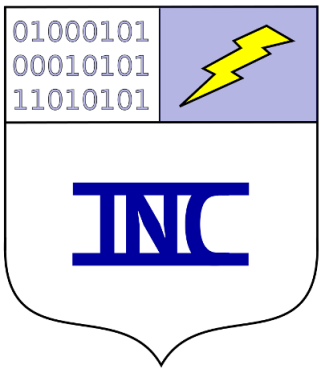
Hacking as an Anathema to Computer Science

B. Kirkpatrick *

June 26, 2016

Intrepid Net Computing



www.intrepidnetcomputing.com

*bbkirk@intrepidnetcomputing.com

Document Revision History

December 27, 2015 Published original on www.intrepidnetcomputing.com

June 26, 2016 Created title page

Abstract

Hacking is an anathema to the field of computer science. Similar to illegal human medical experiments in the field of medicine, hackers and hacking incurs the wrath of honest, hard-working, ethical, and professional computer scientists. Hacking should be considered as a metaphorical equivalent to malpractice in medicine. Hacking is almost as difficult to detect as medical malpractice, and ethical and educational solutions are appropriate.

This document discusses the education background of hacking, open source computing, and how to detect hacking. We believe that computer scientists should take a professional oath, similar to the medical professional oath, to be ethical and not hack.

1 Introduction

Hacking is an anathema to the field of computer science. Computers provide basic economic infrastructure to world economies. While it would be nice to have a fool-proof method of preventing and detecting hacking, the existence of digital computing likely means that hacking will be with us for as long as we have digital computers. Our only reasonable method of detecting hacking is sleuthing and law enforcement.

Most hacks exploit the inherent digital nature of computing. In a digital computer, many real numbers cannot be exactly represented in binary. Also any allocated buffer must have finite length. Both of these features are inherent in digital computing and both can be exploited to drive software behavior that was unintended by the software programmers.

Professional computer scientists that raised professionally through the ACM approved bachelors degree courses know the theory and practice required to understand conceptually why hacking is an anathema. This document will explore the incredible theoretical difficulty of creating working computer programs, including operating systems, and the opportunities for unethical behavior. We will also discuss how to detect hacking.

This document will also introduce ethical computing as a solution to hacking, and will call on the field of computer science to socially expel hackers through the use of a professional oath, through the traditional use of limited access for irresponsible individuals, and through the use of law enforcement.

2 Background

The theory that discusses hacking is as old as its practical applications. Alan Turing was the first to write a proof relevant to computer viruses when he demonstrated that there were self-replicating Turing machines. Another crucial aspect of the relevant theory, program verification, proven to be undecidable by Alan Turing, has essentially motivated the raise of open source computing. We will explore how both self-replicating Turing machines and open source computing help create modern computing and how they are exploited by hackers. We will also briefly explore how to detect hacking.

2.1 Self-Replicating Turing Machines

There are only a few legal self-replicating Turing machines, and they necessarily involve the combination of an operating system and a compiler. The most successful operating systems in the world are the open source operating systems: BSD and Linux. And the most successful compiler family in the world are the GNU compilers. Hence, the most successful self-replicating Turing machines in the world, and the most legal, are some pair of an open source operating system and an open source compiler.

Any use of self-replicating Turing machines as computer viruses is illegal and highly unethical. Given that the field of computer science has so much open source code for new-comers to play with, it seems unlikely that anyone can make any positive case for hacking. While some features of computing can be learned from

hacking, learning proper computer science teaches students so much more. We advocate computer experts learning how to properly program, as it is much more difficult to program properly than it is to hack.

2.2 Open Source Computing

Tremendous effort has been put into open source computing over the years, since the late 1970's when the field collectively began deciding that open source was necessary. Open source software is necessary because program verification, the problem of using a computer algorithm to determine whether a computer operating system (OS) or a compiler works properly, is undecidable. This means that there is no algorithm which can verify that an OS or a compiler works properly. This is closely related to the halting problem, which is the first problem that Alan Turing proved to be undecidable after defining the existence of undecidable problems. Turing defined undecidable problems as those problems having no algorithmic solution.

After Turing's phenomenal proofs in the late 1950's it took the field an additional twenty years to realize that his work means we all have to collaborate on the same code. Hence, we have open source. Since writing an OS or a compiler is an impossible task, it will take all the world's computer scientists working on the same piece of code for an operating system even begin to work properly some of the time.

In the late 1970's at the University of California Berkeley, open source computing was born, as researchers embarked on the task of writing impossible programs, and the BSD operating system was born. This code is so important and so impossible, that the license agreement for modifying or using it is equally non-existent. The Berkeley systems group decided that impossible code should have an equally impossible license agreement. If the BSD operating system ever worked properly, it might be needed for who-knows-what-critical-future-endeavors such as saving the world. Who can write a license that allows saving the world while also outlawing destroying the world? Recall that the Berkeley professors had already seen the effects of nuclear physics up-close-and-personal as it was used both to destroy and save the world. The BSD operating system could easily be combined with physics or other terrible scientific advances, at which time, a license would be measly protection and only humanity can save itself.

The BSD operating system is believed to be at the kernel of most of the successful operating systems in the world. As the field progresses, we believe that there has been a collapse in the number of successful operating systems. At first there was a plethora of attempts to write operating systems. But, as predicted by Turing's hardness proof, those attempts have collapsed as programmers join forces to out-compete the competitor operating systems.

Because the BSD operating system is open source and has no license stipulations, many computer systems professionals believe that it has quietly been borrowed as the kernel for most existing successful operating systems. Algorithmic theory is just now progressing to the point where we will be able to prove that this borrowing has happened. Now that there are practical implementations of graph isomorphism and a claim for a polynomial-time algorithm for graph isomorphism, we will be able to write a computer program to detect whether the BSD operating system was borrowed. While graph isomorphism allows us to detect that borrowing without modification, practical algorithms for sub-graph isomorphism allow us to detect borrowed fragments of the BSD operating system that appear in other kernels.

2.3 Debugging and Testing Software

As indicated by the undecidability of program verification, the sub-field of software engineering which focuses on debugging and testing software is also trying to do the impossible. It is extremely difficult to write code that works properly.

Many lay-people have remarked to me about the dynamics of software production and release. We have heard many business people grumble about having to constantly update their computers and constantly pay for new software. People, the reason you have to pay for new software and updates is because the old version has flaws that were discovered after it was released. Alan Turing proved the existence of such flaws when he mathematically described programming as an impossible task. Inherent in his proof is the dynamics of the field where bugs have to be fixed as they are discovered. Hackers simply take advantage of bugs or vulnerabilities that have not yet been fixed.

Inherent in Turing's proof is also the dynamic where overly ambitious programmers tackle programming jobs that are too big for their skill set and they fail. Anyone who has done a significant amount of programming has discovered the size of project which they can successfully complete in a relatively bug-free fashion. Teams of programmers who have worked together for a while have also discovered the size of codes that the team can handle. The over-ambitious-programmer dynamic is largely responsible for the number of failed start-ups in the field of computer science. Business people from other fields should be wary of trusting a critical aspect of their business with young software or with a young software-development company.

In general for each line of code involving a conditional statement (some form of an if-statement), the complexity of debugging increases exponentially. This means that wise programmers try to limit the number of conditional statements required in their programs and that wise programmers do everything they can to abstract groups of conditional statements into algorithms whose behavior that can be checked easily.

The algorithm's bible, Introduction to Algorithms by Corman, et al., is the classic text to refer to for algorithms whose behaviors are guaranteed by proof. These are a collection of algorithms that have been verified. This means that these are algorithms for which verification is decidable, and Turing's undecidability result does not apply to these algorithms. A successful new program or algorithm should involve modular components existing of provably correct algorithms. Computer scientists who pursue advanced degrees are taught this way of programming in graduate school as the entire field of academic computer science is geared towards proving the correctness of new algorithms.

It is incredibly easy to break code. It is much more difficult to make code work properly. Hacking has the goal of breaking existing codes. Hacking is much easier than doing proper computer science. It is very easy to find a way to break a program, and it is much more difficult to be a productive programmer that makes programs work properly.

2.4 Detecting Hackers

From the objective stand-point of an employer or university trying to detect hacking behaviors from individuals using their computers, hacking is almost impossible to distinguish from legitimate computer uses. On the other hand, from the stand-point of an expert computer scientist who is targeted by hacking, the activities of a hacker on your machine are incredibly easy to distinguish from normal computer behavior. These two statements seem counter-intuitive, but I will give statistical justification for both claims.

For an objective observer, hacking is difficult to distinguish from normal computer activities. This is because any hacker will be engaged in normal computing in addition to hacking, and their hacking activities will be buried in the big-data of all of their computer activities. Additionally, if an employer or university is trying to monitor many computers and many users, the hacking activities are buried in the big-data generated by many legitimate computer users. The objective observer would have to catch the hacker in-the-act, in order to identify the problem computer uses and to prove that the hacker had hacked. Catching the hacker in-the-act is really the only way to address the big-data challenges and to prove that an illegal act occurred.

This means that detecting hackers is the domain of law enforcement as sleuthing is the traditional means of detecting the criminal activities of a few individuals who are surrounded by mostly honest computer users. Electronic monitoring would only be effective after-the-fact to monitor a known hacker, just as parole boards electronically monitor known rapists, but are unable to use electronic means to detect rapists. Sleuthing must be used to detect hackers. Witnesses that observe the hacking in-the-act become the most critical resource for detecting hackers, just as with any other crime.

This means that hackers should be treated like any common criminal. Hackers can form a mafia or mob and can attempt to intimidate witnesses. Hackers can use hacking for witness intimidation, unless the whole field steps up to prevent on-going instances of hacking.

For an expert computer scientist who is targeted by a hacker, detecting hacking is notoriously easy. The only barriers that exist are barriers to reporting, not detection. A computer scientist who is very familiar with their computer hardware, their computer operating system, and is an expert programmer/debugger, will easily be able to detect the activities of an additional user on their machine or on their accounts. This

is especially true if the hackers attempt to harass a particular expert user. Even computer worms or viruses are astoundingly easy for expert users to detect.

There is a statistical formulation of hacking, which we will refer to as statistical forensics, that can be used to prove the existence of hacking on the targeted machines. The expert computer scientist who is targeted essentially uses their intuition to do their own statistical forensics without bother with computing any statistics. However, from the theoretical perspective, we can formalize the notion of statistical forensics in a way that can be statistically verified.

With *statistical forensics*, the goal is to demonstrate the existence of foreign machine code and unauthorized activities using statistics, rather than by isolating the exact mechanism of attack such as the foreign machine code. This can be done by proving that the entropy of the communications from the compromised machine is lower than that of the legitimate machine (i.e., the hacked machine communicates more information).

Statistical forensics assumes that hacking is a form of communication. Indeed, we view hacking as a form of bullying and as such, hacking does indeed communicate information from the bully to the victim, in particular the bully's malevolence. This means that hacking can be detected using information theory. An unhacked machine will communicate less information to the victim than a hacked machine. The following is a short algorithm for doing statistical forensics:

1. Identify the hacking events.
2. Compute the frequency of each event over time.
3. Compare the events of the hacked machine to the events of the unhacked machine.
4. Demonstrate that the hacked machine has higher entropy than the unhacked machine.

If the events are properly defined, the logs of the computer can be used to compare the computer in the hacked and unhacked states. While not all the hacking events will be logged, it is quite possible that enough events themselves or enough user responses to the events would be logged to statistically demonstrate that the computer is communicating more information after being hacking than before.

For example, in a recent hacking case that I know of, the hacking events included the following:

1. The OS querying the user for passwords three times, even when the password was correctly entered.
2. Destruction of encrypted partitions.
3. Files being hidden or deleted.
4. Files being transferred off a computer without permission.
5. Infection by a USB trojan.
6. Destruction of formerly installed programs.
7. Remote access via open wireless signals.
8. Use of the web camera to target a particular user with these hacking events.

Statistical forensics as conceptualized here is a form of sleuthing. It is something that must be done after an attack has been identified by a witness who can identify the hacking events. Statistical forensics is not something that can be remotely monitored to detect hacking as it occurs, as the relevant events would be buried in the big-data of normal computing activities. Indeed, use of the word "forensics" implies that the process is done after an attack occurs.

In all cases, we believe that hackers must be caught using forensics and using sleuthing. The world needs more cyber-sleuths and law-enforcement agencies need to keep pace with the new uses of technology for harassment, bullying, and victimization.

3 Ethical Computing

Ethical computing requires professional computer scientists to be professionally and personally removed from any activities that might be confused with hacking. These activities include, but are not limited to:

1. hacking,
2. fraud,
3. data theft,
4. penetration testing,
5. unauthorized access,
6. subversion of computer nannies,
7. modifying system logs after they are written,
8. subversion of any legitimate permissions system,
9. jail-breaking, or rooting, any machine without permission of the primary user,
10. jail-breaking, or rooting, any machine without notifying all the users, and
11. obstructing justice by failing to report hacking.

In addition to avoiding the above banned activities, ethical computing needs to take several extra steps. These extra steps include promoting equal access and reporting of illegal activities.

To fulfill the professional standards of the ACM, ethical computer scientists must support equal access to secure computing by all users. Ethical computer scientists are professionally obligated to be respectful and proactive users who take an anti-discrimination stand in terms of computing access.

Various reporting ethics have been established and some have been codified into laws. Scientists with medically-related federal grants are required to report data breaches of medical and personal information involving more than 100 individuals, and these reports are to be given directly to the NIH. Ethically speaking, universities and professors should report data breaches of student personal information to the students, just as companies are required to report breaches of financial information to customers. Similarly, for traveling computer scientists and incidents that occur around the world, reports should be made to the systems administrators involved and possibly to the governments of the countries involved, depending on the magnitude of the data breaches or hacking incidents. Certainly fraud should be reported according to standard crime reporting laws in the respective countries.

When, what, and how to report certainly involves some discussion of the magnitude of the problem. Minor issues that can be dealt with locally might not require reporting, unless significant personal data has been compromised. Particularly if the problems are an on-going technical challenge, they should be widely reported. Vulnerability reports are incredibly important, particularly before the vulnerabilities have been fixed, as advertising their severity produces work-arounds and fixes. An expert computer scientist should report on-going technical hacking challenges, even if there has not yet been time to do the forensics, identify the perpetrators, and discover the exact means of attack. Any delay by professionals in reporting any stage of an attack indicates a severe lack of ethics and is tantamount to obstructing response and obstructing justice.

4 Conclusions

We advocate the use of a professional oath for the field of computer science that supports ethical computing. We suggest the following oath:

I will be an ethical computer scientist, respectful of all users and laws, and will immediately report any hacking I observe to the systems administrator(s) of the affected systems at to the government of the country where those computers reside.

We suggest administering this oath to all individuals who graduate with an ACM approved bachelor's degree. This would be similar to the Hippocratic oath which is quite effective in helping medical doctors with similar ethical issues.

Biography

Dr. Kirkpatrick has a bachelor's in computer science from Montana State University-Bozeman, a master's and a Ph.D. in computer science from the University of California, Berkeley. Dr. Kirkpatrick is an expert in deterministic and statistical computer algorithms, and his main application area is the field of computational biology, in particular genetics. However, Dr. Kirkpatrick's algorithms expertise is general to the entire field of computer science, and he has recently had cause to specialize in algorithms for computer systems.