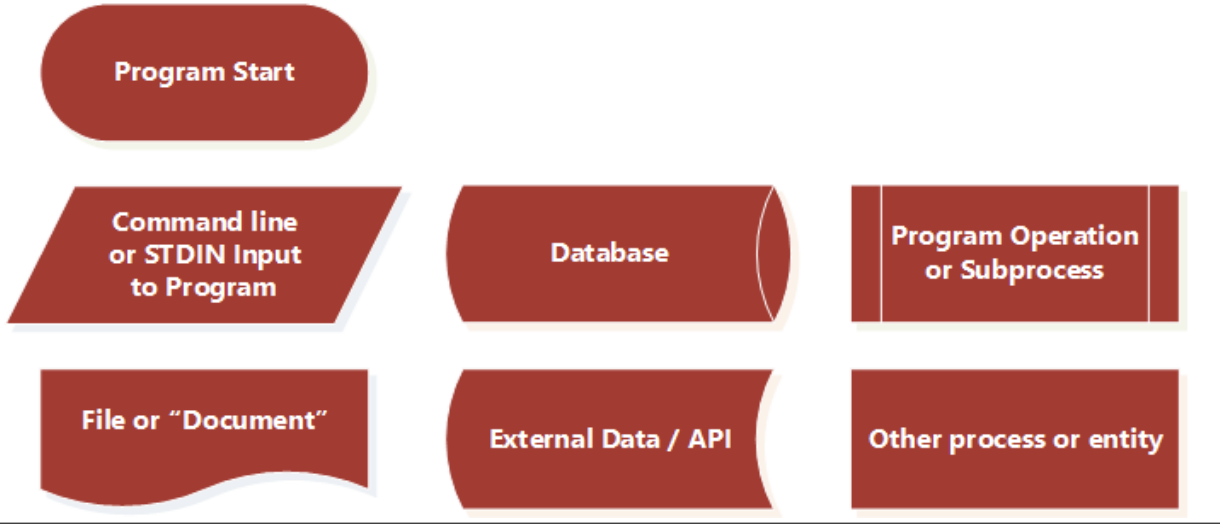
Appendix C: Drawing Shapes

We try to use shapes to mean the same things in every drawing. We try not to use colors to indicate an important part of a drawing because we want the drawings to be useful when printed in black and white or to those who are colorblind. We may use colors to group similar elements of a drawing if it is not that important to helping understand the meaning.

Figure C.1 shows common shapes that we use in diagrams throughout this work. Notes about specific shapes:

- `File or "Document`: When dealing with a webserver, the output from the program is often a document, so we use the same symbol for the program output.

Figure C.1 Shape List for Program Diagrams



Control Flow Graphs (and by extension Configuration Impact Graphs) are often as-

sumed to have a single entry node which, by convention, is usually at the top. Due to figure generation constraints, our convention is that an entry node for a graph will either have an arrow pointing in with no source node or will be a double circle. An exit node will be a square.

Appendix D: Code Listings

D.1 `maitre_d`

This is the code listing for the `maitre_d` server[1], followed by the labeled Configuration Flow Diagram which corresponds to its execution. This program is used as an example mainly in chapter 5, where the CFG labelling is also explained.

```
void main()
{
    // S1
    $CONFIG = load_configuration_from_file("/etc/maitre_d/server.conf");
    assert_true(validate_configuration());
    $socket = open_socket($CONFIG["listen"]);
    setup_signals([ "exit" => &close_program, ])

    match ($CONFIG["mode"]) {
        // S2
        case "static": {
            create_static_content_caches();
        }
        // S3
        case "dynamic": {
            load_dynamic_content_generators();
        }
    } // end match ($CONFIG["mode"])

    // S4
    if ($CONFIG["threaded_mode"]) {
        // S5
        setup_thread_safe_data_structures();
    }
```

[1] A maître d' in a restaurant is combined "host, headwaiter and dining-room manager" and could be considered a type of server.

```
// L
fork_or_thread($CONFIG["threaded_mode"]);

// A - beginning of Main Loop
while (true) {
    if ($SIGNALS["exit"]) {
        // Z
        close_sockets_and_free_memory();
        return 0; // exit
    }

    // This function blocks until a new request is received
    $request = get_request_from_socket($socket);
    $output = "HTTP/1.0 404 Not Found";

    match ($CONFIG["mode"]) {
        // B
        case "static": {
            $output = get_content_from_cache($request);
        }
        // C
        case "dynamic": {
            // directory_list("/a/b/c") => ["/a/b/c/", "/a/b/", "/a/", "/"]
            $directory_list = directory_list($request);
            foreach ($directory_list as $directory) {
                if ($script = has_executable_script($directory)) {
                    // D
                    $output = execute_script($script);
                    break;
                }
            }
        }
    } // end match ($CONFIG["mode"])

    // E
    send_to_client($output);

    // In this application, logging is done entirely based on attributes
    // of the request and response and is not configurable.
    if (should_be_logged($request, $response)) {
        // F
        Log::information($request, $response)
```

```
        }
    } // end while(true)
} // end main()
```

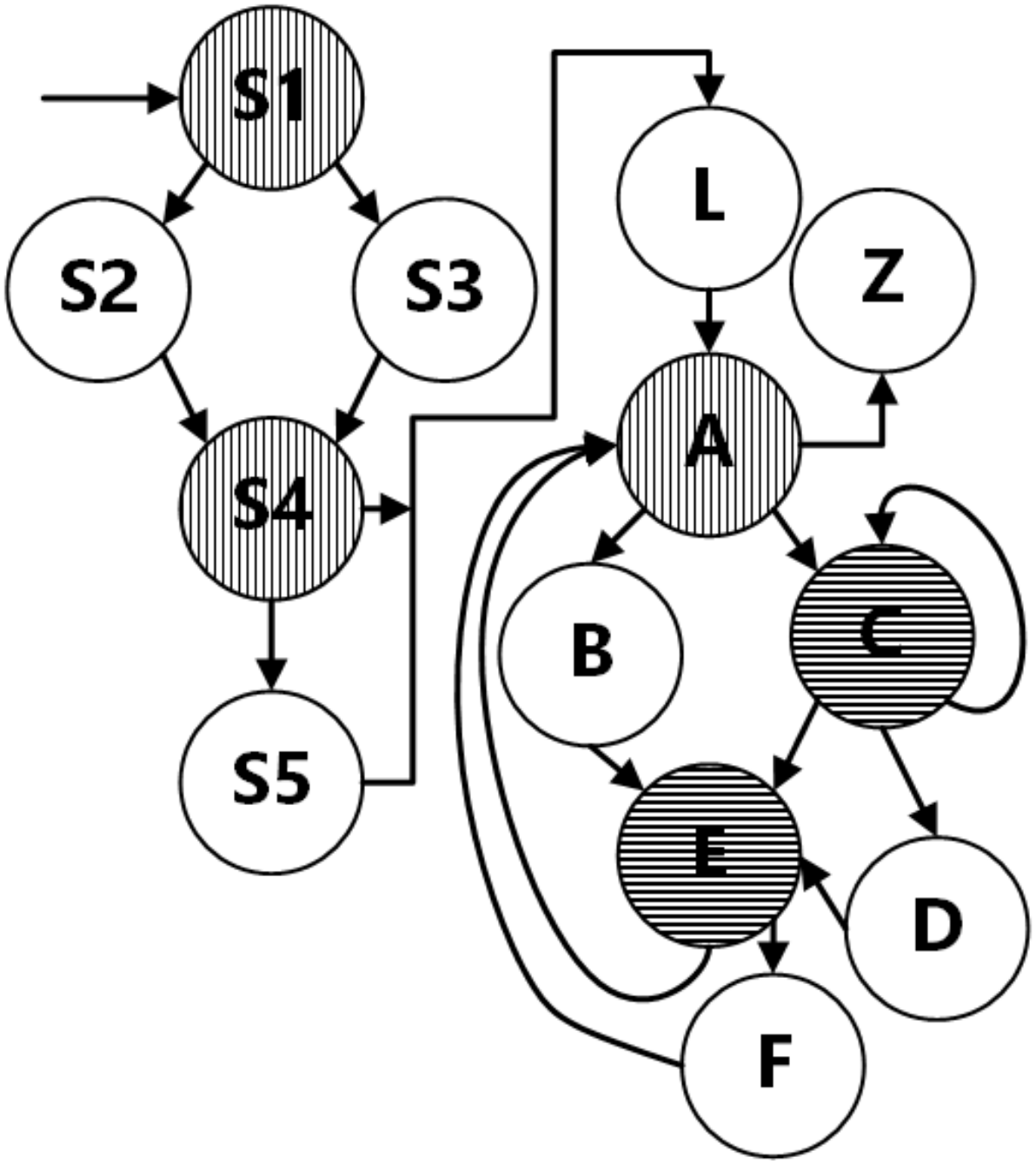
Program and Graph Notes:

- To make the graph and code easier to read and compare, in some versions of the graph we include a node labeled `L` to make a clear distinction of where the setup phase ends and the main loop begins, but it does not actually affect control flow and it is not included in all versions of the graph.
- In a standard program, you would check the return value from the call to fork or thread to see if you are the parent or child. We will assume this code listing and control flow graph are always referring to the forked thread/child process because the only thing the parent thread is doing is monitoring for the program to be signaled to exit which is not part of this discussion.
- For simplicity we only handle exiting the process at the beginning of the main loop and we assume some variable has been populated by a value telling us to exit. Technically, the cleanup phase of a modern (forked or threaded) server is triggered by an interrupt signal and can happen anywhere in the process. Different signals can cause the process to gracefully shutdown by finishing serving existing requests but not accepting any more requests, or may cause an immediate shutdown without returning any further response to the client.
- When running in `dynamic` mode, the program searches for an executable script in every component of the request path, from deepest back to the root.

In the Control Flow Graph in figure D.1, nodes where the exit path is determined

by a configuration value are shaded with vertical lines. Nodes where the exit path is determined by data flow are shaded with horizontal lines.

Figure D.1 Control Flow Graph for `maitre_d`



D.2 Examples from Section 5.4.2

Listing D.1 Example configuration file for dependent knobs

```
<!-- Somewhere else: Define CONFIG{"ENABLE_SSL"} -->
<IfDefined CONFIG="enable_ssl">
    LoadModule "mod_ssl.so"
    SSLEngine On
    SSLCertificateFile /etc/ssl/certificate.pem

    <AllowedUserList>myuser,anotheruser,plony</AllowedUserList>
    Require allowed-user
</IfDefined>
<IfDefined ! CONFIG="enable_ssl">
    SSL Engine Off

    Require all granted
</IfDefined>
```

Listing D.2 Example configuration file where knob dependency is not obvious (*"messy"*)

```
<!-- Somewhere else: Define CONFIG{"ENABLE_SSL"} -->
<IfDefined CONFIG="enable_ssl">
    LoadModule "mod_ssl.so"
</IfDefined>

<IfModule enabled="ssl">
    SSLEngine On
    SSLCertificateFile /etc/ssl/certificate.pem

    <AllowedUserList>myuser,anotheruser,plony</AllowedUserList>
    Require allowed-user
</IfModule>
<IfModule ! enabled="ssl">
    SSL Engine Off

    Require all granted
</IfModule>
```

Listing D.3 Simplified excerpt of configuration with obviously dependent knobs

```
// S1
if ($CONFIG['ssl_server']) {

    // S2
    load_module("ssl_provider.so")
    $socket = create_ssl_socket($CONFIG['other_ssl_configuration']);

    if (user_in_allow_list(
            get_user_with_basic_auth($CONFIG['allowed_user_list'])
    ) {
        // S3
        $action = allow_access();
    } else {
        // S4
        $action = deny_access();
    }
} else {
    // S5
    $socket = create_plain_socket();

    // Don't require authentication because password would be sent in clear text
    $action = allow_access();
}

// W
send_response($socket, $action);
```

Listing D.4 Simplified excerpt of configuration with dependent knobs that are not obvious (*"messy"*)

```
// S1
if ($CONFIG['ssl_server']) {
    // S1A
    load_module("ssl_provider.so")
}

// S1B
if (module_is_loaded("ssl_provider.so")) {
    // S2
    $socket = create_ssl_socket($CONFIG['other_ssl_configuration']);

    if (user_in_allow_list(
            get_user_with_basic_auth($CONFIG['allowed_user_list'])
    ) {
        // S3
        $action = allow_access();
    } else {
        // S4
        $action = deny_access();
    }
} else {
    // S5
    $socket = create_plain_socket();

    // Don't require authentication because password would be sent in clear text
    $action = allow_access();
}
```

Listing D.5 Simplified excerpt of configuration knobs with side effects that make them affect other knobs

```
// Example configuration
$CONFIG = [
    'mode' => 'fork|thread',  'number_of_workers' => 10,
    'count_to' => 100,        'log_every' => 10,
    'output_file' => 'important.log',
];

// S1
$count = 0;
match ($CONFIG['mode']) {
    case 'fork' => { // S2
        fork_processes($CONFIG['number_of_workers']);  },
    case 'thread' => { //S3
        join_threads($CONFIG['number_of_workers']);    },
}

// S4
// This program behaves differently in threaded execution or forked execution.
// Forked Execution: The loop may execute up to ('count_to' * 'number_of_workers')
//     times. Multiple processes will not share a mutex, potentially damaging the
//     output file or crashing the program.
// Threaded Execution: A race condition makes it not possible to determine the number
//     of times the loop will be executed, but at least the mutex will be respected.
while ($count < $CONFIG['count_to']) {

    // S5
    if (is_prime($count)) { // S6
        do_action_1_which_takes_a_long_time();
    }
    if (some other conditions and tasks) ... // Not graphed
    // S7
    if ($count % $CONFIG['log_every'] === 0) { // S8
        // A file can only be written by one thread or process at a time
        get_mutex();
        write_log($CONFIG['output_file']);
        release_mutex();
    }
    // S9 - $count is incremented non-atomically
    $count = $count + 1;
}
```

Bibliography

[1] 2014. Charis SIL — An extended Latin font. Retrieved October 20, 2024 from https://software.sil.org/charis/

[2] Zotero | Your personal research assistant. Retrieved October 20, 2024 from https://www.zotero.org/

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE symposium on security and privacy (SP)*, 2016. 289–305. https://doi.org/10.1109/SP.2016.25

[4] Douglas Adams and Douglas Adams. 2009. *Mostly harmless* (New ed. ed.). Pan Books, London.

[5] Neal Agarwal. The Password Game. Retrieved from https://neal.fun/password-game/

[6] William (Bill) Allcock, Evan Felix, Mike Lowe, Randal Rheinheimer, and Joshi Fullop. 2011. Challenges of HPC monitoring. In *State of the Practice Reports*, November 12, 2011. ACM, Seattle Washington, 1–6. https://doi.org/10.1145/2063348.2063378

[7] Frances E. Allen. 1970. Control flow analysis. *SIGPLAN Not.* 5, 7 (July 1970), 1–19. https://doi.org/10.1145/390013.808479

[8] Ignacio Alvarez, Aqueasha Martin, Jerone Dunbar, Joachim Taiber, Dale-Marie Wilson, and Juan E. Gilbert. 2010. Voice interfaced vehicle user help. In *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, November 11, 2010. ACM, Pittsburgh Pennsylvania, 42–49. https://doi.org/10.1145/1969773.1969782

[9] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, 1967. ACM Press, Atlantic City, New Jersey, 483. https://doi.org/10.1145/1465482.1465560

[10] Ricardo Baeza-Yates and Diego Saez-Trumper. 2015. Wisdom of the Crowd or Wisdom of a Few?: An Analysis of Users' Content Generation. In *Proceedings of the 26th ACM Conference on Hypertext & Social Media - HT '15*, 2015. ACM Press, Guzelyurt, Northern Cyprus, 69–74. https://doi.org/10.1145/2700171.2791056

[11] Huizhi Bai, Zhizi Liu, Ziqi Fu, Zihao Liu, Huihui Zhang, Honghai Zhu, and Liang Zhang. 2023. Objective Metrics for Assessing Visual Complexity of Vehicle Dashboards: A Machine-Learning Based Study. In *HCI in Mobility, Transport, and Automotive Systems*, Heidi Krömker (ed.). Springer Nature Switzerland, Cham, 103–113. https://doi.org/10.1007/978-3-031-35908-8_8

[12] David H. Bailey and Allan Snavely. 2005. Performance Modeling: Understanding the Past and Predicting the Future. In *Euro-Par 2005 Parallel Processing*, José C. Cunha and Pedro D. Medeiros (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 185–195. https://doi.org/10.1007/11549468_23

[13] Hitesh Ballani, Paul Francis, and Xinyang Zhang. 2007. A study of prefix hijacking and interception in the internet. *SIGCOMM Comput. Commun. Rev.* 37, 4 (October 2007), 265. https://doi.org/10.1145/1282427.1282411

[14] Petros C. Benias, Rebecca G. Wells, Bridget Sackey-Aboagye, Heather Klavan, Jason Reidy, Darren Buonocore, Markus Miranda, Susan Kornacki, Michael Wayne, David L. Carr-Locke, and Neil D. Theise. 2018. Structure and Distribution of an Unrecognized Interstitium in Human Tissues. *Sci Rep* 8, 1 (March 2018), 4947. https://doi.org/10.1038/s41598-018-23062-6

[15] Matthew Bernhard, Jonathan Sharman, Claudia Ziegler Acemyan, Philip Kortum, Dan S. Wallach, and J. Alex Halderman. 2019. On the Usability of HTTPS Deployment. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, May 02, 2019. ACM, Glasgow Scotland Uk, 1–10. https://doi.org/10.1145/3290605.3300540

[16] Antonia Bertolino. 2007. Software Testing Research: Achievements, Challenges, Dreams. In *Future of Software Engineering (FOSE '07)*, May 2007. IEEE, Minneapolis, MN, USA, 85–103. https://doi.org/10.1109/FOSE.2007.25

[17] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. 2022. A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, November 07, 2022. ACM, Los Angeles CA USA, 365–379. https://doi.org/10.1145/3548606.3559360

[18] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. 2019. Geneva: Evolving Censorship Evasion Strategies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, November 06, 2019. ACM, London United Kingdom, 2199–2214. https://doi.org/10.1145/3319535.3363189

[19] André B. Bondi. 2016. Incorporating Software Performance Engineering Methods and Practices into the Software Development Life Cycle. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, March 12, 2016. ACM, Delft The Netherlands, 327–330. https://doi.org/10.1145/2851553.2858668

[20] Charles Border. 2020. Development of a configuration management course for computing operations students. *J. Comput. Sci. Coll.* 36, 3 (October 2020), 89–101.

[21] S. Borkar. 2005. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro* 25, 6 (November 2005), 10–16. https://doi.org/10.1109/MM.2005.110

[22] Bryan Buck and Jeffrey K. Hollingsworth. 2000. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications* 14, 4 (November 2000), 317–329. https://doi.org/10.1177/109434200001400404

[23] Peter Calvocoressi. 1977. The secret of Enigma. *Listener (1970)* 97, 2492 (1977), 70.

[24] Taejoong Chung, John Rula, Nick Sullivan, Emile Aben, Tim Bruijnzeels, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Roland van Rijswijk-Deij. 2019. RPKI is Coming of Age: A Longitudinal Study of RPKI Deployment and Invalid Route Origins. In *Proceedings of the Internet Measurement Conference on - IMC '19*, 2019. ACM Press, Amsterdam, Netherlands, 406–419. https://doi.org/10.1145/3355369.3355596

[25] Elizabeth F. Churchill. 2015. Patchwork living, rubber duck debugging, and the chaos monkey. *interactions* 22, 3 (April 2015), 22–23. https://doi.org/10.1145/2752126

[26] Michael Clark and Kent Seamons. 2022. Passwords and Cryptwords: The Final Limits on Lengths. In *Proceedings of the 2022 New Security Paradigms Workshop*, October 24, 2022. ACM, North Conway NH USA, 75–89. https://doi.org/10.1145/3584318.3584324

[27] Dan O'Day. UAC Virtualization. Retrieved from https://github.com/danzek/annotationis/blob/edaa2357d24f02242cdd57e52956aed151c643c2/Operating%20Systems/Windows/UACVirtualization.md

[28] Susan Dart. 1991. Concepts in configuration management systems. In *Proceedings of the 3rd international workshop on Software configuration management*, May 1991. ACM, Trondheim Norway, 1–18. https://doi.org/10.1145/111062.111063

[29] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX security symposium - volume 13* (*SSYM'04*), 2004. USENIX Association, USA, 21. https://doi.org/10.5555/1251375.1251396

[30] Ryan Donovan. 2024. In Rust we trust? White House Office urges memory safety. Retrieved from https://stackoverflow.blog/2024/03/04/in-rust-we-trust-white-house-office-urges-memory-safety/

[31] Dave Dugal and Dale Rich. Announcing CVSS v4.0. In *35th Annual FIRST Conference*. Retrieved December 31, 2023 from https://www.first.org/cvss/v4-0/cvss-v40-presentation.pdf

[32] Ethar Elsaka, Walaa Eldin Moustafa, Bao Nguyen, and Atif Memon. 2010. Using Methods & Measures from Network Analysis for GUI Testing. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, April 2010. IEEE, Paris, France, 240–246. https://doi.org/10.1109/ICSTW.2010.61

[33] Ralph Waldo Emerson, Waldo Emerson Forbes, and Edward Waldo Emerson. 1909. *Journals of Ralph Waldo Emerson, with annotations*. Houghton Mifflin, Boston. Retrieved from https://catalog.hathitrust.org/Record/000557665

[34] Sascha Fahl, Marian Harbach, Yasemin Acar, and Matthew Smith. 2013. On the ecological validity of a password study. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, July 24, 2013. ACM, Newcastle United Kingdom, 1–13. https://doi.org/10.1145/2501604.2501617

[35] Felix Fischer, Yannick Stachelscheid, and Jens Grossklags. 2021. The Effect of Google Search on Software Security: Unobtrusive Security Interventions via Content Re-ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, November 12, 2021. ACM, Virtual Event Republic of Korea, 3070–3084. https://doi.org/10.1145/3460120.3484763

[36] Lance Fortnow. 2009. The status of the P versus NP problem. *Commun. ACM* 52, 9 (September 2009), 78–86. https://doi.org/10.1145/1562164.1562186

[37] Aya Furuta. One Thing Is Certain: Heisenberg's Uncertainty Principle Is Not Dead. *Scientific American*. Retrieved October 15, 2024 from https://www.scientificamerican.com/article/heisenbergs-uncertainty-principle-is-not-dead/

[38] Francis Galton. 1907. Vox Populi. *Nature* 75, 1949 (March 1907), 450–451. https://doi.org/10.1038/075450a0

[39] Ruoyu Gao and Zhen Ming Jiang. 2017. An Exploratory Study on Assessing the Impact of Environment Variations on the Results of Load Tests. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017. IEEE, Buenos Aires, Argentina, 379–390. https://doi.org/10.1109/MSR.2017.22

[40] Anuj Gautam, Shan Lalani, and Scott Ruoti. 2022. Improving password generation through the design of a password composition policy description language. In *Proceedings of the eighteenth USENIX conference on usable privacy and security* (*SOUPS'22*), 2022. USENIX Association, USA.

[41] Tom Gibbs. 2023. AI for a Scientific Computing Revolution. Retrieved April 19, 2024 from https://developer.nvidia.com/blog/ai-for-a-scientific-computing-revolution/

[42] Maximilian Golla and Markus Dürmuth. 2018. On the Accuracy of Password Strength Meters. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, October 15, 2018. ACM, Toronto Canada, 1567–1582. https://doi.org/10.1145/3243734.3243769

[43] Paul A Grassi, James L Fenton, Elaine M Newton, Ray A Perlner, Andrew R Regenscheid, William E Burr, Justin P Richer, Naomi B Lefkovitz, Jamie M Danker, Yee-Yin Choong, Kristen K Greene, and Mary F Theofanos. 2017. *Digital identity guidelines: Authentication and lifecycle management*. National Institute of Standards and Technology, Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-63b

[44] Miranda Green. 2014. 'Cyber' is everywhere, but what does the prefix mean and where did it come from? *WMAR 2 News / ABC Baltimore: DecodeDC*. Retrieved from https://www.wmar2news.com/decodedc/cyber-is-everywhere-but-what-does-the-prefix-mean-and-where-did-it-come-from

[45] Tim Harford. 2019. Lessons from the wreck of the Torrey Canyon.

[46] David Heinemeier Hansson. 2022. Why we're leaving the cloud. Retrieved from https://world.hey.com/dhh/why-we-re-leaving-the-cloud-654b47e0

[47] Jack Hope. 1996. A Better Mousetrap. *American Heritage 47*. Retrieved June 19, 2024 from https://www.americanheritage.com/better-mousetrap

[48] Hynek Schlawack. Hardening Your Web Server's SSL Ciphers. Retrieved December 18, 2019 from https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/

[49] Jarl Jansen, Nathan Cassee, and Alexander Serebrenik. 2024. Sentiment of Technical Debt Security Questions on Stack Overflow: A Replication Study. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, March 12, 2024. IEEE, Rovaniemi, Finland, 821–829. https://doi.org/10.1109/SANER60148.2024.00089

[50] Pontus Johnson, Robert Lagerstrom, Mathias Ekstedt, and Ulrik Franke. 2018. Can the Common Vulnerability Scoring System be Trusted? A Bayesian Analysis. *IEEE Trans. Dependable and Secure Comput.* 15, 6 (November 2018), 1002–1015. https://doi.org/10.1109/TDSC.2016.2644614

[51] Yongnam Jung, Cheng Chen, Eunchae Jang, and S. Shyam Sundar. 2024. Do We Trust ChatGPT as much as Google Search and Wikipedia? In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, May 11, 2024. ACM, Honolulu HI USA, 1–9. https://doi.org/10.1145/3613905.3650862

[52] Shahin Kamali and Alejandro López-Ortiz. 2015. Efficient Online Strategies for Renting Servers in the Cloud. In *SOFSEM 2015: Theory and Practice of Computer Science*, Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater and Roger Wattenhofer (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 277–288. https://doi.org/10.1007/978-3-662-46078-8_23

[53] Poul-Henning Kamp. 2013. Center Wheel for Success: Not invented here syndrome is not unique to the IT world. *Queue* 11, 12 (December 2013), 10–13. https://doi.org/10.1145/2559899.2559901

[54] Albert B. Kao, Andrew M. Berdahl, Andrew T. Hartnett, Matthew J. Lutz, Joseph B. Bak-Coleman, Christos C. Ioannou, Xingli Giam, and Iain D. Couzin. 2018. Counteracting estimation bias and social influence to improve the wisdom of crowds. *J. R. Soc. Interface.* 15, 141 (April 2018), 20180130. https://doi.org/10.1098/rsif.2018.0130

[55] Ruth Kassinger. 2002. *Build a better mousetrap: Make classic inventions, discover your problem-solving genius, and take the inventor's challenge*. John Wiley & Sons, Hoboken.

[56] Moshe Matanya Katz. 2023. Software-Defined Software. University of Maryland, College Park, MD. https://doi.org/10.13016/DSPACE/BUEH-LTGQ

[57] Nick Knight and Jack Poulson. 2013. Scientific computing. *XRDS* 19, 3 (March 2013), 7–7. https://doi.org/10.1145/2425676.2425679

[58] Donald Kossmann, Tim Kraska, and Simon Loesing. 2010. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, June 06, 2010. ACM, Indianapolis Indiana USA, 579–590. https://doi.org/10.1145/1807167.1807231

[59] Paul Krill. 2024. C++ creator rebuts White House warning. *InfoWorld*. Retrieved from https://www.infoworld.com/article/2336463/c-plus-plus-creator-rebuts-white-house-warning.html

[60] Chet Langin. 2017. We Have an HPC System: Now What? In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, July 09, 2017. ACM, New Orleans LA USA, 1–8. https://doi.org/10.1145/3093338.3093341

[61] Kevin Lee, Sten Sjöberg, and Arvind Narayanan. 2022. Password policies of most top websites fail to follow best practices. In *Proceedings of the eighteenth USENIX conference on usable privacy and security* (*SOUPS'22*), 2022. USENIX Association, USA.

[62] N. G. Leveson and C. S. Turner. 1993. An investigation of the Therac-25 accidents. *Computer* 26, 7 (July 1993), 18–41. https://doi.org/10.1109/MC.1993.274940

[63] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th International Conference on Software Engineering*, May 27, 2018. ACM, Gothenburg Sweden, 94–104. https://doi.org/10.1145/3180155.3180195

[64] Wei-Chen Lin and Jens Domke. 2024. Benchmarking in the Datacenter (BID): Expanding to the Cloud. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, May 07, 2024. ACM, London United Kingdom, 94–94. https://doi.org/10.1145/3629527.3651434

[65] Yuri Lin, Jean-Baptiste Michel, Erez Aiden Lieberman, Jon Orwant, Will Brockman, and Slav Petrov. 2012. Syntactic Annotations for the Google Books NGram Corpus. In *Proceedings of the ACL 2012 system demonstrations*, July 2012. Association for Computational Linguistics, Jeju Island, Korea, 169–174. Retrieved from https://aclanthology.org/P12-3029

[66] Teresa Liu. 2000. Configuration management for a distributed and collaborative software development environment. M.Eng. Massachusetts Institute of Technology. Dept. of Civil and Environmental Engineering. Retrieved from https://dspace.mit.edu/handle/1721.1/9019

[67] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D. Le, and David Lo. 2024. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ACM Trans. Softw. Eng. Methodol.* 33, 5 (June 2024), 1–26. https://doi.org/10.1145/3643674

[68] Elizabeth Lopatto. 2024. Google still recommends glue for your pizza. Retrieved from https://www.theverge.com/2024/6/11/24176490/mm-delicious-glue

[69] Roman Lytvynenko. 2024. Automated API documentation generation for Laravel with static code analysis using Scramble. Retrieved from https://laravel-news.com/automated-api-documentation

[70] John MacFarlane, Albert Krewinkel, and Jesse Rosenthal. 2024. Pandoc. Retrieved October 20, 2024 from https://github.com/jgm/pandoc

[71] Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. 2014. Fleet: Defending SDNs from malicious administrators. In *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, 2014. ACM Press, Chicago, Illinois, USA, 103–108. https://doi.org/10.1145/2620728.2620750

[72] Kevin Menear, Ambarish Nag, Jordan Perr-Sauer, Monte Lunacek, Kristi Potter, and Dmitry Duplyakin. 2023. Mastering HPC Runtime Prediction: From Observing Patterns to a Methodological Approach. In *Practice and Experience in Advanced Research Computing*, July 23, 2023. ACM, Portland OR USA, 75–85. https://doi.org/10.1145/3569951.3593598

[73] Harshitha Menon, Abhinav Bhatele, and Todd Gamblin. 2020. Auto-tuning Parameter Choices in HPC Applications using Bayesian Optimization. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020. IEEE, New Orleans, LA, USA, 831–840. https://doi.org/10.1109/IPDPS47924.2020.00090

[74] Barton P. Miller, Lars Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. *Commun. ACM* 33, 12 (December 1990), 32–44. https://doi.org/10.1145/96267.96279

[75] Carol-anne E. Moulton, Glenn Regehr, Maria Mylopoulos, and Helen M. MacRae. 2007. Slowing Down When You Should: A New Model of Expert Judgment: *Academic Medicine* 82, (October 2007), S109–S116. https://doi.org/10.1097/ACM.0b013e3181405a76

[76] Randall Munroe. 2006. *Xkcd: Pointers*. Retrieved from https://xkcd.com/138/

[77] Netcraft. 2024. *February 2024 Web Server Survey*. Retrieved from https://www.netcraft.com/blog/february-2024-web-server-survey/

[78] Peter G. Neumann. 2011. Risks to the public. *SIGSOFT Softw. Eng. Notes* 36, 2 (March 2011), 19–27. https://doi.org/10.1145/1943371.1943376

[79] George V. Neville-Neil. 2010. Literate coding. *Commun. ACM* 53, 12 (December 2010), 37–38. https://doi.org/10.1145/1859204.1859219

[80] Ruben Ortega. 2010. Why Do Software Developers Tolerate "Crunch Time"? Retrieved from https://cacm.acm.org/blogcacm/why-do-software-developers-tolerate-crunch-time/

[81] George Orwell. 1996. *Animal farm: A fairy story* (complete text ed., 89. impr ed.). Longman, Harlow.

[82] David L. Parnas. 2010. Risks of undisciplined development. *Commun. ACM* 53, 10 (October 2010), 25–27. https://doi.org/10.1145/1831407.1831419

[83] Nilay Patel. 2016. The recalled Jeep shifter is just bad user interface design. *The Verge*. Retrieved from https://www.theverge.com/2016/6/27/12043898/chrysler-jeep-dodge-electronic-gear-shift-recall-design-flaw-video

[84] Maya Posch. 2024. The White House Memory Safety Appeal Is A Security Red Herring. Retrieved from https://hackaday.com/2024/02/29/the-white-house-memory-safety-appeal-is-a-security-red-herring/

[85] Kevin Purdy. 2023. Your password must include puzzles — The Password Game will make you want to break your keyboard in the best way. *Ars Technica*. Retrieved from https://arstechnica.com/gaming/2023/06/the-password-game-will-make-you-want-to-break-your-keyboard-in-the-best-way/

[86] Ariel Rabkin and Randy Katz. 2011. Static extraction of program configuration options. In *Proceedings of the 33rd International Conference on Software Engineering*, May 21, 2011. ACM, Waikiki, Honolulu HI USA, 131–140. https://doi.org/10.1145/1985793.1985812

[87] Raghu. 2024. Raghur/mermaid-filter. Retrieved October 20, 2024 from https://github.com/raghur/mermaid-filter

[88] Arthur Railton. 1948. ”CHESSIE” has that new look. *Popular Mechanics 89*, 107–111, 252, 256. Retrieved from https://books.google.com?id=5dgDAAAAMBAJ

[89] Brian Randell. 1979. An Annotated Bibliography on the Origins of Digital Computers. *IEEE Annals Hist. Comput.* 1, 2 (April 1979), 101–207. https://doi.org/10.1109/MAHC.1979.10016

[90] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. 2024. *TLS encrypted client hello*. Internet Engineering Task Force / Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/draft-ietf-tls-esni/22/

[91] Rachael Rettner. 2018. Meet Your Interstitium, a Newfound ”Organ”. *Scientific American*. Retrieved from https://www.scientificamerican.com/article/meet-your-interstitium-a-newfound-organ/

[92] David P. Rodgers. 1985. Improvements in multiprocessor system design. *SIGARCH Comput. Archit. News* 13, 3 (June 1985), 225–231. https://doi.org/10.1145/327070.327215

[93] Gonzalo Pedro Rodrigo Álvarez, Per-Olov Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. 2015. HPC System Lifetime Story: Workload Characterization and Evolutionary Analyses on NERSC Systems. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, June 15, 2015. ACM, Portland Oregon USA, 57–60. https://doi.org/10.1145/2749246.2749270

[94] Stephanie Ricker Schulte. 2008. "The WarGames Scenario": Regulating Teenagers and Teenaged Technology (1980—1984). *Television & New Media* 9, 6 (November 2008), 487–513. https://doi.org/10.1177/1527476408323345

[95] Neil Selwyn. 2009. The digital native – myth and reality. *Aslib Proceedings* 61, 4 (July 2009), 364–379. https://doi.org/10.1108/00012530910973776

[96] Rachee Singh, Rishab Nithyanand, Sadia Afroz, Paul Pearce, Michael Carl Tschantz, Phillipa Gill, and Vern Paxson. 2017. Characterizing the nature and dynamics of tor exit blocking. In *26th USENIX security symposium (USENIX security 17)*, August 2017. USENIX Association, Vancouver, BC, 325–341. Retrieved from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/singh

[97] Joel Spolsky. 2000. Choices. Retrieved from https://www.joelonsoftware.com/2000/04/12/choices/

[98] Dominic Steinhöfel and Andreas Zeller. 2024. Language-Based Software Testing. *Commun. ACM* 67, 4 (April 2024), 80–84. https://doi.org/10.1145/3631520

[99] Sheryl Gay Stolberg and Michael D. Shear. 2013. Inside the Race to Rescue a Health Care Site, and Obama. *The New York Times: U.S.* Retrieved October 15, 2024 from https://www.nytimes.com/2013/12/01/us/politics/inside-the-race-to-rescue-a-health-site-and-obama.html

[100] Eugene Styer. 1997. PC note. *SIGICE Bull.* 22, 4 (April 1997), 25–32. https://doi.org/10.1145/270548.270552

[101] James Surowiecki. 2004. *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations.* Doubleday & Co, New York, NY, US.

[102] Minaoar Hossain Tanzil, Junaed Younus Khan, and Gias Uddin. 2024. ChatGPT Incorrectness Detection in Software Reviews. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, April 12, 2024. ACM, Lisbon Portugal, 1–12. https://doi.org/10.1145/3597503.3639194

[103] Nikos Vasilakis, Konstantinos Kallas, Konstantinos Mamouras, Achilles Benetopoulos, and Lazar Cvetković. 2021. PaSh: Light-touch data-parallel shell processing. In *Proceedings of the Sixteenth European Conference on Computer Systems*, April 21, 2021. ACM, Online Event United Kingdom, 49–66. https://doi.org/10.1145/3447786.3456228

[104] Werner Vogels. 2009. Eventually consistent. *Commun. ACM* 52, 1 (January 2009), 40–44. https://doi.org/10.1145/1435417.1435432

[105] Kenneth F. Wallis. 2014. Revisiting Francis Galton's Forecasting Competition. *Statistical Science* 29, 3 (2014), 420–424. Retrieved September 10, 2024 from http://www.jstor.org/stable/43288519

[106] Juite Wang and Yung-I Lin. 2003. A fuzzy multicriteria group decision making approach to select configuration items for software development. *Fuzzy Sets and Systems* 134, 3 (March 2003), 343–363. https://doi.org/10.1016/S0165-0114(02)00283-X

[107] Name Withheld. 2024. Interview with Mission System Execution Lead.

[108] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, 2015. ACM Press, Bergamo, Italy, 307–319. https://doi.org/10.1145/2786805.2786852

[109] ZDNET. 2024. Linus Torvalds takes on evil developers, hardware errors and 'hilarious' AI hype. Retrieved April 19, 2024 from https://www.zdnet.com/article/linus-torvalds-takes-on-evil-developers-hardware-errors-and-hilarious-ai-hype/

[110] Marvin V. Zelkowitz. 2012. What have we learned about software engineering? *Commun. ACM* 55, 2 (February 2012), 38–39. https://doi.org/10.1145/2076450.2076463

[111] H. Zemanek. 1976. Computer prehistory and history in central Europe. In *Proceedings of the June 7-10, 1976, national computer conference and exposition on - AFIPS '76*, 1976. ACM Press, New York, New York, 15. https://doi.org/10.1145/1499799.1499803

[112] Guido Zuccon, Bevan Koopman, and Razia Shaik. 2023. ChatGPT Hallucinates when Attributing Answers. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, November 26, 2023. ACM, Beijing China, 46–51. https://doi.org/10.1145/3624918.3625329

[113] 2012. *TEDxLiverpool - Tom Scott - A Mobile Future*. Retrieved September 10, 2024 from https://www.youtube.com/watch?v=d68mcH2veCU

[114] 2023. *Common Vulnerability Scoring System version 4.0: Specification Document*. Retrieved December 13, 2023 from https://www.first.org/cvss/v4.0/specification-document

[115] Puppet 7 Server - System requirements. Retrieved from https://www.puppet.com/docs/puppet/7/system_requirements#system_requirements

[116] Apache Performance Tuning. Retrieved from https://httpd.apache.org/docs/2.4/misc/perf-tuning.html

[117] CVSS v3.1 Specification Document - Qualitative Scale. Retrieved from https://www.first.org/cvss/v3.1/specification-document#Qualitative-Severity-Rating-Scale

[118] UPSS Calculator v0.9. Retrieved from https://www.cs.umd.edu/~yakatz/research/upss/calc-v0.9.xlsx

[119] Apache HTTPD Source. Retrieved February 29, 2024 from https://github.com/apache/httpd/tree/2.4.x

[120] Request Processing in the Apache HTTP Server 2.x. Retrieved from https://httpd.apache.org/docs/2.4/developer/request.html

[121] SmartAssembly. Retrieved from https://www.red-gate.com/products/smartassembly/

[122] SourceGuardian. Retrieved from https://www.sourceguardian.com/

[123] *Solid State Logic SL 9000J Seventy Two channel mixing console from Wikimedia Commons - Creative Commons Attribution 2.0 Generic license*. Retrieved from https://commons.wikimedia.org/wiki/File:SSL_SL9000J_(72ch)_@_The_Cutting_Room_Recording_Studios,_NYC.jpg

[124] Obituary of Dr. Kerr Lachlan White. Retrieved from https://www.legacy.com/us/obituaries/washingtonpost/name/kerr-white-obituary?id=6035868

[125] Google Ngram Viewer: cybersecurity. Retrieved August 1, 2024 from https://books.google.com/ngrams/graph?content=cybersecurity&year_start=1970&year_end=2022&corpus=en&smoothing=0&case_insensitive=false

[126] HAProxy Source. Retrieved February 29, 2024 from https://github.com/haproxy/haproxy

[127] *IEEE Standard for Configuration Management in Systems and Software Engineering*. IEEE. https://doi.org/10.1109/IEEESTD.2012.6170935

[128] *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. IEEE. https://doi.org/10.1109/IEEESTD.1991.106963

[129] Apogee Games. 1992. Word Rescue. Retrieved from https://en.wikipedia.org/wiki/Word_Rescue