
DEGREE PROJECT IN TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2022*

# Diverse Double-Compilation For Bitcoin Core

Niklas Rosencrantz

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

## Author

Niklas Rosencrantz <nik@kth.se>
Electrical Engineering and Computer Science
KTH Royal Institute of Technology

## Place for Project

Stockholm, Sweden
Some place

## Examiner

Professor Benoit Baudry
Electrical Engineering and Computer Science
KTH Royal Institute of Technology

## Supervisor

Professor Martin Monperrus

Electrical Engineering and Computer Science

KTH Royal Institute of Technology

# Abstract

The problem of deceptive compilers introducing malicious code is relevant for computer security and a difficult class of attacks to analyse and discover. One solution for this is to use diverse double-compilation as a countermeasure. Yet it is hard to change the compilation pipeline of real software. I learn and write about the design of, and I implement and evaluate diverse double compilation for bitcoin-core. To attack and compromise Bitcoin core one might consider a few attack methods and exploits that can be tried and analysed, for example:

1. Stealing a PGP key from a maintainer and use that to sign new versions of Bitcoin-Core

2. Conducting a domain specific attack where the compiler inserts code that changes the recipient address in 1/10000 Bitcoin transactions.

3. A malevolent maintainer could upload malicious code, hide it. that would get caught by the verify signatures script

Write an abstract. Introduce the subject area for the project and describe the problems that are solved and described in the thesis. Present how the problems have been solved, methods used and present results for the project. Use probably one sentence for each chapter in the final report.

The presentation of the results should be the main part of the abstract. Use about ½ A4-page. English abstract

## Keywords

Template, Thesis, Keywords ...

# Abstract

Svenskt abstract Svensk version av abstract – samma titel på svenska som på engelska.

Problemet med vilseledande kompilatorer som introducerar skadlig kod är relevant för datorsäkerhet och en svår klass av attacker att analysera och upptäcka. En lösning för detta är att använda olika dubbelkompilering som motåtgärd. Ändå är det svårt att ändra kompileringspipelinen för riktig programvara. Jag lär mig och skriver om designen av, och jag implementerar och utvärderar olika dubbelkompilering för bitcoin-core. För att attackera och kompromissa med Bitcoins kärna kan man överväga några attackmetoder och exploateringar som kan prövas och analyseras, till exempel:

## Nyckelord

Kandidat examensarbete, ...

# Acknowledgements

Write a short acknowledgements. Don't forget to give some credit to the examiner and supervisor.

# Acronyms

**CPU** Central Processing Unit

**GCC** GNU Compiler Collection

# Contents

# Chapter 1

# Introduction

One problem with computer systems and networks is that the system will inherently trust the supply-chain. The problem of deceptive compilers introducing malicious code is relevant for computer security and a difficult class of attacks to analyse and discover. One solution for this is to use diverse double-compilation as a countermeasure. Yet it is hard to change the compilation pipeline of real software. I learn and write about the design of, and I implement and evaluate diverse double compilation for bitcoin-core. To attack and compromise Bitcoin core one might consider a few attack methods and exploits that can be tried and analysed, for example:

1. stealing a PGP key from a maintainer and use that to sign new versions of Bitcoin-Core
2. a domain specific attack where the compiler inserts code that changes the recipient address in 1/10000 Bitcoin transactions.
3. a malevolent maintainer could upload malicious code, hide it. that would get caught by the verify signatures script

Provide a general introduction to the area for the degree project. Use references!

Link things together with references. This is a reference to a section: 1.1.

## 1.1 Background

Present the background for the area. Give the context by explaining the parts that are needed to understand the degree project and thesis. (Still, keep in mind that this is an

introductory part, which does not require too detailed description).

Use references[1]

Detailed description of the area should be moved to Chapter 2, where detailed information about background is given together with related work.

This background presents background to writing a report in latex.

Example citation [6] or for two authors: [6, 8]

Look at sample table 1.1.1 for a table sample.

Table 1.1.1: Sample table. Make sure the column with adds up to 0.94 for a nice look.

| SAMPLE | TABLE |
|---|---|
| One | Stuff 1 |
| Two | Stuff 2 |
| Three | Stuff 3 |

Boxes can be used to organize content

Development environment for prototype

```
Operating systems
computer: Linux - kernel 4.18.5-arch1-1-ARCH
android phone: 8.1.0

Build tools
exp (build tool): version 55.0.4

...
```

## 1.2 Problem Statement

Present the problems found in the area. Preferable use and end this section with a question as a problem statement.

Use references Preferable, state the problem, to be solved, as a question. Do not use a question that can be answered with yes and/or no.

[1] You can also add footnotes if you want to clarify the content on the same page.

Use acronyms: The Central Processing Unit (CPU) is very nice. It is a CPU

The purpose of the degree project/thesis is the purpose of the written material, i.e., the thesis. The thesis presents the work / discusses / illustrates and so on.

It is not “The project is about” even though this can be included in the purpose. If so, state the purpose of the project after purpose of the thesis).

The goal means the goal of the degree project. Present following: the goal(s), deliverables and results of the project.

## 1.3 Methodology

Introduce, theoretically, the methodologies and methods that can be used in a project and, then, select and introduce the methodologies and methods that are used in the degree project. Must be described on the level that is enough to understand the contents of the thesis.

Use references!

Preferably, the philosophical assumptions, research methods, and research approaches are presented here. Write quantitative / qualitative, deductive / inductive / abductive. Start with theory about methods, choose the methods that are used in the thesis and apply.

Detailed description of these methodologies and methods should be presented in Chapter 3. In chapter 3, the focus could be research strategies, data collection, data analysis, and quality assurance.

Present the stakeholders for the degree project.

Explain the delimitations. These are all the things that could affect the study if they were examined and included in the degree project. Use references!

## 1.4 Contributors

In text, describe what is presented in Chapters 2 and forward. Exclude the first chapter and references as well as appendix.

# Chapter 2

# Background

The origins of this class of threats can be traced to Karger & Schell in their Multics Security Evaluation: Vulnerability Analysis [7].

Some years later, Ken Thompson wrote in a famous article: ”To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software [13].”

Thompson delivered the original Reflections On Trusting Trust with the idea and article about hidden Trojan horse in the binary parent compiler which is used to compile the next version of the compiler [13].

In the years after Thompson’s original article, compiler security and build security have emerged as an increasingly important research area. The potential for deceptive bugs to propagate in binary without being seen in the source code implies that the possibility for enormous damage is technically feasible. For example, if the GNU Compiler Collection (GCC) would contain such a bug and is used to build the next version of Bitcoin core, an individual or a Government organization could be able to control the cryptocurrrency centrally. This project will investigate the threats, vulnerabilities and means of countermeasure for these types of deceptive compilers.

In a 2015 article named Defending Against Compiler-Based Backdoors, the compiler Trojan horse is discussed and the countermeasures that can be taken to mitigate the threat [11].

The compiler attacks have been called deniable in Deniable Backdoors Using Compiler Bugs due to the fact that the attack cannot be seen when viewing the source code of the

compiler. This term is taken from the term deniable cryptography [1].

The authors David and Baptise writes in their article How a simple bug in ML compiler could be exploited for backdoors that ML, which typically is used to create other languages and compilers, has a theoretical risk of being compromised as described in the original RooT attack [4].

In the recent article named Hypothetical Attacks on Bitcoin Core the author describes a number of techniques and social engineering that would possibly constitute a threat to the cryptocurrency and its build procedures [3]. None or almost none of the attacks have occured in practice. The article did not mention any compiler Trojan hypothesis.

During my own research of the topic I study, create and evaluate some new and established techniques of diverse double-compiling in the context of the build system for Bitcoin Core [9].

David A. Wheeler wrote that randomized testing or fuzz testing would be unlikely to detect a compiler Trojan [15]. Fuzz testing or randmized testing tries to find software defects by creating many random test programs (compared to a large number of monkeys at the keyboard). In the case of testing a compiler, the compiler-under-test will be compared with a reference compiler, and the outcome of to test will be best on whether or not the two compilers produced different binaries. This apprach is described by Faigon in his article ”Zero Bugs” [5]. The approach has been successful in finding many software bugs and compiler errors, but it will be extremely unlikely to detect maliciously corrupted compiler if the compiler only diverts from its specification in 1/1000 executions of its target (as would be the case in a cryptocurrrency system that sends every thousand transaction to a different receiver). For randomized testing to work for compiler-compilers, the input would need to be a randomized new version of the compiler.

The circumstances are that it’s a compiler being compiled and that Trojan horse only targets very specific input such as the compiler itself. On the other hand, it’s a rather general statement which doesn’t go into detail why we cannot input the source code of the compiler to a compiler binary and then do fuzz testing with variations on that.

A sufficiently complicated project often takes a lengthy and complex build process with many different binaries from several different vendors that are utilized as

dependencies and libraries in the toolchain. Compiling Bitcoin Core locally for example takes 40 minutes with an Intel I7 CPU. Compiling GCC itself takes several hours, and compiling Firefox takes a whole day.

1. Ken Thompson's original article named Reflections on Trusting trust [13]. This article is a good introduction to the problem and subject.
2. Defending Against Compiler-Based Backdoors [11].
3. Countering trusting trust through diverse double-compiling [14]. This article describes the DDC in detail and provides the history and context.
4. The original Bitcoin paper and the ongoing work on the build system of Bitcoin Core [9] [2]. I plan to use a code pipeline or create a code pipeline myself for Bitcoin Core in order to examine the builds more easily.

Additionally the following articles might seem relevant but I did not yet study them.

1. How a simple bug in ML compiler could be exploited for backdoors? [4].
2. Deniable Backdoors Using Compiler Bugs [1].

The original Bitcoin article took the project of solving the problem of double-spending

> We propose a solution to the double-spending problem using a peer-to-peer network.

My aim is to give answers and elaborations for the following questions.

1. What are the threats and the vulnerabilities that should be protected against with DDC? What are the technical benefits and shortcomings of DDC compared to other solutions to real and potential problems with compiler vulnerability and build security? What critique is known against DDC and what other critique is reasonable to give?
2. What does a real demonstration of DDC look like? Show (don't tell) the audience a real example of an actual procedure.
3. How can DDC be applied to the build system of Bitcoin Core and what can we learn from it? What would be the challenges and benefits? Which part of the

build process is more likely to be vulnerable than other parts? Can I recompile Bitcon Core as the system-under-test and what will that result be?

4. What possible and provable improvements can be made compared to the current state of the art and how can the technology become portable and easily available for software engineering teams and researchers? Can a compiler binary be made more auditable and examined and if yes, how??

Modern OSs randomize memory section addresses (it makes some attacks more difficult), so if anyone restarts the process, the addresses of the instructions might be different.

I will use the build process of Bitcoin Core as my main applied DDC. I will find out about relevant frameworks, build systems and platforms, such as GUIX (https://guix.gnu.org), maybe Frama-C (https://frama-c.com) and/or the CompCert project (https://compcert.org/).

I can create checks and provisioning a system in a Docker container for portability and reproducibility. I have access to the Bitcoin build toolchain and can build Bitcoin Core with GCC [12]. I can create the demonstration of a compiler Trojan horse in the source code of a compiler to learn and show the concepts for real. I can perform the checks and verify that results are according to expectations [12]. I will learn about and use the Bitcoin core build system and its compiler [2]. The 4 main attacks that can happen directly as a result of the 51% attack are: Selfish mining. Cancelling transactions. Double Spending. Random forks.. I will learn and use the GUIX build system. I will also explore some ideas that are new or rarely used in compiler and build security such as comparing checksums from known ”safe” builds with new builds of the same source [10].

In this chapter, a detailed description about background of the degree project is presented together with related work. Discuss what is found useful and what is less useful. Use valid arguments.

Explain what and how prior work / prior research will be applied on or used in the degree project /work (described in this thesis). Explain why and what is not used in the degree project and give valid reasons for rejecting the work/research.

Use references!

## 2.1 Use headings to break the text

Do not use subtitles after each other without text in between the sections.

### 2.1.1 Related Work

You should probably keep a heading about the related work here even though the entire chapter basically only contains related work.

# Chapter 3

# <Engineering-related content, Methodologies and Methods>

Technologies in provable security have achieved increasing interest in software engineering and related disciplines. With DDC one compiles the source code of the compiler C0 with a proven other compiler C1. Then use the result B1 to compile the compiler C0 and compare it with a compilation from the distributed compiler.

After many years, the idea of countermeasures the Trojan horse using diverse double-compilation appeared

[9].

The ethical grounds are to show openness (open research) and transparency of what is being done, even during the research work as well as after its completion. The ethical ground of white hat hacking are set in this way. (I should refer to papers and posts to read about this. It would be good to discuss some of them in the thesis.)

Describe the engineering-related contents (preferably with models) and the research methodology and methods that are used in the degree project.

Most likely it generally describes the method used in each step to make sure that you can answer the research question.

## 3.1 Engineering-related and scientific content:

Applying engineering related and scientific skills; modelling, analysing, developing, and evaluating engineering-related and scientific content; correct choice of methods based on problem formulation; consciousness of aspects relating to society and ethics (if applicable).

As mentioned earlier, give a theoretical description of methodologies and methods and how these are applied in the degree project.

# Chapter 4

# The work

A difficult challenge with the compiler backdoor Trojan horse is to actually create a real version of the threat.

There are a lot of claims about the Thompson compiler backdoor ( https://stackoverflow.com/questions/781718/thompsons-trojan-compiler/782677 ) without anyone actually showing that the attack can be done in practice.

I did put a complete reproducible example with TinyC on GitHub https://github.com/montao/tinycc/blob/ddc/.github/workflows/action.yml but it had a flaw to begin with which did make the attack unrealistic at first: I could make the Trojan replication to all future versions of the compiler but that is only possible if an extra header file with the attack vector is available when future versions of the compiler is compiled. The header will not be available like that so my attack would not be realistic.

For example if I try to corrupt the TinyC so that it inserts the attack code to future versions I can put an include statement right before it opens the file to compile:

Listing 4.1: C example

```
    ...
   /* open the file */
#include "attack.c"
    fd = _tcc_open(s1, filename);
    ...
```

The file attack.c contains the attack code and will be included by the preprocessor. The attack code detects my dummy login program and lets the username hackerken login without a password. The attack code is also replicated to all future versions of the compiler when TCC compiles itself but only if the attackvector.h is supplied to future generations. How is the attack supposed to take place without supplying the header file? It seems that I got a circular dependency in my attack code.

Listing 4.2: C example

```
#include "attackvector.c"

    // naive signatures for now
    // needs to ./generate-attackvector < attack.c > attackvector.h
    static char compile_sig[] = "/* open the file */";
    static char login_sig[] = "int login(char *user) {";
    static char login_attack[] = "if(strcmp(user, \"hackerken\") == 0) re

    /* attack code follows */
    ...
```

The file attackvector.c is generated from running generate-attack-array

Listing 4.3: C example

```
#include <stdio.h>

int main(void) {

    printf("static char compile_attack[] = {\n");
    int c;
    while ((c = fgetc(stdin)) != EOF) {
        printf("\t%d,\n", c);
    }
    printf("\t0\n};\n\n");
    return 0;
}
```

...so that the attackvector.h contains the array with the attack.c source code itself.

Listing 4.4: C example

```
static char compile_attack[] = {
    47,
    47,
    32,
    ....
```

But that makes the backdoor require that all future versions of TCC is compiled with the header file attack.array.h making the backdoor dependent on the header file being present, which it won't be from an honest clone of the compiler.

The way I fixed it to not need the extra header file was to compile the .c file twice and paste it into the char array and like that it will self-reproduce any (finite) number of generations without being seen in the source code.

Parallell builds with GNU make -j 4 or 8. Get number of cores.

In general it is often desirable to make every technical system secure. The specific problem in this case is the trick of hiding and propagating malicious executable code in binary versions of compilers - a problem that was not mitigated for 20 years during 1983 and 2003.

In the context of security engineering and cryptocurrency, a security engineer might face a reversed burden-of-proof. The engineering wouod need to prove security which is known to be hard, both for practical reasons and for the reason of which and what kind of security models to use. For example, a security engineer might want to demonstrate a new cryptographic alternative to BTC and blockchain. Typically the engineer might get the question if he can prove that there is no security vulnerability.

At least 7 or 8 different operating systems and at least two very different compilers can build Bitcoin Core ( https://github.com/bitcoin/bitcoin/tree/master/doc )

Describe the degree project. What did you actually do? This is the practical description of how the method was applied.

# Chapter 5

# Result

Here I describe the results of the project.  It was possible to implement the trusting trust attack based on Thompson's original description.

# Chapter 6

# Conclusions

If we observe that the two diverse double-compiled compiler binaries are identical then we know that our build system is safe. But if they are not identical we haven't proved that our build system is compromised.

Describe the conclusions (reflect on the whole introduction given in Chapter 1).

Discuss the positive effects and the drawbacks.

Describe the evaluation of the results of the degree project.

Describe valid future work.

The sections below are optional but could be added here.

## 6.1 Discussion

Vice-versa double-compiling: Switch the compiler under-trust that will relax the assumption which one is trusted.

Generated many new randomized version of the compilers

Describe who will benefit from the degree project, the ethical issues (what ethical problems can arise) and the sustainability aspects of the project.

Use references!

### 6.1.1 Future Work

Could it be an idea to use two compilers without knowing which one of them is trusted and perform DDC twice, first trust compiler A and then trust compiler B. Then it wouldn't matter which one is trustworthy as long as both of them are not compromised. Sounds like a good and novel idea. However, maybe as a future research direction instead of a partial inquiry (since you already have lots of things to present and evaluate).

### 6.1.2 Final Words

**If you are using mendeley to manage references, you might have to export them manually in the end as the automatic ways removes the ”date accessed” field**

# Bibliography


[1] Bauer, Scott. *Deniable Backdoors Using Compiler Bugs*. 2015. URL: https://www.alchemistowl.org/pocorgtfo/pocorgtfo08.pdf.

[2] Bitcoin. *Bitcoin-core*. https://github.com/bitcoin/bitcoin. 2021.

[3] Chipolina, Scott. *Hypothetical Attack on Bitcoin Core*. 2021. URL: https://decrypt.co/51042/a-hypothetical-attack-on-the-bitcoin-codebase.

[4] David, Baptiste. "How a simple bug in ML compiler could be exploited for backdoors?" In: *arXiv preprint arXiv:1811.10851* (2018).

[5] Faigon, Ariel. *Testing for zero bugs*. 2005.

[6] Jones, Gareth J., Edwards, Michael B., Bocarro, Jason N., Bunds, Kyle S., and Smith, Jordan W. "Leveraging community sport organizations to promote community capacity: Strategic outcomes, challenges, and theoretical considerations". In: *Sport Management Review* (2017). ISSN: 14413523. DOI: 10.1016/j.smr.2017.07.006. URL: http://dx.doi.org/10.1016/j.smr.2017.07.006.

[7] Karger, Paul A and Schell, Roger R. *Multics Security Evaluation Volume II. Vulnerability Analysis*. Tech. rep. ELECTRONIC SYSTEMS DIV LG HANSCOM FIELD MASS, 1974.

[8] Liu, Bo, Zhang, Yanxin, Liu, Fangcui Jiang Lei, and Gao, Yanping. "Group consensus for a class of discrete-time heterogeneous multi-agent systems in directed topology". In: *Proceedings - 2017 32nd Youth Academic Annual Conference of Chinese Association of Automation, YAC 2017* (2017), pp. 376–381. DOI: 10.1109/YAC.2017.7967437.

[9] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized Business Review* (2008), p. 21260.

[10] Nikitin, Kirill, Kokoris-Kogias, Eleftherios, Jovanovic, Philipp, Gailly, Nicolas, Gasser, Linus, Khoffi, Ismail, Cappos, Justin, and Ford, Bryan. "{CHAINIAC}: Proactive software-update transparency via collectively signed skipchains and verified builds". In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1271–1287.

[11] Regehr, John. *Defending Against Compiler-Based Backdoors*. 2015. URL: https://blog.regehr.org/archives/1241.

[12] Rosencrantz, Niklas. *Compile Bitcoin Core*. Youtube. 2021. URL: https://www.youtube.com/watch?v=A3NnKuKwiAg.

[13] Thompson, Ken. "Reflections on trusting trust". In: *ACM Turing award lectures*. 2007, p. 1983.

[14] Wheeler, David A. "Countering trusting trust through diverse double-compiling". In: *21st Annual Computer Security Applications Conference (ACSAC'05)*. IEEE. 2005, 13–pp.

[15] Wheeler, David A. "Fully countering trusting trust through diverse double-compiling". PhD thesis. George Mason University, 2010.

# Appendix - Contents

# Appendix A

# First Appendix

This is only slightly related to the rest of the report

# Appendix B

# Second Appendix

this is the information