

Alexander Blyth

June 2021

# Contents

0.1	Topic	c Definition		
0.2	Backg	round		
	0.2.1	Information Security		
	0.2.2	Information Flow Control		
	0.2.3	Information Flow Security in Concurrency		
	0.2.4	Compilers and Security		
	0.2.5	Annotations		
	0.2.6	Related Work		
0.3	Appro	ach		
	0.3.1	Test Cases		
	0.3.2	Quality Analysis		
	0.3.3	Efficiency and Optimisation		
	0.3.4	Tool Development		
0.4	Execu	tion		
	0.4.1	CompCert AIS Annotations		
	0.4.2	Inline Assembly		
	0.4.3	CompCert Builtin Annotations		
	0.4.4	LLVM Compiler Modification		
Appen	Appendices 23			

Abstract - Information flow security in concurrency is difficult due to the increasing complexity introduced with multiple threads. Additionally, compiler optimisations can break security guarantees that have been verified in source code. In this paper, we propose a thesis to explore these issues through providing annotations in C source code that propagate through to the binary or assembly. These annotations could then be used to guide a static analysis of information flow security in concurrency. This approach involves (1) capturing C source code annotations provided by the user about the security policy of data and variables and (2) passing these annotations down to lower representations where static analysis tools can be utilised to identify security vulnerabilities in the produced binary.

# 0.1 Topic Definition

This paper describes the motivation, background knowledge and plan for the proposed thesis Compiler Annotation Solutions for Concurrent Information Flow Security.

There is a high degree of complexity in verifying security guarantees in concurrent programs [19][27][29]. Additionally, aggressive compiler optimisations can modify the binary output in unexpected ways [7]. To preserve the security of a program, the flow of sensitive information must be protected to avoid flowing in to untrusted sources [2]. This is where static analysis tools can be used to verify the integrity of security guarantees and the flow of sensitive information. In this thesis, we look to explore a solution to information flow security in concurrent programs through analysing the output after aggressive compiler optimisations.

We propose a tool to analyse C programs to detect security violations in information flow control. This tool will preserve annotations provided by the programmer in source code through lowering passes and aggressive compiler optimisations. The tool will work alongside the Weakest Precondition for Information Flow (wpif) transformer described by Win-

ter et al. [31] to allow the programmer to assess the security of information flow in their concurrent programs.

Similar tools for propagating annotations and properties through compiler optimisations have been explored [30] [25] [18], however, these tools focus on either generic solutions for propagating properties or to assist the static analysis of the Worst Case Execution Time.

# 0.2 Background

Vulnerabilities in software can lead to catastrophic consequences when manipulated by attackers. In an open-source cryptographic software library (OpenSSL) used by an estimated two-thirds of web servers [16] a security flaw called Heartbleed was discovered. Secure secrets such as financial data, encryption keys, or anything else stored in the server's memory could be leaked. Normally, one would send a Heartbeat request with a text string payload and the length of the payload. For example, a message of "hello" could be sent with the length of the message, 5. However, due to a improper input validation (buffer over-read), one could send a length longer that the string they actually sent. This would cause the server to respond with the original message and anything that was in the allocated memory at the time, including any potentially sensitive information. An example of this is shown in figure 1 [13].

Heartbleed was one of the most dangerous security bugs ever, and calls for major reflection by everyone in industry and research [2].

### 0.2.1 Information Security

Computer security is defined as a preservation of integrity, availability and confidentiality of information, and extends to include not only software but hardware, firmware, information, data and telecommunications [15]. Confidentiality requires that data is not available to unauthorised users, and that individuals can control what information can be collected and disclosed to others. Data integrity requires that only authorised sources can modify data,

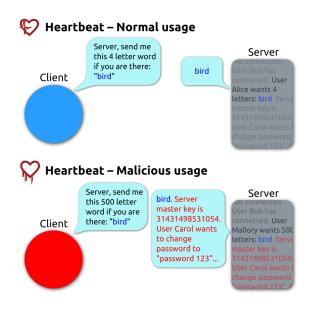


Figure 1: The Heartbleed bug. [13]

and that the system can perform tasks without interference from outside sources. Finally, availability of a system requires that service is not denied to authorised users. Together, these principles create the CIA triad [28]. To enforce a secure system, all three principles must be upheld.

Modern programs are becoming increasingly complex with potential for networking, multi-threading and storage permissions and more. As such, security mechanisms must be put in place to verify and enforce the information security requirements. The adequacy of a security mechanism depends on the adversary model. The adversary model is a formal definition of the attacker and their abilities in a system, and defines who we are protecting against [10]. Ideally we would like to design a system to protect against the strongest adversary or attacker, however, this is often not required or even possible. Instead, we must consider the security policy, security mechanism and strongest adversary model to make a system secure [2].

Standard security processes that handle access control such as a firewall or antivirus software can

```
secret := 0xCODE mod 2
public := 1
if secret = 1
public := 0
```

Figure 2: Implicit flow of data to a public variable

fail as they do not constrain where information is allowed to flow, meaning that once access is granted it can propagate to insecure processes where it can be accessed by attackers. Where a large system is being used, it is often the case that not all components of the codebase can be trusted, often containing potentially malicious code [24]. Take for example your modern-day web project. Where a package manager such as Node Package Manager (npm) could be used to utilise open-source packages to speed up development progress, it could also inadvertently introduce security vulnerabilities. Rewriting all packages used to ensure security would be time-consuming and expensive and is not a viable option. Instead, controlling where information can flow and preventing secure data from flowing into untrusted sources or packages can maintain confidentiality of a system.

One may suggest runtime monitoring the flow of data to prevent leakage of secure data. Aside from the obvious computational and memory overhead, this method can have its own issues. Although it can detect an *explicit* flow of data from a secure variable to a public variable, it is unable to detect *implicit* data flow, where the state of secure data can be inferred from the state of public data or a public variable [9]. Take for example figure 2. In this example, a public, readable variable is initially set to the value of 1. There is also a secret variable which may contain a key, password or some other secret that must be kept secure from any attackers. Depending on the value of the secret variable an attacker can infer information about this variable depending on whether the value of the public variable is updated to a value of 0. Assuming that the inner workings of the system is known by the attacker, information about the secret variable can be leaked implicitly and inferred by the state of public variables.

Security concerns do not only exist at the application level. In a huge codebase such as an OS, different low-level bugs can be exploited to gain access to data, such as by using buffer overflows to inject viruses or trojans [1].

# High Variable Low Data Low Variable

Figure 3: Permitted flow of data

### 0.2.2 Information Flow Control

As seen by the issues that can be introduced via implicit and explicit flow of data, there is room to improve on the existing techniques imposed by current security measures. To protect confidentiality, secure or sensitive information must be prevented from flowing into public on insecure variables. Additionally, to protect integrity, untrusted data from public sources must be prevented from flowing into secure or trusted destinations [2]. An information flow security policy can be introduced to classify or label data, or more formally, a set of security levels to which each object is bound by across a multi-level security lattice [8]. In this thesis, we will focus primarily on preserving confidentiality.

Many security levels can be identified to classify different classes of objects, however, for now we will consider two security levels: high and low. Data labelled as high signifies that the data is secret, and low data is classified as non-sensitive data, such that it does not need to be protected from an attacker or adversary. Variables that can hold data in a program can additionally be classified as high or low as a security classification. A variable's security classification shows the highest classification of data it can safely contain [31]. A high variable can hold both high and low data, whereas a low variable which is visible to an attacker can only safely hold low data. As mentioned previously, confidentiality must be upheld by preventing high or secret data from flowing to low or public variables where an attacker can observe it. The permitted flow of data can be observed in 3. Note that high data is not allowed to flow into low variables.

# 0.2.3 Information Flow Security in Concurrency

Controlling the flow of information is a difficult problem, however, this is only exacerbated in concurrent programs, which are a well known source of security issues [19][27][29]. Research has been conducted into concurrent programs to explore ways the security of concurrent programs can be verified. Mantel et al. [20] introduced the concept of assumption and guarantee conditions, where assumptions are made about how concurrent threads access shared memory and guarantees are made about how an individual thread access shared memory that other threads may rely upon. Each thread can be observed individually using assumptions over the environment behaviour of other threads that can be then used to prove a guarantee about that individual thread. As two concurrent threads can interleave their steps and behaviour, there is a lot of complexity and possibilities for the overall behaviour. This concept of assumptions (or rely) and guarantee conditions can reduce the complexity of understanding interleaving behaviour in threads and assist in verifying the correctness of information flow security in concurrency. However, this approach is limited in the types of assumptions and guarantees it supports. Building on this, Murray et al. [12] [21] provide information flow logic on how to handle dynamic, value-dependent security levels in concurrent programs. In this case, the security level of a particular variable may depend on one or more other variables in the program. As such, the variable's security level can change as the state of the program changes. This logic is essential where the security level of data depends on its source. However, this approach is not sufficient when analysing non-blocking programs. The approach relies heavily on locks which block particular threads from executing. This in turn leads to slower processing due to blocked threads [23].

To overcome information flow security in nonblocking concurrent threads, Winter et al. explores verifying security properties such as noninterference through the use of general rely/guarantee conditions using backwards, weakest precondition reasoning. Such an analysis would additionally handle implicit flows as shown in figure 3. Ideally a tool could be created to verify security policies required for sensitive processes. Users of this system could provide rely/guarantee conditions for each thread as well as security levels for data and variables i.e. high or low data and variables. Working backwards through the execution of the program, violations of the security policy will be detected. Detected violations could be due to an incorrect assumption of the rely and guarantee conditions or a failure to uphold the security policy. This thesis will focus on the compilation stage of this tool.

### 0.2.4 Compilers and Security

Compilers are well known to be a weak link between source code and the hardware executing it. Source code that has been verified to provide a security guarantee, potentially using formal techniques, may not hold those security guarantees when being executed. This is caused by compiler optimisations that may be technically correct, however, a compiler has no notion of timing behaviour or on the expected state of memory after executing a statement [7]. This problem is known as the *correctness* security gap. One example of the correctness security gap is caused by an optimisation called dead store elimination. Figure 4 was derived from CWE-14 [6] and CWE-733 [5] and used by D'Silva et al. [7]. Here a secret key was retrieved and stored in a local variable to perform some work. After completing the work, and to prevent sensitive data from flowing into untrusted sources, the key is wiped from memory by assigning it the value 0x0.

From the perspective of the source code, a programmer would expect the sensitive data from key

```
crypt() {
    key := 0xCODE // Read key
    ... // Work with the key
    key := 0x0 // Clear memory
}
```

Figure 4: Implicit flow of data to a public variable [7]

to be scrubbed after exiting the function. However, key is a variable local to the function. As key is not read after exiting the function, the statement that assigns key to a value of 0x0 will be removed as part of dead store elimination. This results in lingering memory that could be exploited by an attacker. In GCC, with compiler optimisations on, dead store elimination is performed by default [22]. Additionally, dead store elimination has been proven to be functionally correct [3][17].

This leads to the question, what security guarantees in source code are being violated by compiler optimisations? Although one could analyse each individual compiler optimisation to check for potential security violations in source code, defensively programming against the compiler can be counterinitiative. Additionally, compilers are getting better at optimising away tricks programmers write to work against the compiler, and thus is not a future-proof solution [26]. One might also suggest turning compiler optimisations off, however, this leads to slower code. In a concurrent system where execution time is critical, turning compiler optimisations off is not a viable option. Instead an alternative solution is to perform a static analysis on binary or assembly for security violations. As compilation has already been executed, such analysis would reveal security guarantee violations that result due to compiler optimisations.

### 0.2.5 Annotations

This project can take two routes; the proposed tool will be required to run an analysis on either binary or assembly. For either route, annotations used to guide a static security analysis will need to be provided by the user in the C programs they write. The tool will then be required to propagate these annotations down to compiled forms, i.e. binary or assembly. From here, a static analysis can be conducted as described by Winter et al. [31]. Ideally these annotations can be propagated through with little to no modification of the C Compiler being used as to reduce complexity and increase modularity and reusability of such a a tool. However, it is unclear as to whether passing annotations down with no modification to the compiler is currently possible. In this thesis, this issue will be explored.

Running a static analysis on a binary can be difficult due to the low level nature of a binary file. As such, to sufficiently perform such an analysis, the binary would be required to be decompiled to a higherlevel form, such as an assembly file. From here a static analysis could be conducted. The alternative approach would be to perform the analysis directly on the compiled assembly output files rather than reducing these to binary. Currently, it is unclear as to what compiler optimisations are made when reducing an assembly file to binary, and will be explored further throughout the lifetime of this thesis. The flow of information can be viewed in Figure 5, where formats a static analysis can be performed are outlined in a dashed line. In GCC, "temporary" intermediate files can be stored using the flag save-temps [14]. These stored files can then be used for analysis.

### 0.2.6 Related Work

In safety-critical real-time software such as flight control systems, it is required to analyse the Worst Case Execution Time (WCET). This kind of analysis can be conducted using static analysis tools to estimate safe upper bounds. In the case of AbsInt's aiT tool this analysis is conducted alongside compiler annotations to assist where loop bounds cannot be computed statically. In these cases, the user can provide annotations to guide the analysis tools [25]. This tool builds on an existing annotation mechanism that exist in CompCert, a C compiler that has been formally verified for use in life-critical and mission-critical software [4][18]. CompCert an-

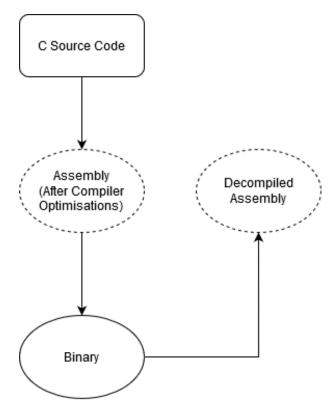


Figure 5: The static analysis options after compilation.

notations are not limited to WCET analysis. general mechanism for attaching free-form annotations that propagate through to assembly files can be achieved with CompCert. This approach is able to reliably transmit compiler annotations through to binary through method calls which are carried through compilation and the linked executable without using external annotation files. CompCert prints annotation strings as a comment in the generated assembly code, and an additional tool is used to parse these comments and generate annotations. However, due to its treatment as an external function, annotations cannot be placed at the top level of a compilation unit, unlike a variable declaration. Compiler optimisations can additionally cause further issues when trying to preserve annotations through compilation. If dead code is eliminated, annotations associated with that code can be lost as well. Extra care needs to be taken to avoid these optimisations destroying links between properties and the code they refer to during such transformations.

TODO: Include further documentation on how to use Compiler & inline assembly

A similar approach to CompCert is used by The ENTRA (Whole-Systems ENergy TRAnsparency). As part of providing a common assertion language, pragmas are used to propagate information through to comments in the assembler files. Information is retained in LLVM IR and ISA representations. However, these annotations are not stored in the final binary and thus comments must be extracted from assembler files [11].

Vu et al. [30] explore capturing and propagating properties from the source code level though though lowering passes and intermediate representations. Their goal was to maintain these properties to binary through aggressive compiler optimisations. As compilers only care about functional correctness, they have no notion of the link between properties and the code it refers to. Thus, there is no way to constrain transformations to preserve this link or to update these properties after the transformation. As such, they approached the problem to create a generic solution, modifying a LLVM compiler with virtually no optimisation changes. This was done by creating a library in LLVM. The prop-

erties were stored in strings, and these strings were parsed to build a list of observed variables and memory location. A LLVM pass was inserted to store all these properties in metadata. After each optimisation pass, a verification pass was inserted to check the presence of metadata representing the properties, variables amd memory locations. If an optimisation pass had cased the verification to fail the programmer would then be notified, to which they could annotate differently or disable the optimisation.

# 0.3 Approach

The approach was set out by first analysing existing methods of preserving annotations through intermediate representations. These include the:

- compCert Verified C Compiler,
- GNU Extension for Extended Inline Assembly, and
- Modifying the LLVM compiler to preserve annotations throughout intermediate representations.

Each of these approaches will be analysed individually for viability across each of the test cases outlined in section 0.3.1. For approaches that pass all necessary test cases, a further analysis will be conducted into its suitability and development of any necessary tools to assist in the preservation technique, as outlined in sections 0.3.2 and 0.3.4. Finally, an analysis on the runtime efficiency of the program will be conducted to assess the success of the annotations with various levels of optimisation. The approach for this analysis is outlined in 0.3.3.

### 0.3.1 Test Cases

A suite of test C programs (See Appendix .1) were created to assist in guiding the process of evaluating each approach as a possible means of preserving annotations. Each program has inline comments documenting the annotation that should be preserved and its location within the program. Additionally, each program aims to test a separate element required to perform a static wpif analysis. Namely, these are to preserve the following through to the assembly output:

- 1. comments,
- 2. simple and complex variables (e.g. struct elements and volatile global variables),
- 3. security policies,
- 4. predicates on the initial state, and

### 5. loop invariants.

Each test was conducted to assess the viability of each approach of preserving annotations. If the approach cannot preserve all the required annotations described in the aforementioned list, then it is not viable for a wpif analysis and another technique must be explored.

The justification for each of the test files are as follows:

#### comment.c

This test case is primarily a stepping stone to testing more complex scenarios. Here we have a generic comment "critical comment" and we are looking to preserve it through to the assembly. As well as preserving the comment itself, the location of the comment within the source code is to be preserved.

#### variable.c

The test file *variable.c* builds off *comment.c*, however, we are additionally looking to preserve annotations about local variables within the program. Here multiple variable types are tested:

- int,
- char,
- unsigned int,
- short,
- long.
- float, and
- double.

With each of these variables their type data is included as an annotation. This test is particularly interesting as with higher levels of optimisation we can observe how the annotations behave when a variable is optimised out.

### volatile.c

This test program looks at how the technique handles volatile variables. A variable declared as volatile tells the program its value could change unexpectedly. This is especially important when dealing with concurrent programs. If the technique cannot handle

volatile variables it is unable to be used for a wpif analysis.

### loop.c

This test program tests how the annotator handles loops and loop invariants. It contains security policies, predicates on the initial state and loop invariants.

#### rooster.c

The test program, *rooster.c* delves into a more complex program, combining several features of the previous tests. It contains annotations within functions and global variables.

### password.c

This program tests how annotations are preserved within structs, a user-defined data type. Additionally, *password.c* is a more complex program with multiple functions.

#### deadStoreElimination.c

Testing dead store elimination is a bit more complex, as it requires comparing the compiled output before and after compiler optimisations are turned on. Here, the test program simulates the program described in section 0.2.4.

### pread.c

The program *pread.c* is a culmination of all the previous test cases, and is similar to *loop.c*, however, the global variables within it are volatile. It requires all the necessary components for a wpif analysis.

### 0.3.2 Quality Analysis

Although a method of preserving c annotations may be able to successfully pass all the test cases, it is important to avoid modifying the assembly instructions. The reason for performing a static analysis on the compiled output is due to the optimisations performed by the compiler. As such, it is important to ensure that preserved annotations do not remove or undo any optimisations that may have been performed by the compiler.

The methodology for testing quality in this manner is to compare the compiled assembly output for a program with annotations on to the compiled assembly output for the same program without annotations. If unnecessary assembly instructions have

been added, it is indicative that the annotations has modified the program in unintended ways.

### 0.3.3 Efficiency and Optimisation

In the case that the annotations have introduced additional statements into the compiled assembly output, understanding the extent of these changes is important. Here, a efficiency analysis can be conducted on the assembly. Using big O notation, an upper bound can be placed on the program. Doing so allows for a comparison of the efficiency of the annotated and non-annotated assembly.

Let A(n) be the function describing the annotated assembly, and B(n) be the function describing the non-annotated assembly. Then,

$$A(n) \in \Theta(g(n))$$

$$B(n) \in \Theta(h(n))$$

If the non-annotated assembly has a lower bound than the annotated assembly, such that

$$h(n) \in O(q(n)), and$$

$$q(n) \notin O(h(n))$$

then the annotations have modified the program in a way that reduces runtime efficiency. It is important to detect when this has happened, as it indicates the annotations have reversed the intended compiler optimisations.

In the case where the annotation process has resulted in additional assembly instructions inserted into the compiled output, however, they do not reduce runtime efficiency in terms of big O notation, a empirical analysis of the runtime duration of a program can be conducted to assess the disadvantage of the annotated program.

### 0.3.4 Tool Development

In cases where it is appropriate, a tool may be developed to assist in the annotating process. This tool may either:

• assist in the annotating process,

- verify the correctness of the annotations, or
- perform additional analysis on the compiled output.

If the approach of modifying the LLVM compiler is pursued, developing such a tool to assist the annotating process will be necessary.

### 0.4 Execution

Experimentation began with the CompCert compiler and the provided assembly annotation tools, outlined in section 0.4.1. It was found that the CompCert compiler could not handle all cases necessary for the wpif analysis, specifically volatile variables. As a result, the testing moved on to other techniques. Following this, the GNU C extension for inline assembly was explored as a possibility to preserve annotations in C in section 0.4.2. This technique prevailed and was found to be excellent in handling assembly annotations by injecting comments in to the compiled assembly output. This technique was enhanced by developing a python program to inject inline assembly into the source C files to allow for enhanced analysis and furthermore avoids restricting the program to GNU extension supporting compilers. As a result of the success, modifying the compiler was not explored due to success documented in other research such as the work conducted by Vu et al. [30]. This allowed for further development and improvement of the inline assembly method.

### 0.4.1 CompCert AIS Annotations

CompCert is unfortunately not a free tool, however, for research purposes it can be used freely. The specifications of the CompCert install can be seen in Table 1.

Testing was initially conducted using the *comment.c* test file. The goal is to propagate the comment down to assembly where it can be used and interpreted. To do so, the comment in the source code needs to be replaced with a call to generate an annotation in the compiled assembly. Fortunately, with the CompCert compiler, this functionality is builtin.

000	TT TTC
OS Name	Ubuntu 20.04.2 LTS
OS Type	64-bit
Processor	Intel® $Core^{TM}$ i7-6700K CPU
	$@4.00\mathrm{GHz} \times 8$
Instruction Set	x86-64
CompCert	The CompCert C verified
Version	compiler, version 3.7

Table 1: CompCert install specifications

This assembly annotation is created through the use of the \_\_builtin\_annot function described in 0.2.6. The following builtin annotation was placed in line 2, within the main function in *comment.c.* 

```
__builtin_ais_annot("%here Critical Comment");
```

Listing 1: comment.c

Within this annotation, "here is used to represent the location within the program. If the location is not important, "here can be omitted. The comment, "Critical Comment", has been included to represent some kind of critical comment that is required to conduct a static analysis on the output. To compile the source to assembler only the following command was used:

```
$ ccomp comment.c -00 -S
```

Here -O0 is used to specify to perform no optimisations during compilation. The full compiled output can be seen in Appendix .4. Below is a snippet of the compiled assembly.

```
16    .cfi_endproc
17    .type main, @function
18    .size main, . - main
19    .section
        "__compcert_ais_annotations","",@note
20    .ascii "# file:comment.c line:2
        function:main\n"
21    .byte 7,8
22    .quad .L100
23    .ascii " Critical Comment\n"
```

Listing 2: comment-O0.s

The annotation is stored within assembler directives. Assembler directives are not a part of the pro-

cessor instruction set, however, are a part of the assembler syntax. Assembler directives all start a period (.). On line 19 a new section has been created, named "\_\_compcert\_ais\_annotations". Following the declaration of the section is an ascii string, locating the source of the annotation within the source program comment.c. Line 23 provides the comment we aimed to preserve with our annotation. Thus, CompCert has shown an initial success in preserving annotations in the form of comments.

Additionally, one major benefit of compCert annotations is that they do not modify the source program, as they are inserted at the end of the program as an assembler directive metadata.

When experimenting with annotated variables, the first issues began to arise. The test file *variable.c* contains several variables with their types to preserve to assembly. The annotations behaved as expected for the types:

- int,
- char,
- short.
- long, and
- any signed or unsigned variations of the above mentioned types.

However, the CompCert annotations does not support floating point types. Upon compiling variable.c the following errors were generated.

```
variable.c:13: error: floating point types
for parameter '%e1' are not supported
in ais annotations
variable.c:15: error: floating point types
for parameter '%e1' are not supported
in ais annotations
2 errors detected.
```

This result shows that it is impossible to use the CompCert embedded program annotations for floating point types, vastly restricting its potential use as a technique for a wpif analysis.

It was discovered soon after that the CompCert  $^{99}$  annotations are unable to handle volatile variables,  $_{100}$  generating the follow error upon compiling *volatile.c.*  $_{101}$ 

```
volatile.c:4: error: access to volatile
   variable 'x' for parameter '%e1' is not
   supported in ais annotations
1 error detected.
```

Unfortunately, this result shows that the CompCert AIS annotations approach is not suitable for wpif analysis. The wpif analysis requires use of volatile variables. This is because the primary purpose of the wpif technique is to verify security policy across concurrent programs. Shared variables within concurrent programs can change at any time, and as such it is imperative that shared variables are marked as volatile. As the CompCert AIS annotations cannot handle volatile variables, annotations required for wpif analysis cannot be generated.

Aside from the aforementioned issues, the CompCert AIS annotations performed excellently in generating annotations. The location of global variables in memory are easily identified, as shown in rooster.c. The CompCert AIS annotations must be placed within a method and called as if it was its own function. This creates some confusion when dealing with global variables. However, placing annotations on global variables at the start of main is a perfectly valid method of preserving these annotations. As the location of the annotation within the program is no longer important, the here format specifier can be omitted.

```
.cfi_endproc
.type main, Ofunction
.size main, . - main
section
 "__compcert_ais_annotations","", @note
.ascii "# file:rooster.c line:6
 function:fun\n"
.byte 7,8
.quad .L100
.ascii " CRITICAL COMMENT\n"
.ascii "# file:rooster.c line:26
 function:main\n"
.byte 7,8
.quad .L107
.ascii " L(mem("
.byte 7,8
.quad goose
.ascii ", 4)) = medium\n"
.ascii "# file:rooster.c line:27
 function:main\n"
.byte 7,8
```

90

```
.ascii " EXCEPTIONAL\n"
```

Listing 3: rooster-O0.s

From rooster.c, the comment "CRITICAL COMMENT" has been annotated from lines 88 to 91, and the comment "EXCEPTIONAL" has been annotated from lines 99 to 102. Most notably, the global variable goose has been annotated from lines 92 to 98. Reconstructed, the string "L(mem(goose, 321 4)) = medium" has been preserved. Thus, the CompCert annotations can successfully preserve annotations on global variables.

Another interesting problem faced when work-325 ing with CompCert AIS annotations is found when working with structs. If the programmer wants 326 to annotate a member of a struct for all structs 328 of that type, each instance of that type of struct 329 must be annotated when using CompCert AIS annotations. This is because CompCert treats 330 annotations. This is because CompCert treats 331 \_\_builtin\_ais\_annot() as a call to an external func-332 tion. As such, an annotation cannot be created from outside a method, similar to when dealing with global variables. An example of this process can be seen in password.c. Within the program, each instantiation of the struct user\_t requires another annotation.

```
user_t* user_admin =
      malloc(sizeof(user_t));
      strcpy(user_admin->name, "admin");
      strcpy(user_admin->password,
19
      "4dm1n__4eva");
      __builtin_ais_annot("%here L(%e1) =
      high", user_admin->password);
      user_admin->balance = 1000000;
22
      user_t* user_alice =
23
      malloc(sizeof(user_t));
      strcpy(user_alice->name,
                                "alice"):
24
      strcpy(user_alice->password,
25
      "!alice12!_veuje@@hak");
       __builtin_ais_annot("%here L(%e1) =
26
      high", user_alice->password);
      user_alice->balance = 783;
27
28
      user_t* user_abdul =
29
      malloc(sizeof(user_t));
      strcpy(user_abdul->name, "abdul");
30
      strcpy(user_abdul->password,
31
      "passw0rd123");
       __builtin_ais_annot("%here L(%e1) =
32
      high", user_abdul->password);
```

```
user_abdul->balance = 2;
```

Listing 4: password.c

The compiled output is as expected, with an annotation within the assembly for each of the annotations created within the source file.

```
.section
 "__compcert_ais_annotations","", @note
.ascii "# file:password.c line:20
 function:setup_users\n"
.byte 7,8
.quad .L100
.ascii " L((reg(\rbp\") + 264)) = high\n"
.ascii "# file:password.c line:26
 function:setup_users\n"
.byte 7,8
.quad .L101
.ascii " L((reg(\"r12\") + 264)) = high\n"
.ascii "# file:password.c line:32
 function:setup_users\n"
.byte 7,8
.quad .L102
.ascii " L((reg(\"rbx\") + 264)) = high\n"
```

Listing 5: password-O0.s

As seen in the assembly annotations, the location of the struct members have been preserved. Line 324 contains the annotation L((reg("rbp") + 264)) = high. This annotation notifies that the variable stored in register rbp with an offset of 264 has a security classification of high. Thus, another success for CompCert AIS annotations.

### Quality Analysis

To complete a quality analysis, a comparison of the assembly will need to be conducted with and without annotations. The CompCert assembly output can be seen in Appendix .3. As we are primarily concerned with the annotated assembly after aggressive optimisations have been performed, the assembly with optimisation level 3 will be compared. To begin with, the assembly for *comment.c* will be compared. Performing a diff on the annotated and non annotated assembly produces the following diff:

```
5 11a12
6 > .L100:
7 17a19,23
8 > .section
        "__compcert_ais_annotations","",@note
9 > .ascii "# file:comment.c line:2
        function:main\n"
10 > .byte 7,8
11 > .quad .L100
12 > .ascii " Critical Comment\n"
```

Listing 6: comment-O3.s diff

The diff explains some interesting differences in the assembly. To begin with, an additional label .L100: has been inserted. The only other notable difference is in the compcert annotations. The reason behind the additional label is to allow the location of the annotation to be identified, as can be seen in line 11 of the diff. Thus, this shows a success. There is no difference in the compiled output, even with aggressive compiler optimisations turned on.

Next, variable-O3.s will be compared.

```
2 < # Command line: variable.c -S -03 -o
      compCert/out/variable-03.s
4 > # Command line: variable.c -S -03
5 11a12,16
6 > .L100:
7 > .L101:
8 > .L102:
9 > .L103:
10 > .L104:
11 17a23,43
       .section
      "__compcert_ais_annotations","", @note
13 >
      .ascii "# file:variable.c line:3
      function:main\n"
14 >
      .byte 7,8
15 >
      .quad .L100
      .ascii " -10 = int\n"
16 >
      .ascii "# file:variable.c line:5
17 >
      function:main\n"
      .byte 7,8
18 >
19 >
      .quad .L101
      .ascii " 98 = char n"
20 >
      .ascii "# file:variable.c line:7
21 >
      function:main\n"
22 >
      .byte 7,8
23 >
      .quad .L102
24 >
      .ascii " -98 = unsigned int\n"
      .ascii "# file:variable.c line:9
25 >
      function:main\n"
     .byte 7,8
```

```
27 > .quad .L103
28 > .ascii " 1 = short\n"
29 > .ascii "# file:variable.c line:11
    function:main\n"
30 > .byte 7,8
31 > .quad .L104
32 > .ascii " 4294967296 = long\n"
```

Listing 7: variable-O3.s diff

Similar to before, additional labels .L100: to .L104: have been inserted to identify the annotations from within the source code. However, with aggressive optimisations turned on, an interesting change has occurred to the annotations. As the variables have been optimised out of the compiled assembly, the annotations no longer make sense. Line 16 of the diff shows the annotation for variable a from comment.c. However, as the variable has been completely optimised out, rather than the location of the register being preserved in the annotation, only the value stored within a has been preserved. In this case, that value was -10.

Although the annotations do not interfere with the compiler optimisations, the compiler optimisations have rendered the annotations useless. Unfortunately, in cases such as these, there is not much to be done.

Another interesting case arises when a loop is optimised out by the compiler. Take for example count.c

Listing 8: count.c

As count will always be zero, the loop will be optimised out when optimisations are turned on. As CompCert treats \_\_builtin\_ais\_annot() as a call to an external function, it too will be optimised out with aggressive compiler optimisations. Let's compare the assembly with and without compiler optimisations.

```
# File generated by CompCert 3.7
2 # Command line: count.c -S -00 -o
      annotated/count-00.s
    .align 16
    .globl main
5
6 main:
    .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset
10
    leaq
          16(%rsp), %rax
          %rax, 0(%rsp)
    movq
    xorl
          %ecx, %ecx
12
    xorl
          %eax, %eax
14 . L100:
15
    cmpl
         %ecx, %eax
16
    jge .L101
17 . L102:
   leal
         1(%eax), %eax
   jmp .L100
19
20 .L101:
   xorl
          %eax, %eax
21
    addq $8, %rsp
22
    ret
23
    .cfi_endproc
24
    .type main, @function
    .size main, . - main
26
27
    .section
      "__compcert_ais_annotations","", @note
    .ascii "# file:count.c line:5
28
      function:main\ntry loop "
    .byte 7,8
29
    .quad .L102
30
    .ascii "
              bound: reg(\"rcx\");\n"
```

Listing 9: count-O0.s

With optimisations turned off, the annotation is preserved, as seen from lines 27-31. Following the annotation, the location of the annotation can be located with label .L102. However, with compiler optimisations, it would be expected that .L100 and .L102 will be optimised out. This can be seen in count-O3.s.

```
xorl %eax, %eax
addq $8, %rsp
ret
cfi_endproc
type main, @function
size main, . - main
```

Listing 10: count-O3.s

As expected, the optimised assembly has completely removed the annotation. In the case of a wpif analysis, this is not a large concern. Although preserving loop invariants is necessary, if the loop is optimised out by the compiler it is no longer of concern and the annotations are no longer necessary.

### 0.4.2 Inline Assembly

Extended asm is a GNU Extension supported by many compilers. As such, it presents itself as an excellent method for preserving annotations to assembly. As it allows programmers to write assembly code within the C program, it provides an opportunity to hook in to this functionality and utilise it for annotation purposes. To begin with, a very simple program will be experimented on to find the limits of inline assembly and to assess if it is fit for wpif analysis purposes. To begin with, *comment.c* will be used.

Within the assembly, an inline comment can be created by inserting a # at the beginning of the line or comment. The goal with this test is to try and preserve the annotation "CRITICAL COMMENT" to assembly by inserting it as a comment within the assembly. The following line was inserted within the main method of *comment.c.* All the source files used for the inline assembly method can be seen in Appendix .5.

```
asm("# CRITICAL COMMENT");
Listing 11: comment.c
```

Here, the call to insert inline assembly is treated as a call to a method, similar to CompCert. The first argument takes the AssemblerTemplate. The AssemblerTemplate is the template used for formatting the input operands, output operands and the goto parameters. In this case only a comment is inserted in assembly, and as such the input operands

and output operands parameters are omitted. The full compiled output can be seen in Appendix .6.

```
@main
    .cfi_startproc
  # %bb.0:
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
12
    mova
         %rsp, %rbp
    .cfi_def_cfa_register %rbp
13
         $0, -4(%rbp)
    movl
14
    #APP
    # CRITICAL COMMENT
16
17
    #NO APP
    xorl %eax, %eax
18
    popq %rbp
19
    .cfi_def_cfa %rsp, 8
    retq
```

Listing 12: comment-O0.s

Lines 15 to 17 show the annotation preserved within the assembly. As can be seen, the compiler does not understand the extent of the assembly provided to it. As such, it treats it as a 'black box', and inserts it where relevant within the assembly instructions. In comparison to CompCert its placement within the assembly is inconvenient. As it is not neatly packaged at the bottom of the assembly it must be parsed out. To distinguish assembly comments from important annotations, the string "annotation: " will be inserted before any annotations to allow for easier parsing. Thus, the annotation described from comment.c would be inserted as:

```
asm ("# annotation: CRITICAL COMMENT");
```

Following, the compiled output would be:

```
#APP
# annotation: CRITICAL COMMENT
# #NO_APP
```

One may suggest using the APP and NO\_APP comments above and below our annotation to identify it, however, these comments are system and compiler dependent. As such, a more robust solution has been used here.

Following this success, annotating the location of a variable is the next challenge. With extended <code>asm</code>

input and output operands are available to allow programmers to work with variables. The goal is to use these mechanisms to identify the location of variables alongside their annotated data. To begin with, the location of a simple integer will be attempted to be preserved. The format of the extended asm is as follows:

```
asm asm-qualifiers ( AssemblerTemplate
    : OutputOperands
    : InputOperands
    : Clobbers
    : GotoLabels)
```

Where asm-qualifiers, OutputOperands, InputOperands, Clobbers and GotoLabels are optional parameters. In this experimentation, asm-qualifiers will not be used as any inline assembly generated will not modify the value of any variables nor jump to any labels.

Previously the AssemblerTemplate was used as a string. However, the string can be templated to locate the value of a variable. Take the following example:

```
asm("# annotation: %0 = int" : "=m"(a))
```

Here %0 is used to refer to the first output operand, a. Output constraints must begin with either a '=' or a '+'. A constraint beginning with a '=' is for variable overwriting an existing value, whereas a '+' is used when reading and writing. Constraints are used to specify what operands are permitted. In this case, 'm' is used, signifying a memory operand is allowed, with any kind of address the machine supports. Compiling the inline assembly with no optimisations creates the following annotation.

```
30 #APP
31 # annotation: -20(%rbp) = int
32 #NO_APP
```

Listing 13: variable-O0.s

This annotation was created without modifying any additional assembly instructions, seemingly a success. However, when optimisations are turned on an interesting result occurs.

```
6 main: #
     @main
7     .cfi_startproc
8 # %bb.0:
```

```
kill: def $edi killed $edi def $rdi
         $-10, -4(%rsp)
    movl
11
    #APP
    # annotation: -4(\%rsp) = int
12
    #NO_APP
13
14
    movl
           -4(%rsp), %eax
          %edi, %eax
    addl
          $134217635, %eax
      0x7FFFFA3
    retq
```

Listing 14: variable-O3.s

Although the annotation is preserved to assembly, a number of optimisations performed by the compiler have been reverted to allow the location of the variable a to be preserved. This is because the compiler does not understand what the inline assembly does outside of what information has been provided to it. In this case, the compiler was provided inline assembly that used some kind of memory operand with the intention to overwrite it. Thus, the compiler was required to remove optimisations to allow the the rewritten value of a to be propagated through the program successfully.

This appears like a failure of the inline assembly method, however, constructing the statement differently should allow for less modification of the assembly by the compiler. Rather than instructing the compiler that the assembly overwrites the existing value, '+' can be used to instruct it that our assembly reads and writes. The following statement is inserted in the C source.

```
asm("# annotation: %0 = int" : "+m"(a))
```

However, this results in the same outcome as Listing 14. The issue occurring here is caused due to the output operand. As the inline asm is notifying the compiler that the value is modified, it removes optimisations to allow this. Instead, using an input operand may allow the optimisation to run, as instead the compiler has been notified that only the value is being read. The following asm is inserted in variable.c.

```
asm("# annotation: %0 = int" : : "m"(a))
```

The '+' constraint has been removed from the operand as it no longer applies to an input operand. Listing 15 shows the assembly with no optimisations.

```
movl
           $0, -4(\%rbp)
23
    movl
           %edi, -8(%rbp)
          %rsi, -16(%rbp)
    movq
25
           $-10, -20(%rbp)
26
    movl
    #APP
    # annotation: -20(\%rbp) = int
28
    #NO_APP
    movsd .LCPIO_0(%rip),
                                      xmmO =
      mem[0],zero
    movss .LCPIO_1(%rip), %xmm1
                                     \# xmm1 =
      mem[0],zero,zero,zero
    xorl %eax, %eax
```

Listing 15: variable-O0.s

Here, the annotation can be seen in line 28, with the location of the variable successfully identified. However, interestingly the instructions from lines 30-32 were previously above the instructions from lines 23-26. Although this makes no difference to the runtime of the program, it is an interesting change caused by the inline asm.

Additionally, the optimised assembly using the input operands has the same issue as before from Listing 14.

```
main:
        @main
          .cfi_startproc
      # %bb.0:
      # kill: def $edi killed $edi def $rdi
          movl
                $-10, -4(%rsp)
          #APP
          # annotation: -4(\%rsp) = int
          #NO_APP
                -4(%rsp), %eax
14
          movl
          addl
                %edi, %eax
          addl
                $134217635, %eax
                                          # imm
      = 0x7FFFFA3
```

Listing 16: variable-O3.s

Further investigation revealed that the issue arises as the inline asm specifies the location of the operand must be in memory with the constraint "m"(a). Rather than limiting the location to memory, allowing any operand available to be used allows for more optimisations to be performed by the compiler. As such, the following line of asm was tested.

```
asm("# annotation: %0 = int" : : "X"(a))
```

Here the constraint "X"(a) specifies that any 47 operand whatsoever is allowed. The resulting optimised assembly is as follows.

```
.text
    .file "variable.c"
    .section .rodata.cst4, "aM", @progbits,4
                                # -- Begin
    .p2align 2
      function main
5 .LCPI0_0:
    .long 1078530011
                                     # float
      3.14159274
    .section .rodata.cst8, "aM", @progbits,8
    .p2align
9 . I.CPTO 1:
    .quad 4608627556095693531
                                     # double
      1.3208849398808329
    .text
11
12
    .globl main
    .p2align 4, 0x90
13
    .type main, @function
14
15 main:
      @main
     .cfi_startproc
17 # %bb.0:
18
      kill: def $edi killed $edi def $rdi
19
    # annotation: \$-10 = int
20
    #NO_APP
21
    #APP
    # annotation: $98 = char
23
    #NO_APP
    #APP
25
    # annotation: $-98 = unsigned int
26
27
    #NO APP
    #APP
28
    # annotation: $1 = short
29
    #NO_APP
30
31
    #APP
    # annotation: $4294967296 = long
32
    #NO_APP
33
    movss .LCPIO_0(%rip), %xmm0
                                     \# xmm0 =
     mem[0],zero,zero,zero
35
    #APP
    # annotation: %xmm0 = float
36
    #NO_APP
37
    movsd .LCPIO_1(%rip), %xmm0
                                     \# xmm0 =
     mem[0],zero
    #APP
40
    # annotation: %xmm0 = double
    #NO_APP
41
    leal
          134217625(%rdi), %eax
42
    reta
43
44 .Lfunc_end0:
    .size main, .Lfunc_end0-main
    .cfi_endproc
```

```
# -- End function
48 .ident "clang version 10.0.0-4ubuntu1"
49 .section ".note.GNU-stack","",@progbits
50 .addrsig
```

Listing 17: variable-O3.s

Each of the different types of annotations were additionally annotated using the same method. Some interesting results occur from this annotation technique. To begin with, because all variables have been optimised away, the location of the simple variables have instead been replaced with their value. For example, line 20 shows that the value of -10 was stored in an integer. This behaviour is identical to that observed by the CompCert annotations. Additionally, an interesting scenario occurs when working with floating point numbers. Although they too have been fully optimised out, because the inline asm has required reading their values, their value has been placed within a register designated for floating point arithmetic.

The next goal was to attempt preserving annotations from more complex variables. To begin with, a volatile, global variable will be annotated. The test file volatile.c was experimented on.

```
volatile int x;

int main() {
   asm("# annotation: %0 = High" : :
   "X"(x));
   return x + 1;
}
```

Listing 18: volatile.c

On line 4, an asm statement has been inserted, containing the same format and information as with *variable.c.* However, the variable x is instead a volatile variable. The annotated assembly with full optimisation is listed below.

```
8 # %bb.0:
    movl x(%rip), %eax
    #APP
    # annotation: %eax = High
    #NO APP
    movl
          x(%rip), %eax
13
14
    addl
          $1, %eax
    reta
  .Lfunc_end0:
    .size main, .Lfunc_end0-main
17
18
    .cfi_endproc
      -- End function
     .type x,@object
                                     # @x
    .comm x,4,4
21
            "clang version 10.0.0-4ubuntu1 "
22
23
    .section
               ".note.GNU-stack","",@progbits
    .addrsig
24
    .addrsig_sym x
```

Listing 19: volatile-O3.s

Line 11 successfully shows the annotation referencing the global variable x. However, an additional move statement has been inserted on line 9. This additional move statement was quite puzzling, however, it is inserted due to the nature of the inline asm. As the compiler does not know or understand the assembly created by the programmer, additional move statements may need to be inserted by the compiler. By working backwards, the location of x can be located within memory at the location x(%rip). Because the constraint 'X' has been provided, any operand whatsoever is permitted. As a result, the compiler chose a register as the solution to fill the annotation's requirements. As such, an unnecessary move statement has been inserted. In such a case, the programmer should observe the result and update the asm restraint to only allow memory locations. Doing so results in the preferred behaviour with only the annotation inserted into the assembly.

This form of inline asm was used to test annotations across the remaining test files. The annotated assembly can be viewed in Appendix .6.

### Quality Analysis

Although the annotated assembly does contain additional move statements, the program still behaves as expected. However, it does result in heightened difficulty to parse the true location of the variable.

For a large program with many annotations, systematically identifying where each variable is stored and modifying the constraints until move statements are eliminated is time consuming and impractical. Therefore, an analysis of the program with the additional move statements preserved will be performed. One proposed method of handling this issue is to build a tool to assist in this parsing and analysis. This tool is covered in section 0.4.2.

To analyse the affect of these additional move statements inserted, big O notation will be used. From the comprehensive testing done, no compiler optimisations were removed when annotations were addeded. Thus, for this analysis, it will be assumed that no optimisations will be reverted when annotations are added. Let A(n) be the function describing the annotated assembly, and B(n) be the function describing the non-annotated assembly. Then,

$$B(n) \in \Theta(h(n))$$

As move statements are constant, any move statement is an element of  $\Theta(1)$ . Thus, for m annotations added to the program,

$$A(n,m) \in \Theta(m \cdot h(n))$$

As can be seen, in the worst case, the program runtime increases linearly with each annotation added. For time critical or concurrent programs, this is unacceptable. Ideally, a static analysis should be possible without slowing the program for each annotation added. The added move statements are only used to perform the analysis and to identify the location of variables within the program. Therefore, one should be able to compile the program with and without annotations. Assuming all optimisations are performed with annotations turned on, an analysis conducted on a program will still be valid when annotations are turned off. As such, a tool can be created to assist in this annotation and analysis process.

### **Tool Assisted Annotations**

To allow for the program to compile without annotations, the annotations need to be stored where they

cannot affect the program output. Littering the program with extended asm statements results in a difficult and tedious process of removing them once the program is ready to be compiled. Instead, annotations can be written by the programmer using inline comments. As these comments will be ignored by the compiler, the program can be compiled on any compiler supporting their C version. To reference a variable within the annotation, the variable name can be wrapped in a var keyword. An example of an annotated file can be seen below in pread.c.

```
volatile int z;
volatile int x;
4 int main() {
      // security policies:
      // annotation: L(var(z)) = true
      // annotation: L(var(x)) = var(z) \% 2
      // annotation: var(x) < var(x);</pre>
      int r1 = 0;
9
      int r2 = 0;
      // annotation: _P_0: var(r1) % 2 == 0
      // annotation: _Gamma_0: var(r1) ->
LOW, var(r2) -> LOW
12
      // annotation: L(var(r2)) = false
13
14
      while(1) {
16
               // annotation: _invariant:
17
      var(r1) % 2 == 0 /\ var(r1) <= z
               // annotation: _Gamma: var(r1)
      -> LOW, var(r2) -> (var(r1) == var(z)),
      var(z) \rightarrow LOW
               do {
19
                    // annotation: _invariant:
20
      var(r1) \le var(z)
                    // annotation: _Gamma:
      var(r1) -> LOW
                    r1 = z:
               } while (r1 %2 != 0);
23
               r2 = x;
24
           } while (z != r1);
25
26
      return r2:
27
28 }
```

Listing 20: pread.c

Annotations have been listed with an inline comment beginning with // annotation:. Following the declaration of a variable, the annotation can be listed in whatever format the programmer prefers. Line 6 shows the security policy for the variable

z, denoting that its security policy is always high. Whereas, on line 7, the security policy of the variable x is dependent on the value of variable z. Also shown in this example are all the necessary annotations required for a wpif analysis.

The goal of the tool is to preserve these annotations stored in these comments and compile to a format that allows for

1. Install specifications

Tool building:

- 1. escaping special strings
- 2. Source Code documentation

### 0.4.3 CompCert Builtin Annotations

TODO: Revisit compact using builtin annoations rather than AIS annotations

### 0.4.4 LLVM Compiler Modification

the final technique of modifying the LLVM compiler was not experimented on. This was primarily due to two reasons. To begin with, the primary objective of this thesis is to explore techniques that do not modify the compiler, and instead work alongside the functionality of the compiler to preserve annotations. It is well known and documented that modifying the compiler to preserve annotations is possible and successful, as in the case of Vu et al. [30] Additionally, earlier success through the technique of using inline assembly allowed for more time to be allocated to exploring and improving this technique, as seen in 0.4.2. Therefore, evaluating compiler modification for static analysis purposes was not performed in this research.

# **Bibliography**

- [1] Pieter Agten et al. "Recent developments in low-level software security". In: *IFIP Interna*tional Workshop on Information Security Theory and Practice. Springer. 2012, pp. 1–16.
- [2] Musard Balliu. "Logics for information flow security: from specification to verification". PhD thesis. KTH Royal Institute of Technology, 2014.
- [3] Nick Benton. "Simple relational correctness proofs for static analyses and program transformations". In: *ACM SIGPLAN Notices* 39.1 (2004), pp. 14–25.
- [4] CompCert The CompCert C compiler. Accessed: 2020-09-01. 2020. URL: http://compcert.inria.fr/compcert-C.html.
- [5] Compiler Optimization Removal or Modification of Security-critical Code. Accessed: 2020-09-01. 2008. URL: https://cwe.mitre.org/ data/definitions/733.html.
- [6] Compiler Removal of Code to Clear Buffers. Accessed: 2020-09-01. 2006. URL: https:// cwe.mitre.org/data/definitions/14. html.
- [7] Vijay D'Silva, Mathias Payer, and Dawn Song. "The correctness-security gap in compiler optimization". In: 2015 IEEE Security and Privacy Workshops. IEEE. 2015, pp. 73–87.
- [8] Dorothy E Denning. "A lattice model of secure information flow". In: Communications of the ACM 19.5 (1976), pp. 236–243.
- [9] Dorothy E Denning and Peter J Denning. "Certification of programs for secure information flow". In: Communications of the ACM 20.7 (1977), pp. 504–513.

- [10] Quang Do, Ben Martini, and Kim-Kwang Raymond Choo. "The role of the adversary model in applied security research". In: Computers & Security 81 (2019), pp. 156–181.
- [11] K Eder, K Georgiou, and N Grech. Common Assertion Language. ENTRA Project: Whole-Systems Energy Transparency (FET project 318337). Deliverable 2.1. 2013.
- [12] Gidon Ernst and Toby Murray. "SecCSL: Security concurrent separation logic". In: International Conference on Computer Aided Verification. Springer. 2019, pp. 208–230.
- [13] File:Simplified Heartbleed explanation.svg Wikimedia Commons. Accessed: 2020-09-02. 2014. URL: https://commons.wikimedia.org/wiki/File:Simplified\_Heartbleed\_explanation.svg # mediaviewer / File:Simplified\_Heartbleed\_explanation.svg.
- [14] GCC Developer Options. Accessed: 2020-09-02. URL: https://gcc.gnu.org/onlinedocs/gcc/Developer-Options.html.
- [15] Barbara Guttman and Edward A Roback. An introduction to computer security: the NIST handbook. Diane Publishing, 1995.
- [16] Heartbleed Bug. Accessed: 2020-09-02. 2020. URL: https://heartbleed.com/.
- [17] Xavier Leroy. "Formal certification of a compiler back-end or: programming a compiler with a proof assistant". In: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2006, pp. 42–54.
- [18] Xavier Leroy et al. "CompCert-a formally verified optimizing compiler". In: 2016.

- [19] Heiko Mantel, Matthias Perner, and Jens Sauer. "Noninterference under weak memory models". In: 2014 IEEE 27th Computer Security Foundations Symposium. IEEE. 2014, pp. 80–94.
- [20] Heiko Mantel, David Sands, and Henning Sudbrock. "Assumptions and guarantees for compositional noninterference". In: 2011 IEEE 24th Computer Security Foundations Symposium. IEEE. 2011, pp. 218–232.
- [21] Toby Murray, Robert Sison, and Kai Engelhardt. "COVERN: A logic for compositional verification of information flow control". In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE. 2018, pp. 16–30.
- [22] Options That Control Optimization. Accessed: 2020-09-01. URL: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html.
- [23] Sundeep Prakash, Yann-Hang Lee, and Theodore Johnson. "Non-blocking algorithms for concurrent data structures". MA thesis. Citeseer, 1991.
- [24] Andrei Sabelfeld and Andrew C Myers. "Language-based information-flow security". In: *IEEE Journal on selected areas in communications* 21.1 (2003), pp. 5–19.
- [25] Bernhard Schommer et al. "Embedded program annotations for WCET analysis". In: 18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.
- [26] Laurent Simon, David Chisnall, and Ross Anderson. "What you get is what you C: Controlling side effects in mainstream C compilers". In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE. 2018, pp. 1–15.
- [27] Graeme Smith, Nicholas Coughlin, and Toby Murray. "Value-Dependent Information-Flow Security on Weak Memory Models". In: International Symposium on Formal Methods. Springer. 2019, pp. 539–555.

- [28] William Stallings et al. Computer security: principles and practice. Pearson Education Upper Saddle River, NJ, USA, 2012.
- [29] Jeffrey A Vaughan and Todd Millstein. "Secure information flow for concurrent programs under total store order". In: 2012 IEEE 25th Computer Security Foundations Symposium. IEEE. 2012, pp. 19–29.
- [30] Son Tuan Vu et al. "Secure delivery of program properties through optimizing compilation". In: *Proceedings of the 29th International Conference on Compiler Construction*. 2020, pp. 14–26.
- [31] Kirsten Winter, Graeme Smith, and Nicholas Coughlin. "Information flow security in the presence of fine-grained concurrency". Unpublished. Aug. 2020.

# Appendices

# .1 Test C Programs

### .1.1 comment.c

```
int main() {
    // Critical Comment
    return 0;
4 }
```

### .1.2 variable.c

```
int main(int argc, char* argv[]) {
       // a = int
       // b = char
       // c = unsigned int
       // d = short
// e = long
 5
 6
       // x = float
     // y = double
int a = -10;
char b = 'b';
unsigned int c = -b;
9
10
11
      short d = 0x1;
12
    long e = 4294967296;
    float x = 3.141592653589793;
double y = x / 2.3784;
return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int) y + argc;
14
15
16
```

### .1.3 volatile.c

### .1.4 loop.c

```
int z;
int x;

// security policies
// {L(z)=true}
// {L(x)=z % 2 == 0}

// predicates on initial state
// {_P_0: r1 % 2 == 0}
// {_P_0: r1 % 2 == 0}
// {_Gamma_0: r1 -> LOW, r2 -> LOW}

int main() {
    int r1 = 0;
    // {L(r2)=False}
    int r2 = 0;
```

```
while(1) {
17
18
            // {_invariant: r1 % 2 == 0 /\ r1 <= z}
19
             // \{ Gamma: r1 \rightarrow LOW, r2 \rightarrow (r1 == z), z \rightarrow LOW \}
20
            do {
21
                 // {_invariant: r1 <= z}
// {_Gamma: r1 -> LOW}
22
23
                 r1 = z;
24
            } while (r1 %2 != 0);
                 r2 = x;
26
            } while (z != r1);
27
28
29
       return r2;
30 }
```

### .1.5 rooster.c

```
1 int rooster;
2 int drake;
3 // MEDIUM
4 int goose;
6 int fun(int a, int b, int c) {
      // CRITICAL COMMENT
      static int count = 0;
      int sum = a + b + c;
9
      if (sum < 0) {</pre>
10
          return sum;
12
      if (a < b && b < c) {</pre>
13
          while (a != b) {
14
15
               a++;
               count++;
16
17
               while (b != c) {
                   c--;
18
                   count++;
19
               }
20
           }
21
22
      return count;
23
24 }
25
26 int main(void) {
27
      // EXCEPTIONAL
      rooster = 1;
28
      drake = 5;
29
     goose = 10;
30
      int result;
31
32
      result = fun(rooster, drake, goose);
      return 0;
33
34 }
```

### .1.6 password.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
10
     user_t* next;
      char name[BUFF_LEN];
11
     // L(password) = High
     char password[BUFF_LEN];
13
      size_t balance;
14
15 };
16
17 user_t* setup_users() {
     user_t* user_admin = malloc(sizeof(user_t));
18
      strcpy(user_admin->name, "admin");
19
      strcpy(user_admin->password, "4dm1n__4eva");
20
     user_admin->balance = 1000000;
21
22
      user_t* user_alice = malloc(sizeof(user_t));
23
      strcpy(user_alice->name, "alice");
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
     user_alice->balance = 783;
26
27
      user_t* user_abdul = malloc(sizeof(user_t));
28
29
      strcpy(user_abdul->name, "abdul");
      strcpy(user_abdul ->password, "passw0rd123");
30
     user_abdul -> balance = 2;
31
32
      user_admin->next = user_alice;
33
34
      user_alice->next = user_abdul;
      user_abdul ->next = NULL;
35
37
      return user_admin;
38 }
39
40 void print_users(user_t* users) {
     printf("--- USERS ---\n");
      size_t count = 0;
42
43
      while (users != NULL) {
          printf(" %02ld. %s\n", ++count, users->name);
44
          users = users->next;
45
46
      printf("\n");
47
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
      while (user_list != NULL) {
          if (strcmp(user_list->name, name) == 0) {
52
53
              return user_list;
54
55
          user_list = user_list->next;
56
      return NULL;
57
58 }
60 int main() {
```

```
user_t* users = setup_users();
61
      printf("Welcome to BigBank Australia!\n");
63
64
      char username[BUFF_LEN];
65
      printf("Username: ");
66
      scanf("%255s", username);
67
68
69
      user_t* user = getUser(users, username);
      if (user == NULL) {
70
          printf("User < %s > does not exist.\n", username);
71
72
          return 0;
73
74
      char password[BUFF_LEN];
75
      printf("Password: ");
76
      scanf("%255s", password);
77
      if (strcmp(user->password, password) != 0) {
78
79
          printf("ERROR: incorrect password\n");
          return 0;
80
81
82
      printf("Logged in as < %s >!\n", user->name);
83
84
      printf("\n");
      printf("Welcome, %s!\n", user->name);
85
      printf("Your balance: $%ld\n", user->balance);
86
87 }
```

### .1.7 deadStoreElimination.c

```
int deadStore(int i, int n) {
      int key = 0xabcd;
      // L(key) = high
      // do some work
5
     int result = 0;
6
      while (i > n) {
          result += key;
8
9
10
11
      // clear out our secret key
12
      key = 0;
13
14
      return i + n;
15 }
16
int main(int argc, char *argv[]) {
      deadStore(argc, 2);
18
19 }
```

### .1.8 pread.c

```
volatile int z;
volatile int x;

// security policies
// {L(z)=true}
```

```
6 // \{L(x)=z \% 2 == 0\}
8 // predicates on initial state
9 // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
     int r1 = 0;
13
      // {L(r2)=False}
14
      int r2 = 0;
16
      while(1) {
17
18
      do {
           // {_invariant: r1 % 2 == 0 /\ r1 <= z}
19
           // \{ Gamma: r1 \rightarrow LOW, r2 \rightarrow (r1 == z), z \rightarrow LOW \}
20
21
               // {_invariant: r1 <= z}
22
               // {_Gamma: r1 -> LOW}
23
               r1 = z;
           } while (r1 %2 != 0);
25
               r2 = x;
           } while (z != r1);
27
28
29
      return r2;
```

# .2 CompCert Annotated C Programs

### .2.1 comment.c

```
int main() {
    __builtin_ais_annot("%here Critical Comment");
    return 0;
4 }
```

### .2.2 variable.c

```
int main(int argc, char* argv[]) {
    int a = -10;
      __builtin_ais_annot("%here %e1 = int", a);
     char b = 'b';
      __builtin_ais_annot("%here %e1 = char", b);
     unsigned int c = -b;
6
     __builtin_ais_annot("%here %e1 = unsigned int", c);
     short d = 0x1;
8
      __builtin_ais_annot("%here %e1 = short", d);
     long e = 4294967296;
10
      __builtin_ais_annot("%here %e1 = long", e);
11
     float x = 3.141592653589793;
12
     __builtin_ais_annot("%here %e1 = float", x);
13
14
     double y = x / 2.3784;
      __builtin_ais_annot("%here %e1 = double", y);
15
16
      return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int) y + argc;
```

### .2.3 volatile.c

```
volatile int x;

int main() {
    __builtin_ais_annot("%here L(%e1)= false", x);
    return x + 1;
6 }
```

### .2.4 loop.c

```
2 int z;
3 int x;
5 int main() {
      // Security Policies
      __builtin_ais_annot("%here L(%e1) = true", z);
      __builtin_ais_annot("%here L(%e1)= %e2 %% 2 == 0", x, z);
9
      int r1 = 0;
10
      int r2 = 0;
11
      __builtin_ais_annot("%here L(%e1) = false", r2);
12
     // Predicates on initial state
13
     __builtin_ais_annot("%here _P_0: %e1 %% 2 == 0", r1);
__builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1, r2);
14
15
16
      while(1) {
17
18
          __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1 <= %e2", r1, z);
19
           __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1 == %e3), %e3 -> LOW",
20
      r1, r2, z);
          do {
21
               __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
22
               __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
23
24
               r1 = z;
           } while (r1 %2 != 0);
25
              r2 = x;
26
           } while (z != r1);
      }
28
      return r2;
29
30 }
```

### .2.5 rooster.c

```
int rooster;
2 int drake;
3 int goose;
5 int fun(int a, int b, int c) {
      __builtin_ais_annot("%here CRITICAL COMMENT");
      static int count = 0;
      int sum = a + b + c;
     if (sum < 0) {</pre>
9
10
          return sum;
11
12
     if (a < b && b < c) {</pre>
         while (a != b) {
13
          a++;
```

```
count++;
               while (b != c) {
16
                   c--;
17
                   count++;
18
               }
19
          }
20
21
      return count;
22
23 }
24
25 int main(void) {
      __builtin_ais_annot("%here L(%e1) = medium", goose);
26
       __builtin_ais_annot("%here EXCEPTIONAL");
27
28
      rooster = 1;
      drake = 5;
29
      goose = 10;
30
31
      int result;
      result = fun(rooster, drake, goose);
32
33
      return 0;
34 }
```

### .2.6 password.c

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
     user_t* next;
      char name[BUFF_LEN];
11
                                       //
                                                     { L(password) = High }
     char password[BUFF_LEN];
12
13
     size_t balance;
14 };
15
16 user_t* setup_users() {
17
     user_t* user_admin = malloc(sizeof(user_t));
      strcpy(user_admin->name, "admin");
18
      strcpy(user_admin->password, "4dm1n__4eva");
19
      __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
20
      user_admin->balance = 1000000;
21
22
      user_t* user_alice = malloc(sizeof(user_t));
23
      strcpy(user_alice->name, "alice");
24
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
      __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
26
      user_alice->balance = 783;
27
28
      user_t* user_abdul = malloc(sizeof(user_t));
29
      strcpy(user_abdul->name, "abdul");
30
      strcpy(user_abdul->password, "passw0rd123");
31
      __builtin_ais_annot("%here L(%e1) = high", user_abdul->password);
32
      user_abdul->balance = 2;
33
```

```
user_admin->next = user_alice;
35
      user_alice->next = user_abdul;
      user_abdul ->next = NULL;
37
38
      return user_admin;
39
40 }
41
42 void print_users(user_t* users) {
      printf("--- USERS ---\n");
      size_t count = 0;
44
      while (users != NULL) {
45
          printf(" %02ld. %s\n", ++count, users->name);
46
          users = users->next;
47
48
      printf("\n");
49
50 }
51
52 user_t* getUser(user_t* user_list, char* name) {
      while (user_list != NULL) {
          if (strcmp(user_list->name, name) == 0) {
54
55
               return user_list;
56
57
          user_list = user_list->next;
58
      return NULL;
59
60 }
61
62 int main() {
      user_t* users = setup_users();
63
64
65
      printf("Welcome to BigBank Australia!\n");
66
      char username[BUFF_LEN];
67
      printf("Username: ");
68
      scanf("%255s", username);
69
70
      user_t* user = getUser(users, username);
71
72
      if (user == NULL) {
          printf("User < %s > does not exist.\n", username);
73
74
          return 0;
75
76
      char password[BUFF_LEN];
77
      printf("Password: ");
78
      scanf("%255s", password);
79
      if (strcmp(user->password, password) != 0) {
80
          printf("ERROR: incorrect password\n");
81
82
          return 0;
83
      printf("Logged in as < %s >! \n", user->name);
85
      printf("\n");
86
      printf("Welcome, %s!\n", user->name);
87
      printf("Your balance: $%ld\n", user->balance);
88
89 }
```

### .2.7 deadStoreElimination.c

```
int deadStore(int i, int n) {
      int key = 0xabcd;
      __builtin_ais_annot("%here L(%e1) = high", key);
3
      // do some work
5
      int result = 0;
6
      while (i > n) {
         result += key;
          i--;
     }
10
11
     // clear out our secret key
12
     key = 0;
13
14
     return i + n;
15 }
16
int main(int argc, char *argv[]) {
     deadStore(argc, 2);
18
19 }
```

### .2.8 pread.c

```
volatile int z;
volatile int x;
4 int main() {
      // Security Policies
      __builtin_ais_annot("%here L(%e1) = true", z);
      __builtin_ais_annot("%here L(%e1)= %e2 %% 2 == 0", x, z);
8
9
      int r1 = 0;
     int r2 = 0;
                             //
                                             {L(r2)=False}
10
11
      __builtin_ais_annot("%here L(%e1)= false", r2);
12
      // Predicates on initial state
13
      __builtin_ais_annot("%here _P_0: %e1 %% 2 == 0", r1);
__builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1, r2);
14
15
16
17
      while(1) {
18
19
          do {
               __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1 <= %e2", r1, z);
20
               __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1 == %e3), %e3 ->
21
      LOW", r1, r2, z);
22
               do {
23
                   __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
                    __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
24
25
                   r1 = z;
               } while (r1 %2 != 0);
26
                   r2 = x;
27
           } while (z != r1);
28
29
30
      return r2;
31 }
```

# .3 CompCert Assembly Output

### .3.1 comment-O0.s

```
# File generated by CompCert 3.7
# Command line: comment.c -S -00 -o compCert/out/comment-00.s
.text
.align 16
.globl main
main:
.cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
xorl %eax, %eax
addq $8, %rsp
ret
.cfi_endproc
.type main, @function
.size main, - main
```

### .3.2 comment-O3.s

```
# File generated by CompCert 3.7
2 # Command line: comment.c -S -03 -o compCert/out/comment-03.s
   .text
align 16globl main
6 main:
   .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
12 xorl %eax, %eax
   addq $8, %rsp
13
.cfi_endproc
16 .type main, @function
.size main, . - main
```

### .3.3 variable-O0.s

```
# File generated by CompCert 3.7

# Command line: variable.c -S -00 -o compCert/out/variable-00.s

.text

align 16
.globl main

main:

.cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
movl $-10, %esi
```

```
movl $98, %ecx
13
    negl %ecx
movl $1, %r8d
14
15
    movabsq $4294967296, %rax
16
    movsd .L100(%rip), %xmm1 # 3.14159265358979312
17
    cvtsd2ss %xmm1, %xmm3
cvtss2sd %xmm3, %xmm0
18
19
    movsd .L101(%rip), %xmm2 # 2.3784000000000007
20
    divsd %xmm2, %xmm0
22
    cqto
    shrq $59, %rdx
leaq 0(%rax, %rdx, 1), %rax
23
24
    sarq $5, %rax
25
    leal 0(%eax, %esi, 1), %r9d
    leal 0(%r9d,%ecx,1), %r10d
leal 0(%r10d,%r8d,1), %r8d
27
28
29
    cvttss2si %xmm3, %edx
    leal 0(%r8d,%edx,1), %r8d
30
    cvttsd2si %xmm0, %eax
    leal 0(%r8d,%eax,1), %r11d
leal 0(%r11d,%edi,1), %eax
32
    addq $8, %rsp
34
    ret
35
    .cfi_endproc
36
    .type main, Ofunction
37
    .size main, . - main
    .section .rodata.cst8, "aM", @progbits,8
39
   .align 8
41 .L100: .quad 0x400921fb54442d18
42 .L101: .quad 0x400306f694467382
```

### .3.4 variable-O3.s

```
# File generated by CompCert 3.7
2 # Command line: variable.c -S -03 -o compCert/out/variable-03.s
    .text
   .align 16
   .globl main
5
6 main:
   .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
leal 134217625(%edi), %eax
10
11
12
    addq $8, %rsp
13
14
    ret
    .cfi_endproc
15
    .type main, Ofunction
.size main, . - main
```

### .3.5 volatile-O0.s

```
# File generated by CompCert 3.7

# Command line: volatile.c -S -00 -o compCert/out/volatile-00.s

.comm x, 4, 4

.text
```

```
5 .align 16
6 .globl main
7 main:
   .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10
11
11 leaq 16(%rsp), %rax
12 movq %rax, 0(%rsp)
movl x(\%rip), \%eax
   leal 1(%eax), %eax
addq $8, %rsp
14
15
16
    ret
   .cfi_endproc
17
18 .type main, Ofunction
19 .size main, . - main
```

### .3.6 volatile-O3.s

```
# File generated by CompCert 3.7
 2 # Command line: volatile.c -S -03 -o compCert/out/volatile-03.s
    .comm x, 4, 4
    .text
 5 .align 16
 6 .globl main
 7 main:
8 .cfi_startproc
subq $8, %rsp
    .cfi_adjust_cfa_offset 8
10
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
movl x(%rip), %eax
    leal 1(%eax), %eax
    addq $8, %rsp
15
16
    ret
    .cfi_endproc
17
   .type main, @function
19 .size main, . - main
```

### .3.7 loop-O0.s

```
# File generated by CompCert 3.7
2 # Command line: loop.c -S -00 -o compCert/out/loop-00.s
    .comm z, 4, 4
    .comm x, 4, 4
   .text
   .align 16
7 .globl main
8 main:
9 .cfi_startproc
10 subq $8, %rsp
   .cfi_adjust_cfa_offset 8
11
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:
movl z(%rip), %edx
movq %rdx, %rax
testl %eax, %eax
```

```
18 leal 1(%eax), %ecx
19
    cmovl %rcx, %rax
    sarl $1, %eax
20
    leal 0(,%eax,2), %esi
21
   movq %rdx, %rcx
subl %esi, %ecx
testl %ecx, %ecx
22
23
24
    jne .L100
25
movl z(%rip), %esi
27 jmp .L100
    .cfi_endproc
28
    .type main, @function
30 .size main, . - main
```

# .3.8 loop-O3.s

```
# File generated by CompCert 3.7
 2 # Command line: loop.c -S -03 -o compCert/out/loop-03.s
    .comm z, 4, 4
    .comm x, 4, 4
    .text
 6 .align 16
 7 .globl main
8 main:
    .cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:
movl z(%rip), %edx
movq %rdx, %rax
17 testl %eax, %eax
   leal 1(%eax), %ecx
cmovl %rcx, %rax
18
19
    sarl $1, %eax
20
21
    leal 0(,%eax,2), %esi
movq %rdx, %rcx
subl %esi, %ecx
testl %ecx, %ecx
    jne .L100
25
   movq %rdx, %rsi
27 jmp .L100
    .cfi_endproc
    .type main, @function
30 .size main, . - main
```

#### .3.9 rooster-O0.s

```
# File generated by CompCert 3.7

# Command line: rooster.c -S -00 -o compCert/out/rooster-00.s

.comm rooster, 4, 4

.comm drake, 4, 4

.comm goose, 4, 4

.data

.align 4

count:
```

```
9 .long 0
    .type count, @object
    .size count, . - count
11
12
13 .align 16
.globl fun
15 fun:
.cfi_startproc
17
    subq $8, %rsp
18
    .cfi_adjust_cfa_offset 8
    leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
leal 0(%edi,%esi,1), %r8d
19
20
21
    leal 0(%r8d, %edx, 1), %eax
    testl %eax, %eax
23
    jl .L100
cmpl %esi, %edi
24
25
    jl .L101
26
27 xorl %r8d, %r8d
28 jmp .L102
29 .L101:
30 cmpl %edx, %esi
31 setl %r8b
movzbl %r8b, %r8d
33 .L102:
34 cmpl $0, %r8d
35 je .L103
36 .L104:
37 cmpl %esi, %edi
je .L103
39 leal 1(%edi), %edi
40 movl count(%rip), %eax
    leal 1(%eax), %ecx
movl %ecx, count(%rip)
43 .L105:
44 cmpl %edx, %esi
    je .L104
45
    leal -1(\%edx), \%edx
   movl count(%rip), %r9d leal 1(%r9d), %r8d movl %r8d, count(%rip)
47
48
49
50 jmp .L105
51 .L103:
52 movl
          count(%rip), %eax
53 .L100:
54 addq $8, %rsp
55
    ret
56
    .cfi_endproc
57
    .type fun, @function
58
    .size fun, . - fun
59
    .text
60
   .align 16
61 .globl main
62 main:
63 .cfi_startproc
64 subq $8, %rsp
65 .cfi_adjust_cfa_offset 8
```

```
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
movl $1, %eax
66
68
    movl %eax, rooster(%rip)
69
    movl $5, %eax
70
    movl %eax, drake(%rip)
movl $10, %eax
71
72
    movl %eax, goose(%rip)
73
    movl rooster(%rip), %edi
    movl drake(%rip), %esi
movl goose(%rip), %edx
call fun
75
76
77
    xorl %eax, %eax
78
79
    addq $8, %rsp
80
    ret
    .cfi_endproc
81
82
    .type main, @function
ssize main, . - main
```

## .3.10 rooster-O3.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -S -03 -o compCert/out/rooster-03.s
   .comm rooster, 4, 4
   .comm drake, 4, 4
5 .comm goose, 4, 4
  .data
7 .align 4
8 count:
9 .long 0
   .type count, @object
11
   .size count, . - count
    .text
   .align 16
13
   .globl fun
14
15 fun:
   .cfi_startproc
16
17
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
18
19
   leaq 16(%rsp), %rax
   movq %rax, 0(%rsp)
20
    leal 0(%edi,%esi,1), %r9d
leal 0(%r9d,%edx,1), %eax
21
    testl %eax, %eax
23
    jl .L100
24
   cmpl %edx, %esi
setl %al
25
26
    movzbl %al, %eax
27
    xorl %r8d, %r8d
28
    cmpl %esi, %edi
    cmovge %r8, %rax
30
    cmpl $0, %eax
31
32
    je .L101
33 .L102:
34 cmpl %esi, %edi
35 je .L101
16 leal 1(%edi), %edi
```

```
movl count(%rip), %ecx
leal 1(%ecx), %r8d
movl %r8d, count(%rip)
40 .L103:
41 cmpl %edx, %esi
42
43
    je .L102
    leal -1(%edx), %edx
44 movl count(%rip), %r10d
45 leal 1(%r10d), %r8d
movl %r8d, count(%rip)
47 jmp .L103
48 .L101:
movl count(%rip), %eax
50 .L100:
addq $8, %rsp
52
53
    .cfi_endproc
    .type fun, @function
54
55
    .size fun, . - fun
    .text
56
.align 16
.globl main
59 main:
60 .cfi_startproc
    subq $8, %rsp
61
    .cfi_adjust_cfa_offset 8
63
    leaq 16(%rsp), %rax
63 leaq 10(%1sp), %1ax
64 movq %rax, 0(%rsp)
65 movl $1, %eax
66 movl %eax, rooster(%rip)
67 movl $5, %eax
68 movl %eax, drake(%rip)
    movl $10, %eax
69
70
    movl %eax, goose(%rip)
    movl $1, %edi
movl $5, %esi
71
72
    movl $10, %edx
73
    call fun
    xorl %eax, %eax
75
76
    addq $8, %rsp
77
    ret
78
    .cfi_endproc
.type main, Ofunction
80 .size main, . - main
```

# .3.11 password-O0.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -00 -o compCert/out/password-00.s
3    .section    .rodata
4    .align 1
5    __stringlit_7:
6    .ascii "--- USERS ---\012\000"
7    .type __stringlit_7, @object
8    .size __stringlit_7, . - __stringlit_7
9    .section    .rodata
10    .align 1
```

```
11 __stringlit_6:
   .ascii "passw0rd123\000"
   .type __stringlit_6, @object
13
   .size __stringlit_6, . - __stringlit_6
14
15 .section .rodata
16 .align 1
17 __stringlit_4:
   .ascii "!alice12!_veuje@@hak\000"
18
   .type __stringlit_4, @object
20
   .size __stringlit_4, . - __stringlit_4
21
   .section .rodata
22
   .align 1
23 __stringlit_14:
.ascii "Password: \000"
   .type __stringlit_14, @object
25
   .size __stringlit_14, . - __stringlit_14
26
27
   .section .rodata
   .align 1
28
29 __stringlit_18:
30 .ascii "Your balance: $%1d\012\000"
   .type __stringlit_18, @object
.size __stringlit_18, . - __stringlit_18
32
   .section .rodata
33
34 .align 1
35 __stringlit_13:
   .ascii "User < %s > does not exist.\012\000"
   .type __stringlit_13, @object
37
   .size __stringlit_13, . - __stringlit_13
38
39
   .section .rodata
   .align 1
40
41 __stringlit_8:
   .ascii " %02ld. %s\012\000"
42
   .type __stringlit_8, @object
44
   .size __stringlit_8, . - __stringlit_8
   .section .rodata
45
46
    .align 1
47 __stringlit_1:
48 .ascii "admin\000"
   .type __stringlit_1, @object
49
   .size __stringlit_1, . - __stringlit_1
50
   .section .rodata
51
52
   .align 1
53 __stringlit_2:
.ascii "4dm1n_4eva\000"
   .type __stringlit_2, @object
55
56
   .size __stringlit_2, . - __stringlit_2
57
   .section .rodata
ss .align 1
59 __stringlit_3:
   .ascii "alice\000"
   .type __stringlit_3, @object
61
   .size __stringlit_3, . - __stringlit_3
62
63 .section .rodata
64 .align 1
65 __stringlit_11:
   .ascii "Username: \000"
.type __stringlit_11, @object
```

```
.size __stringlit_11, . - __stringlit_11
    .section .rodata
    .align 1
70
71 __stringlit_5:
^{72} .ascii "abdul\000"
    .type __stringlit_5, @object
73
74
    .size __stringlit_5, . - __stringlit_5
    .section .rodata
75
    .align 1
77 __stringlit_17:
    .ascii "Welcome, %s!\012\000"
78
79
    .type __stringlit_17, @object
    .size __stringlit_17, . - __stringlit_17
80
    .section .rodata
82
   .align 1
83 __stringlit_12:
    .ascii "%255s\000"
84
    .type __stringlit_12, @object
85
.size __stringlit_12, . - __stringlit_12
87
    .section .rodata
    .align 1
89 __stringlit_9:
90 .ascii "\012\000"
    .type __stringlit_9, @object
    .size __stringlit_9, . - __stringlit_9
92
93
    .section .rodata
    .align 1
94
95 __stringlit_15:
96 .ascii "ERROR: incorrect password\012\000"
    .type __stringlit_15, @object
97
    .size __stringlit_15, . - __stringlit_15
99
    .section .rodata
    .align 1
101 __stringlit_10:
    .ascii "Welcome to BigBank Australia!\012\000"
102
103
    .type __stringlit_10, @object
.size __stringlit_10, . - __stringlit_10
    .section .rodata
    .align 1
106
107 __stringlit_16:
    .ascii "Logged in as < %s >!\012\000"
108
    .type __stringlit_16, @object
109
110
    .size __stringlit_16, . - __stringlit_16
    .text
111
    .align 16
.globl setup_users
112
113
114 setup_users:
115
    .cfi_startproc
    subq $40, %rsp
116
    .cfi_adjust_cfa_offset 40
117
    leaq 48(%rsp), %rax
118
    movq %rax, 0(%rsp)
119
120
    movq %rbx, 8(%rsp)
    movq %rbp, 16(%rsp)
movq %r12, 24(%rsp)
121
122
movq $528, %rdi
124 call malloc
```

```
movq %rax, %rbp
125
126
     leaq 8(%rbp), %rdi
     leaq
            __stringlit_1(%rip), %rsi
127
     call
            strcpy
128
     leaq 264(%rbp), %rdi
129
     leaq __stringlit_2(%rip), %rsi
call strcpy
130
131
     call
            strcpy
     movq $1000000, %rax
132
     movq %rax, 520(%rbp)
133
     movq $528, %rdi
134
     call malloc
movq %rax, %r12
135
136
     leaq 8(%r12), %rdi
137
138
     leaq __stringlit_3(%rip), %rsi
     call
            strcpy
139
     leaq 264(%r12), %rdi
140
141
     leaq __stringlit_4(%rip), %rsi
     call strcpy
142
143
     movq $783, %rsi
     movq %rsi, 520(%r12)
movq $528, %rdi
call malloc
144
145
146
     movq %rax, %rbx
147
148
     leaq 8(%rbx), %rdi
     leaq __stringlit_5(%rip), %rsi
call strcpy
149
150
     call
            strcpy
     leaq 264(%rbx), %rdi
151
     leaq __stringlit_6(%rip), %rsi
call strcpy
152
153
     movq $2, %r10
movq %r10, 520(%rb
movq %r12, 0(%rbp)
154
155
            %r10, 520(%rbx)
156
     movq %rbx, 0(%r12)
157
     xorq %r8, %r8
158
     movq %r8, 0(%rbx)
movq %rbp, %rax
159
160
     movq 8(%rsp), %rbx
161
162
     movq 16(%rsp), %rbp
     movq 24(%rsp), %r12
163
164
     addq $40, %rsp
165
     ret
     .cfi_endproc
166
     .type setup_users, @function
     .size setup_users, . - setup_users
168
169
     .text
     .align 16
170
     .globl print_users
171
172 print_users:
    .cfi_startproc
173
174
     subq $24, %rsp
     .cfi_adjust_cfa_offset 24
175
     leaq 32(%rsp), %rax
176
177
     movq %rax, 0(%rsp)
     movq %rbx, 8(%rsp)
178
179
     movq
            %rbp , 16(%rsp)
    movq %rdi, %rbx
180
leaq __stringlit_7(%rip), %rdi
```

```
movl $0, %eax
     call
            printf
xorq %rbp, %rbp
185 .L100:
186 cmpq $0, %rbx
     je .L101
187
     leaq 1(%rbp), %rbp
188
     leaq __stringlit_8(%rip), %rdi
leaq 8(%rbx), %rdx
189
190
    movq %rbp, %rsi
movl $0, %eax
call printf
movq 0(%rbx), %rbx
191
192
193
194
195 jmp .L100
196 .L101:
     leaq __stringlit_9(%rip), %rdi
movl $0, %eax
197
198
     call printf
199
     movq 8(%rsp), %rbx
     movq 16(%rsp), %rbp
addq $24, %rsp
201
202
203
     ret
     .cfi_endproc
204
     .type print_users, @function
205
     .size print_users, . - print_users
206
207
     .text
     .align 16
208
.globl getUser
210 getUser:
211
     .cfi_startproc
212
     subq $24, %rsp
     .cfi_adjust_cfa_offset 24
213
     leaq 32(%rsp), %rax
214
215 movq %rax, 0(%rsp)
216 movq %rbx, 8(%rsp)
217 movq %rbp, 16(%rsp)
218 movq %rsi, %rbp
219 movq %rdi, %rbx
220 .L102:
221
     cmpq $0, %rbx
     je .L103
222
     leaq 8(%rbx), %rdi
223
movq %rbp, %rsi
     call strcmp
225
     testl %eax, %eax
226
227
     je .L104
228 movq 0(%rbx), %rbx
jmp .L102
230 .L103:
xorq %rbx, %rbx
232 .L104:
movq %rbx, %rax
    movq 8(%rsp), %rbx
234
235
     movq 16(%rsp), %rbp
addq $24, %rsp
236
237 ret
.cfi_endproc
```

```
.type getUser, @function
     .size getUser, . - getUser
241
     .text
    .align 16
242
    .globl main
243
244 main:
245
    .cfi_startproc
     subq $536, %rsp
246
247
     .cfi_adjust_cfa_offset 536
     leaq 544(%rsp), %rax
248
     movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
249
250
     call
           setup_users
251
252
     movq %rax, %rbx
     leaq
            __stringlit_10(%rip), %rdi
253
     movl
           $0, %eax
254
255
     call printf
     leaq __stringlit_11(%rip), %rdi
256
257
     movl $0, %eax
     call printf
258
259
     leaq
            __stringlit_12(%rip), %rdi
     leaq 16(%rsp), %rsi
260
     movl $0, %eax
261
     call __isoc99_scanf
262
     leaq 16(%rsp), %rsi
movq %rbx, %rdi
263
264
            getUser
     call
265
     movq %rax, %rbx
266
     cmpq $0, %rbx
267
     jne .L105
268
     leaq __stringlit_13(%rip), %rdi
leaq 16(%rsp), %rsi
269
270
     movl $0, %eax
271
     call printf
xorl %eax, %eax
272
273
274
     jmp .L106
275 .L105:
276 leaq __stringlit_14(%rip), %rdi
277 mov1 $0, %eax
278
     call printf
            __stringlit_12(%rip), %rdi
279
     leaq
            272(%rsp), %rsi
     leaq
280
281
     movl $0, %eax
     call __isoc99_scanf
leaq 264(%rbx), %rdi
282
283
     leaq 272(%rsp), %rsi
284
     call strcmp
285
     testl %eax, %eax
286
287
     je .L107
     leaq __stringlit_15(%rip), %rdi
movl $0, %eax
288
289
     call printf
290
    xorl %eax, %eax
291
    jmp .L106
292
293 .L107:
294 leaq
            __stringlit_16(%rip), %rdi
295 leaq 8(%rbx), %rsi
```

```
movl $0, %eax
296
     call
            printf
     leaq
            __stringlit_9(%rip), %rdi
298
     movl $0, %eax
299
     call printf
300
     leaq __stringlit_17(%rip), %rdi
leaq 8(%rbx), %rsi
301
302
     movl $0, %eax
303
     call printf
    leaq __stringlit_18(%rip), %rdi
movq 520(%rbx), %rsi
movl $0, %eax
305
306
307
     call printf
308
309 xorl %eax, %eax
310 .L106:
    movq 8(%rsp), %rbx addq $536, %rsp
311
312
313 ret
314 .cfi_endproc
.type main, @function
316 .size main, . - main
```

## .3.12 password-O3.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -03 -o compCert/out/password-03.s
   .section .rodata
    .align 1
5 __stringlit_7:
6 .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
   .section .rodata
   .align 1
10
11 __stringlit_6:
.ascii "passw0rd123\000"
   .type __stringlit_6, @object
13
14
    .size __stringlit_6, . - __stringlit_6
   .section .rodata
15
   .align 1
17 __stringlit_4:
   .ascii "!alice12!_veuje@@hak\000"
18
.type __stringlit_4, @object
.size __stringlit_4, . - __stringlit_4
21 .section .rodata
   .align 1
22
23 __stringlit_14:
   .ascii "Password: \000"
   .type __stringlit_14, @object
25
   .size __stringlit_14, . - __stringlit_14
27
   .section .rodata
   .align 1
28
29 __stringlit_18:
.ascii "Your balance: $%ld\012\000"
.type __stringlit_18, @object
.size __stringlit_18, . - __stringlit_18
33 .section .rodata
```

```
34 .align 1
35 __stringlit_13:
   .ascii "User < %s > does not exist.\012\000"
   .type __stringlit_13, @object
   .size __stringlit_13, . - __stringlit_13
38
section .rodata
40
   .align 1
41 __stringlit_8:
.ascii " %021d. %s\012\000"
   .type __stringlit_8, @object
43
44
   .size __stringlit_8, . - __stringlit_8
   .section .rodata
45
   .align 1
46
47 __stringlit_1:
48 .ascii "admin\000"
   .type __stringlit_1, @object
49
50
   .size __stringlit_1, . - __stringlit_1
   .section .rodata
51
52 .align 1
53 __stringlit_2:
   .ascii "4dm1n_4eva\\000"
   .type __stringlit_2, @object
55
   .size __stringlit_2, . - __stringlit_2
56
57 .section .rodata
58
   .align 1
59 __stringlit_3:
   .ascii "alice\000"
60
   .type __stringlit_3, @object
61
.size __stringlit_3, . - __stringlit_3
   .section .rodata
63
   .align 1
65 __stringlit_11:
66 .ascii "Username: \000"
   .type __stringlit_11, @object
67
   .size __stringlit_11, . - __stringlit_11
68
69
   .section .rodata
   .align 1
70
71 __stringlit_5:
_{72} .ascii "abdul\000"
   .type __stringlit_5, @object
73
   .size __stringlit_5, . - __stringlit_5
74
75
   .section .rodata
76
   .align 1
77 __stringlit_17:
   .ascii "Welcome, %s!\012\000"
   .type __stringlit_17, @object
79
so .size __stringlit_17, . - __stringlit_17
81 .section .rodata
82 .align 1
83 __stringlit_12:
   .ascii "%255s\000"
84
   .type __stringlit_12, @object
85
86
   .size __stringlit_12, . - __stringlit_12
87 .section .rodata
   .align 1
89 __stringlit_9:
90 .ascii "\012\000"
```

```
.type __stringlit_9, @object
     .size __stringlit_9, . - __stringlit_9
    .section .rodata
93
    .align 1
95 __stringlit_15:
    .ascii "ERROR: incorrect password\012\000"
96
     .type __stringlit_15, @object
    .size __stringlit_15, . - __stringlit_15
98
    .section .rodata
    .align 1
100
101 __stringlit_10:
     .ascii "Welcome to BigBank Australia!\012\000"
102
    .type __stringlit_10, @object
103
     .size __stringlit_10, . - __stringlit_10
     .section .rodata
105
     .align 1
106
107 __stringlit_16:
.ascii "Logged in as < %s >!\012\000"
109
    .type __stringlit_16, @object
     .size __stringlit_16, . - __stringlit_16
110
111
    .align 16
112
    .globl setup_users
113
114 setup_users:
    .cfi_startproc
115
116
     subq $40, %rsp
     .cfi_adjust_cfa_offset 40
117
     leaq 48(%rsp), %rax
118
     movq %rax, 0(%rsp)
119
    movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
movq %r12, 24(%rsp)
120
121
122
     movq $528, %rdi
123
     call malloc
124
     movq %rax, %rbp
leaq 8(%rbp), %rdi
125
126
     leaq __stringlit_1(%rip), %rsi
127
128
     call strcpy
     leaq 264(%rbp), %rdi
129
     leaq __stringlit_2(%rip), %rsi
call strcpy
130
131
     movq $1000000, %rax
132
133
     movq %rax, 520(%rbp)
     movq $528, %rdi
134
     call
           malloc
135
     movq %rax, %r12
136
     leaq 8(%r12), %rdi
137
    leaq __stringlit_3(%rip), %rsi
call strcpy
leaq 264(%r12), %rdi
138
139
     leaq __stringlit_4(%rip), %rsi
141
     call strcpy
142
     movq $783, %rsi
143
     movq %rsi, 520(%r12)
movq $528, %rdi
144
145
    call malloc
146
movq %rax, %rbx
```

```
leaq 8(%rbx), %rdi
148
149
     leaq
            __stringlit_5(%rip), %rsi
     call
            strcpy
150
     leaq 264(%rbx), %rdi
151
     leaq __stringlit_6(%rip), %rsi
152
     call
            strcpy
153
     movq $2, %r10
154
     movq %r10, 520(%rbx)
155
     movq %r12, 0(%rbp)
156
     movq %rbx, 0(%r12)
157
     xorq %r8, %r8
movq %r8, 0(%rbx)
158
159
     movq %rbp, %rax
160
     movq 8(%rsp), %rbx
161
     movq 16(%rsp), %rbp
movq 24(%rsp), %r12
addq $40, %rsp
162
163
164
     ret
165
166
     .cfi_endproc
     .type setup_users, @function
167
     .size setup_users, . - setup_users
     .text
169
    .align 16
170
.globl print_users
172 print_users:
173
     .cfi_startproc
     subq $24, %rsp
174
     .cfi_adjust_cfa_offset 24
175
     leaq 32(%rsp), %rax
176
    movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
177
178
179
     movq %rdi, %rbp
180
     leaq
            __stringlit_7(%rip), %rdi
181
     movl $0, %eax
182
183
     call
            printf
    xorq %rbx, %rbx
184
185 .L100:
    cmpq $0, %rbp
186
187
     je .L101
     leaq 1(%rbx), %rbx
188
     leaq
            __stringlit_8(%rip), %rdi
189
     leaq 8(%rbp), %rdx
190
    movq %rbx, %rsi
movl $0, %eax
call printf
191
192
193
    movq 0(%rbp), %rbp
194
195 jmp .L100
196 .L101:
     leaq __stringlit_9(%rip), %rdi
movl $0, %eax
197
     leaq
198
     call printf
199
     movq 8(%rsp), %rbx
200
     movq 16(%rsp), %rbp
addq $24, %rsp
201
202
203
    ret
cfi_endproc
```

```
.type print_users, @function
     .size print_users, . - print_users
207
     .text
    .align 16
208
    .globl getUser
209
210 getUser:
211
     .cfi_startproc
     subq $24, %rsp
212
    .cfi_adjust_cfa_offset 24
213
    leaq 32(%rsp), %rax
214
    movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
215
216
217
     movq %rsi, %rbp
218
    movq %rdi, %rbx
219
220 .L102:
    cmpq $0, %rbx
221
     je .L103
222
223 leaq 8(%rbx), %rdi
    movq %rbp, %rsi
call strcmp
224
     testl %eax, %eax
226
    je .L104
227
228 movq 0(%rbx), %rbx
    jmp .L102
229
230 .L103:
xorq %rbx, %rbx
232 .L104:
movq %rbx, %rax
     movq 8(%rsp), %rbx
234
235
     movq
          16(%rsp), %rbp
     addq $24, %rsp
236
237
     .cfi_endproc
238
     .type getUser, @function
239
240
     .size getUser, . - getUser
    .text
241
    .align 16
    .globl main
243
244 main:
245
    .cfi_startproc
246
     subq $536, %rsp
     .cfi_adjust_cfa_offset 536
    leaq 544(%rsp), %rax
248
     movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
249
250
     call setup_users
251
252
     movq %rax, %rbx
     leaq __stringlit_10(%rip), %rdi
movl $0, %eax
253
254
     call printf
255
     leaq
           __stringlit_11(%rip), %rdi
256
     movl $0, %eax
257
     call printf
258
__stringlit_12(%rip), %rdi
261 movl $0, %eax
```

```
call __isoc99_scanf
262
263
     leaq 16(%rsp), %rsi
     movq %rbx, %rdi
264
     call
           getUser
265
     movq %rax, %rbx
266
     cmpq $0, %rbx
267
268
     jne .L105
     leaq __stringlit_13(%rip), %rdi
269
    leaq 16(%rsp), %rsi
270
     movl $0, %eax
271
    call printf
xorl %eax, %eax
272
273
    jmp .L106
274
275 .L105:
          __stringlit_14(%rip), %rdi
$0, %eax
    leaq
276
    movl
277
    call printf
278
    leaq __stringlit_12(%rip), %rdi
279
    leaq 272(%rsp), %rsi
     movl $0, %eax
281
     call
           __isoc99_scanf
    leaq 264(%rbx), %rdi
283
    leaq 272(%rsp), %rsi
284
285
     call strcmp
    testl %eax, %eax
286
287
     je .L107
    leaq __stringlit_15(%rip), %rdi
288
    movl $0, %eax
289
    call printf
xorl %eax, %eax
290
291
292
    jmp .L106
293 .L107:
           __stringlit_16(%rip), %rdi
294
    leaq
    leaq 8(%rbx), %rsi
295
    movl $0, %eax
296
297
     call printf
    leaq
           __stringlit_9(%rip), %rdi
298
299
     movl $0, %eax
     call printf
300
    leaq __stringlit_17(%rip), %rdi
leaq 8(%rbx), %rsi
301
302
     movl $0, %eax
303
304
     call printf
    leaq __stringlit_18(%rip), %rdi
305
           520(%rbx), %rsi
306
    movq
     movl $0, %eax
307
    call printf
308
    xorl %eax, %eax
309
310 .L106:
    movq
           8(%rsp), %rbx
312
    addq $536, %rsp
    ret
313
    .cfi_endproc
314
    .type main, @function
315
316 .size main, . - main
```

## .3.13 deadStoreElimination-O0.s

```
# File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -00 -o compCert/out/deadStoreElimination-00.s
    .text
   .align 16
   .globl deadStore
6 deadStore:
    .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
cmpl %esi, %edi
   jle .L101
15 leal -1(%edi), %edi
    jmp .L100
16
17 .L101:
18 leal 0(%edi,%esi,1), %eax
19
   addq $8, %rsp
   ret
20
    .cfi_endproc
    .type deadStore, @function
22
   .size deadStore, . - deadStore
23
    .text
   .align 16
25
    .globl main
26
27 main:
   .cfi_startproc
28
    subq $8, %rsp
29
    .cfi_adjust_cfa_offset 8
30
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
32
    movl $2, %esi
33
   call deadStore
xorl %eax, %eax
addq $8, %rsp
34
35
36
37
    ret
   .cfi_endproc
   .type main, @function
39
40 .size main, . - main
```

## .3.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7

# Command line: deadStoreElimination.c -S -03 -o compCert/out/deadStoreElimination-03.s

.text

.align 16
.globl deadStore
deadStore:
.cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
.L100:
cmpl %esi, %edi
jle .L101
```

```
15 leal -1(%edi), %edi
   jmp .L100
17 .L101:
18 leal 0(%edi,%esi,1), %eax
   addq $8, %rsp
19
   ret
20
21
    .cfi_endproc
   .type deadStore, @function
22
   .size deadStore, . - deadStore
24
   .text
   .align 16
25
   .globl main
26
27 main:
28 .cfi_startproc
   subq $8, %rsp
29
    .cfi_adjust_cfa_offset 8
30
16(%rsp), %rax
16(%rsp), %rax
12 movq %rax, 0(%rsp)
33
   movl $2, %esi
   call deadStore
xorl %eax, %eax
34
35
    addq $8, %rsp
36
37
    ret
38 .cfi_endproc
   .type main, @function
39
40 .size main, . - main
```

# .3.15 pread-O0.s

```
# File generated by CompCert 3.7
2 # Command line: pread.c -S -00 -o compCert/out/pread-00.s
3 .comm z, 4, 4
    .comm x, 4, 4
   .text
   .align 16
7 .globl main
8 main:
    .cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:
movl z(%rip), %edx
movq %rdx, %rsi
movq %rsi, %rax
   testl %eax, %eax leal 1(%eax), %ecx
18
19
   cmovl %rcx, %rax
20
    sarl $1, %eax
21
   leal 0(,%eax,2), %edi
subl %edi, %esi
testl %esi, %esi
22
23
24
   jne .L100
25
movl x(%rip), %esi
27 movl z(%rip), %esi
28 jmp .L100
```

```
29 .cfi_endproc
30 .type main, @function
31 .size main, . - main
```

# .3.16 pread-O3.s

```
# File generated by CompCert 3.7
 2 # Command line: pread.c -S -03 -o compCert/out/pread-03.s
    .comm z, 4, 4
    .comm x, 4, 4
    .text
   .align 16
 7 .globl main
8 main:
9
    .cfi_startproc
10
   subq $8, %rsp
    .cfi_adjust_cfa_offset 8
11
12 leaq 16(%rsp), %rax
   movq %rax, 0(%rsp)
13
14 .L100:
movl z(%rip), %edx
movq %rdx, %rsi
movq %rsi, %rax
   testl %eax, %eax
leal 1(%eax), %ecx
18
   cmovl %rcx, %rax
20
    sarl $1, %eax
21
22 leal 0(,%eax,2), %edi
23 subl %edi, %esi
24 testl %esi, %esi
    jne .L100
25
26 movl x(%rip), %esi
27 movl z(%rip), %esi
28 jmp .L100
    .cfi_endproc
   .type main, @function
31 .size main, . - main
```

# .4 CompCert Annotated Assembly Output

## .4.1 comment-O0.s

```
# File generated by CompCert 3.7
# Command line: comment.c -S -00 -o annotated/comment-00.s
.text
.align 16
.globl main
main:
.cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
.L100:
xorl %eax, %eax
addq $8, %rsp
```

```
ret
cfi_endproc
type main, @function
size main, . - main
section "__compcert_ais_annotations","",@note
ascii "# file:comment.c line:2 function:main\n"
byte 7,8
quad .L100
ascii " Critical Comment\n"
```

#### .4.2 comment-03.s

```
# File generated by CompCert 3.7
2 # Command line: comment.c -S -03 -o annotated/comment-03.s
   .align 16
5 .globl main
6 main:
   .cfi_startproc
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
xorl %eax, %eax
   addq $8, %rsp
14
    ret
   .cfi_endproc
16
    .type main, @function
17
    .size main, . - main
18
   .section "__compcert_ais_annotations","", @note
19
   .ascii "# file:comment.c line:2 function:main\n"
21 .byte 7,8
.quad .L100
.ascii " Critical Comment\n"
```

#### .4.3 variable-O0.s

```
# File generated by CompCert 3.7
2 # Command line: variable.c -S -00
3 .text
   .align 16
   .globl main
6 main:
7 .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
9
10
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
movl $-10, %esi
13 .L100:
14 movl $98, %ecx
15 .L101:
16 negl %ecx
17 .L102:
movl $1, %r10d
19 .L103:
```

```
20 movabsq $4294967296, %rax
21 .L104:
        movsd .L105(%rip), %xmm1 # 3.14159265358979312
22
          cvtsd2ss %xmm1, %xmm3
23
         cvtss2sd %xmm3, %xmm0
24
          movsd .L106(%rip), %xmm2 # 2.3784000000000007
25
          divsd %xmm2, %xmm0
26
          cqto
27
          shrq $59, %rdx
         leaq 0(%rax, %rdx, 1), %rax
29
         sarq $5, %rax
leal 0(%eax,%esi,1), %r9d
30
31
          leal 0(%r9d, %ecx, 1), %r8d
32
         leal 0(%r8d,%r10d,1), %r10d
          cvttss2si %xmm3, %edx
34
          leal 0(%r10d, %edx, 1), %r11d
35
36
          cvttsd2si %xmm0, %r8d
         leal 0(%r11d,%r8d,1), %ecx
37
         leal 0(%ecx, %edi, 1), %eax
          addq $8, %rsp
39
40
          ret
         .cfi_endproc
41
         .type main, @function
42
43
         .size main, . - main
         .section .rodata.cst8, "aM", @progbits,8
44
45
           .align 8
46 .L105: .quad 0x400921fb54442d18
47 .L106: .quad 0x400306f694467382
         .section "__compcert_ais_annotations","", @note
48
         .ascii "# file:variable.c line:3 function:main\n"
49
50
         .byte 7,8
         .quad .L100
51
         .ascii " reg(\"rsi\") = int\n"
52
          .ascii "# file:variable.c line:5 function:main\n"
53
54
         .byte 7,8
55
          .quad .L101
         .ascii " reg(\"rcx\") = char\n"
56
         .ascii "# file:variable.c line:7 function:main\n"
57
          .byte 7,8
58
59
          .quad .L102
          .ascii " reg(\"rcx\") = unsigned int\n"
60
         .ascii "# file:variable.c line:9 function:main\n"
61
         .byte 7,8
62
          .quad .L103
63
          .ascii " reg(\"r10\") = short\n"
          .ascii "# file:variable.c line:11 function:main\n"
65
         .byte 7,8
66
         .quad .L104
.ascii " reg(\running (\running \running \runnning \running \running \running \runnning \running \running \running \ru
```

# .4.4 variable-O3.s

```
# File generated by CompCert 3.7
# Command line: variable.c -S -03
text
align 16
glob1 main
```

```
6 main:
    .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
   movq %rax, 0(%rsp)
11
12 .L100:
13 .L101:
14 .L102:
15 .L103:
16 .L104:
17 leal 134217625(%edi), %eax
   addq $8, %rsp
18
19
   ret
   .cfi_endproc
20
   .type main, @function
21
22
   .size main, . - main
   .section "__compcert_ais_annotations","", @note
23
   .ascii "# file:variable.c line:3 function:main\n"
   .byte 7,8
25
   .quad .L100
26
   .ascii " -10 = int \n"
27
   .ascii "# file:variable.c line:5 function:main\n"
28
29
   .byte 7,8
   .quad .L101
30
    .ascii " 98 = char n"
31
   .ascii "# file:variable.c line:7 function:main\n"
32
   .byte 7,8
33
   .quad .L102
34
   .ascii " -98 = unsigned int\n"
35
   .ascii "# file:variable.c line:9 function:main\n"
36
   .byte 7,8
37
   .quad .L103
38
   .ascii " 1 = short n"
39
   .ascii "# file:variable.c line:11 function:main\n"
40
41
   .byte 7,8
42 .quad .L104
.ascii " 4294967296 = long \n"
```

# .4.5 volatile-O0.s

```
volatile.c:4: error: access to volatile variable 'x' for parameter '%e1' is not supported
in ais annotations
1 error detected.
```

# .4.6 volatile-O3.s

```
volatile.c:4: error: access to volatile variable 'x' for parameter '%e1' is not supported
in ais annotations
1 error detected.
```

## .4.7 loop-O0.s

```
# File generated by CompCert 3.7

# Command line: loop.c -S -00 -o annotated/loop-00.s

.comm z, 4, 4

.comm x, 4, 4
```

```
5 .text
align 16globl main
8 main:
9 .cfi_startproc
subq $8, %rsp
11
    .cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
14 .L100:
15 .L101:
16 xorl %edx, %edx
xorl %edi, %edi
18 .L102:
19 .L103:
20 .L104:
21 nop
22 .L105:
23 .L106:
24 .L107:
25 nop 26 .L108:
27 .L109:
28 .L110:
movl z(%rip), %edx
movq %rdx, %rax
    testl %eax, %eax
31
    leal 1(%eax), %ecx
32
    cmovl %rcx, %rax
33
    sarl $1, %eax
leal 0(,%eax,2), %edi
34
35
    movq %rdx, %rcx subl %edi, %ecx
36
37
    testl %ecx, %ecx
38
    jne .L108
39
    movl x(%rip), %edi
40
    movl z(%rip), %esi
41
42
    jmp .L105
    .cfi_endproc
43
44
    .type main, Ofunction
    .size main, . - main
45
    .section "__compcert_ais_annotations","", @note
46
    .ascii "# file:loop.c line:7 function:main\n"
47
    .byte 7,8
48
    .quad .L100
49
    .ascii " L(mem("
50
    .byte 7,8
51
52
    .quad z
    .ascii ", 4)) = true\n"
.ascii "# file:loop.c line:8 function:main\n"
53
    .byte 7,8
55
    .quad .L101
56
    .ascii " L(mem("
57
    .byte 7,8
58
59
    .quad x
    .ascii ", 4))= mem("
60
61 .byte 7,8
```

```
.quad z
62
     .ascii ", 4) \% 2 == 0 \n"
    .ascii "# file:loop.c line:11 function:main\n"
64
    .byte 7,8
    .quad .L102
66
    .ascii " L(reg(\"rdi\"))= false\n"
67
     .ascii "# file:loop.c line:14 function:main\n"
68
    .byte 7,8
69
    .quad .L103
    .ascii " _P_0: reg(\"rdx\") % 2 == 0\n"
.ascii "# file:loop.c line:15 function:main\n"
71
72
73
    .byte 7,8
    .quad .L104
74
    .ascii " _Gamma_0: reg(\"rdx\") -> LOW, reg(\"rdi\") -> LOW\n"
    .ascii "# file:loop.c line:19 function:main\n"
76
    .byte 7,8
77
    .quad .L106
78
    .ascii " _invariant: reg(\"rdx\") % 2 == 0 & reg(\"rdx\") <= mem("
79
    .byte 7,8
    .quad z
81
     .ascii ", 4)\n"
    .ascii "# file:loop.c line:20 function:main\n"
83
    .byte 7,8
84
    .quad .L107
85
    .ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\") == mem("
86
     .byte 7,8
87
    .quad z
88
    .ascii ", 4)), mem("
89
    .byte 7,8
90
    .quad z
91
     .ascii ", 4) -> LOW\n"
92
    .ascii "# file:loop.c line:22 function:main\n"
93
    .byte 7,8
    .quad .L109
95
    .ascii " _invariant: reg(\"rdx\") <= mem("
96
97
     .byte 7,8
    .quad z
98
    .ascii ", 4)\n"
    .ascii "# file:loop.c line:23 function:main\n"
100
101
    .byte 7,8
102
    .quad .L110
.ascii " _Gamma: reg(\"rdx\") -> LOW\n"
```

#### .4.8 loop-O3.s

```
# File generated by CompCert 3.7
2 # Command line: loop.c -S -03 -o annotated/loop-03.s
.comm z, 4, 4
.comm x, 4, 4
.text
.align 16
.globl main
main:
.cfi_startproc
subq $8, %rsp
.cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
```

```
movq %rax, 0(%rsp)
14 .L100:
15 .L101:
16 xorl %edx, %edx
17 xorl %edi, %edi
18 .L102:
19 .L103:
20 .L104:
21 nop
22 .L105:
23 .L106:
24 .L107:
25 nop
26 .L108:
27 .L109:
28 .L110:
29 movl z(\%rip), \%edx
movq %rdx, %rax
testl %eax, %eax
   leal 1(%eax), %ecx
cmovl %rcx, %rax
32
33
    sarl $1, %eax
34
    leal 0(,%eax,2), %edi
35
    movq %rdx, %rcx
subl %edi, %ecx
testl %ecx, %ecx
36
37
38
    jne .L108
39
    movl x(%rip), %edi
40
    movq %rdx, %rsi
41
    jmp .L105
42
43
    .cfi_endproc
    .type main, @function
44
    .size main, . - main
45
    .section "__compcert_ais_annotations","", @note
46
    .ascii "# file:loop.c line:7 function:main\n"
47
    .byte 7,8
48
    .quad .L100
49
50
    .ascii " L(mem("
    .byte 7,8
51
52
    .quad z
    .ascii ", 4)) = true\n"
53
    .ascii "# file:loop.c line:8 function:main\n"
54
55
    .byte 7,8
    .quad .L101
56
    .ascii " L(mem("
57
    .byte 7,8
58
    .quad x
59
    .ascii ", 4))= mem("
60
    .byte 7,8
61
62
    .quad z
    .ascii ", 4) % 2 == 0\n"
63
    .ascii "# file:loop.c line:11 function:main\n"
64
    .byte 7,8
65
    .quad .L102
66
    .ascii " L(0) = false \n"
    .ascii "# file:loop.c line:14 function:main\n"
68
69 .byte 7,8
```

```
.quad .L103
70
    .ascii " _P_0: 0 % 2 == 0\n"
.ascii "# file:loop.c line:15 function:main\n"
72
    .byte 7,8
73
    .quad .L104
74
    .ascii " _Gamma_0: 0 -> LOW, 0 -> LOW\n"
.ascii "# file:loop.c line:19 function:main\n"
75
76
    .byte 7,8
77
    .quad .L106
    .ascii " _invariant: reg(\"rdx\") % 2 == 0 & reg(\"rdx\") <= mem("
79
     .byte 7,8
80
81
     .quad z
    .ascii ", 4)\n"
82
    .ascii "# file:loop.c line:20 function:main\n"
    .byte 7,8
84
    .quad .L107
85
     .ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\") == mem("
86
    .byte 7,8
87
    .quad z
     .ascii ", 4)), mem("
89
     .byte 7,8
    .quad z
91
    .ascii ", 4) -> LOW\n"
92
    .ascii "# file:loop.c line:22 function:main\n"
93
    .byte 7,8
94
95
     .quad .L109
    .ascii " _invariant: reg(\"rdx\") <= mem("
96
    .byte 7,8
97
98
    .quad z
    .ascii ", 4)\n"
99
    .ascii "# file:loop.c line:23 function:main\n"
    .byte 7,8
101
    .quad .L110
.ascii " _Gamma: reg(\"rdx\") -> LOW\n"
```

# .4.9 rooster-O0.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -S -00 -o annotated/rooster-00.s
   .comm rooster, 4, 4
   .comm drake, 4, 4
   .comm goose, 4, 4
   .data
   .align 4
8 count:
   .long 0
9
   .type count, @object
10
11
    .size count, . - count
   .text
12
   .align 16
.globl fun
15 fun:
16
   .cfi_startproc
17 subq $8, %rsp
.cfi_adjust_cfa_offset 8
19 leaq 16(%rsp), %rax
20 movq %rax, 0(%rsp)
```

```
21 .L100:
   leal 0(%edi,%esi,1), %r8d
leal 0(%r8d,%edx,1), %eax
23
    testl %eax, %eax
25
    jl .L101
26
    cmpl %esi, %edi
27
    jl .L102
28 xorl %r8d, %r8d
29 jmp .L103
30 .L102:
cmpl %edx, %esi setl %r8b
movzbl %r8b, %r8d
34 .L103:
35 cmpl $0, %r8d
36
    je .L104
37 .L105:
38 cmpl %esi, %edi
39
    je .L104
    leal 1(%edi), %edi
movl count(%rip), %eax
leal 1(%eax), %ecx
40
41
42
43
   movl %ecx, count(%rip)
44 .L106:
cmpl %edx, %esi
46  je .L105
47  leal -1(%edx), %edx
movl count(%rip), %r9d
49 leal 1(%r9d), %r8d
50 movl %r8d, count(%rip)
50 movl %r80
51 jmp .L106
52 .L104:
53 movl
          count(%rip), %eax
54 .L101:
55
   addq $8, %rsp
56
    ret
    .cfi_endproc
57
58
    .type fun, @function
    .size fun, . - fun
59
60
    .text
61
    .align 16
62 .globl main
63 main:
64 .cfi_startproc
    subq $8, %rsp
65
    .cfi_adjust_cfa_offset 8
66
67 leaq 16(%rsp), %rax
68 movq %rax, 0(%rsp)
69 .L107:
70 .L108:
_{71} movl $1, %eax
    movl %eax, rooster(%rip)
72
    movl $5, %eax
73
   movl %eax, drake(%rip)
movl $10, %eax
74
75
movl %eax, goose(%rip)
movl rooster(%rip), %edi
```

```
movl drake(%rip), %esi
78
    movl goos
          goose(%rip), %edx
80
    xorl %eax, %eax
81
    addq $8, %rsp
82
    ret
83
84
    .cfi_endproc
    .type main, @function
85
    .size main, . - main
    .section "__compcert_ais_annotations","", @note
87
    .ascii "# file:rooster.c line:6 function:fun\n"
88
    .byte 7,8
89
    .quad .L100
90
    .ascii " CRITICAL COMMENT\n"
    .ascii "# file:rooster.c line:26 function:main\n"
92
    .byte 7,8
93
    .quad .L107
94
    .ascii " L(mem("
95
96
    .byte 7,8
    .quad goose
97
    .ascii ", 4)) = medium \n"
    .ascii "# file:rooster.c line:27 function:main\n"
99
    .byte 7,8
100
    .quad .L108
.ascii " EXCEPTIONAL\n"
```

#### .4.10 rooster-O3.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -S -03 -o annotated/rooster-03.s
3 .comm rooster, 4, 4
   .comm drake, 4, 4
   .comm goose, 4, 4
   .data
6
   .align 4
8 count:
   .long 0
9
    .type count, @object
   .size count, . - count
11
12
   .text
   .align 16
13
   .globl fun
14
15 fun:
.cfi_startproc
   subq $8, %rsp
17
   .cfi_adjust_cfa_offset 8
18
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
19
20
21 .L100:
leal 0(%edi,%esi,1), %r9d
   leal 0(%r9d,%edx,1), %eax
23
    testl %eax, %eax
24
    jl .L101
25
cmpl %edx, %esi
27 setl %al
28 movzbl %al, %eax
29 xorl %r8d, %r8d
```

```
30 cmpl %esi, %edi
    cmovge %r8, %rax
cmpl $0, %eax
32
   je .L102
33
34 .L103:
35
   cmpl %esi, %edi
36
    je .L102
37 leal 1(%edi), %edi
movl count(%rip), %ecx
   leal 1(%ecx), %r8d
movl %r8d, count(%rip)
39
40
41 .L104:
   cmpl %edx, %esi
42
43
    je .L103
   leal -1(%edx), %edx
movl count(%rip), %r10d
leal 1(%r10d), %r8d
44
45
46
47 movl %r8d, count(%rip)
48 jmp .L104
49 .L102:
movl count(%rip), %eax
51 .L101:
52 addq $8, %rsp
53
    ret
    .cfi_endproc
54
55
    .type fun, @function
    .size fun, . - fun
56
    .text
    .align 16
58
    .globl main
59
60 main:
61 .cfi_startproc
    subq $8, %rsp
62
    .cfi_adjust_cfa_offset 8
63
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
64
65
66 .L105:
67 .L106:
    movl $1, %eax
68
69
    movl
          %eax , rooster(%rip)
    movl $5, %eax
70
    movl %eax, drake(%rip)
movl $10, %eax
71
72
    movl %eax, goose(%rip)
73
    movl
          $1, %edi
74
    movl $5, %esi
75
    movl $10, %edx
76
77
    call fun
    xorl %eax, %eax
78
    addq $8, %rsp
79
80
    ret
    .cfi_endproc
81
    .type main, @function
82
    .size main, . - main
83
    .section "__compcert_ais_annotations","", @note
    .ascii "# file:rooster.c line:6 function:fun\n"
85
86 .byte 7,8
```

```
.quad .L100
87
    .ascii " CRITICAL COMMENT\n"
   .ascii "# file:rooster.c line:26 function:main\n"
89
   .byte 7,8
   .quad .L105
91
   .ascii " L(mem("
92
   .byte 7,8
93
   .quad goose
94
   .ascii ", 4)) = medium \n"
   .ascii "# file:rooster.c line:27 function:main\n"
96
97
   .byte 7,8
   .quad .L106
99 .ascii " EXCEPTIONAL\n"
```

# .4.11 password-O0.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -00 -o annotated/password-00.s
   .section .rodata
   .align 1
5 __stringlit_7:
6 .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
   .section .rodata
10 .align 1
11 __stringlit_6:
   .ascii "passw0rd123\000"
12
   .type __stringlit_6, @object
13
   .size __stringlit_6, . - __stringlit_6
14
.section .rodata
16 .align 1
17 __stringlit_4:
   .ascii "!alice12!_veuje@@hak\000"
18
   .type __stringlit_4, @object
19
.size __stringlit_4, . - __stringlit_4
.section .rodata
22
   .align 1
23 __stringlit_14:
24 .ascii "Password: \000"
   .type __stringlit_14, @object
   .size __stringlit_14, . - __stringlit_14
26
   .section .rodata
   .align 1
28
29 __stringlit_18:
30 .ascii "Your balance: $%ld\012\000"
   .type __stringlit_18, @object
31
32
   .size __stringlit_18, . - __stringlit_18
   .section .rodata
33
   .align 1
35 __stringlit_13:
   .ascii "User < %s > does not exist.\012\000"
36
   .type __stringlit_13, @object
.size __stringlit_13, . - __stringlit_13
39 .section .rodata
40 .align 1
41 __stringlit_8:
```

```
42 .ascii " %02ld. %s\012\000"
   .type __stringlit_8, @object
.size __stringlit_8, . - __stringlit_8
44
45 .section .rodata
46
   .align 1
47 __stringlit_1:
   .ascii "admin\000"
   .type __stringlit_1, @object
49
   .size __stringlit_1, . - __stringlit_1
   .section .rodata
51
52
   .align 1
53 __stringlit_2:
.ascii "4dm1n__4eva\000"
   .type __stringlit_2, @object
   .size __stringlit_2, . - __stringlit_2
56
57
   .section .rodata
58
    .align 1
59 __stringlit_3:
60 .ascii "alice\000"
   .type __stringlit_3, @object
61
    .size __stringlit_3, . - __stringlit_3
   .section .rodata
63
   .align 1
64
65 __stringlit_11:
.ascii "Username: \000"
   .type __stringlit_11, @object
   .size __stringlit_11, . - __stringlit_11
68
   .section .rodata
69
   .align 1
70
71 __stringlit_5:
   .ascii "abdul\000"
.type __stringlit_5, @object
72
73
   .size __stringlit_5, . - __stringlit_5
74
75
   .section .rodata
76
   .align 1
77 __stringlit_17:
   .ascii "Welcome, %s!\012\000"
78
   .type __stringlit_17, @object
   .size __stringlit_17, . - __stringlit_17
80
81
   .section .rodata
82
    .align 1
83 __stringlit_12:
84 .ascii "%255s\000"
   .type __stringlit_12, @object
85
    .size __stringlit_12, . - __stringlit_12
87
   .section .rodata
   .align 1
88
89 __stringlit_9:
   .ascii "\012\000"
90
    .type __stringlit_9, @object
   .size __stringlit_9, . - __stringlit_9
92
   .section .rodata
93
   .align 1
95 __stringlit_15:
.ascii "ERROR: incorrect password\012\000"
.type __stringlit_15, @object
size __stringlit_15, . - __stringlit_15
```

```
99 .section .rodata
    .align 1
101 __stringlit_10:
.ascii "Welcome to BigBank Australia!\012\000"
    .type __stringlit_10, @object
103
    .size __stringlit_10, . - __stringlit_10
104
105
    .section .rodata
    .align 1
106
107 __stringlit_16:
    .ascii "Logged in as < %s >!\012\000"
108
     .type __stringlit_16, @object
109
110
    .size __stringlit_16, . - __stringlit_16
    .text
111
    .align 16
.globl setup_users
114 setup_users:
115
    .cfi_startproc
     subq $40, %rsp
116
117
    .cfi_adjust_cfa_offset 40
    leaq 48(%rsp), %rax
movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
118
119
120
     movq %rbp, 16(%rsp)
121
122
     movq %r12, 24(%rsp)
    movq $528, %rdi
123
124
     call
           {\tt malloc}
    movq %rax, %rbp
125
    leaq 8(%rbp), %rdi
126
    leaq __stringlit_1(%rip), %rsi
call strcpy
127
    call strcpy
leaq 264(%rbp), %rdi
128
129
    leaq
130
           __stringlit_2(%rip), %rsi
    call
131
           strcpy
132 .L100:
    movq $1000000, %r10
movq %r10, 520(%rbp)
133
134
           %r10, 520(%rbp)
    movq $528, %rdi
135
136
     call malloc
    movq %rax, %r12
137
     leaq 8(%r12), %rdi
138
139
     leaq
           __stringlit_3(%rip), %rsi
    call strcpy
140
    leaq 264(%r12), %rdi
    leaq __stringlit_4(%rip), %rsi
142
    call
143
           strcpy
144 .L101:
movq $783, %r9
146
    movq %r9, 520(%r12)
    movq $528, %rdi
147
     call
           malloc
    movq %rax, %rbx
149
    leaq 8(%rbx), %rdi
150
151
    leaq __stringlit_5(%rip), %rsi
           strcpy
    call
152
    leaq 264(%rbx), %rdi
153
154 leaq
           __stringlit_6(%rip), %rsi
call strcpy
```

```
156 .L102:
     movq $2, %r11
     movq %r11, 520(%rbx)
158
     movq %r12, 0(%rbp)
159
     movq %rbx, 0(%r12)
160
     xorq %r8, %r8
161
            %r8, 0(%rbx)
162
     movq
     movq %rbp, %rax
163
     movq 8(%rsp), %rbx
164
     movq 16(%rsp), %rbp
movq 24(%rsp), %r12
addq $40, %rsp
165
166
167
168
     ret
     .cfi_endproc
     .type setup_users, @function
170
     .size setup_users, . - setup_users
171
172
     .text
    .align 16
173
.globl print_users
175 print_users:
     .cfi_startproc
     subq $24, %rsp
177
     .cfi_adjust_cfa_offset 24
178
179
     leaq 32(%rsp), %rax
     movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
180
181
182
     movq %rdi, %rbx
183
     leaq __stringlit_7(%rip), %rdi
movl $0, %eax
184
185
     call
            printf
    xorq %rbp, %rbp
187
188 .L103:
189 cmpq $0, %rbx
     je .L104
190
191
     leaq 1(%rbp), %rbp
     leaq __stringlit_8(%rip), %rdi
192
     leaq 8(%rbx), %rdx
193
    movq %rbp, %rsi
movl $0, %eax
call printf
194
195
196
     movq 0(%rbx), %rbx
197
    jmp .L103
199 .L104:
    leaq
            __stringlit_9(%rip), %rdi
200
     movl $0, %eax
201
     call printf
202
     movq 8(%rsp), %rbx
203
     movq 16(%rsp), %rbp
addq $24, %rsp
204
205
206
     ret
     .cfi_endproc
207
208
     .type print_users, @function
     .size print_users, . - print_users
209
210
     .text
     .align 16
211
212 .globl getUser
```

```
213 getUser:
214
    .cfi_startproc
     subq $24, %rsp
215
    .cfi_adjust_cfa_offset 24
216
    leaq 32(%rsp), %rax
217
    movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
218
219
220
movq %rsi, %rbp
movq %rdi, %rbx
223 .L105:
    cmpq $0, %rbx
224
    je .L106
225
226
    leaq 8(%rbx), %rdi
    movq %rbp, %rsi
227
    call
          strcmp
228
    testl %eax, %eax
229
    je .L107
230
movq 0(\%rbx), \%rbx
232 jmp .L105
233 .L106:
234 xorq %rbx, %rbx
235 .L107:
movq %rbx, %rax
    movq 8(%rsp), %rbx
237
238
     movq 16(%rsp), %rbp
    addq $24, %rsp
239
240
    ret
    .cfi_endproc
241
    .type getUser, @function
242
243
     .size getUser, . - getUser
244
    .text
245
    .align 16
246 .globl main
247 main:
248
    .cfi_startproc
     subq $536, %rsp
249
250
    .cfi_adjust_cfa_offset 536
    leaq 544(%rsp), %rax
251
    movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
252
253
    call setup_users
254
255
     movq %rax, %rbx
    leaq
           __stringlit_10(%rip), %rdi
256
          $0, %eax
257
     movl
     call printf
258
    leaq __stringlit_11(%rip), %rdi
259
     movl $0, %eax
260
    call printf
261
    leaq __stringlit_12
leaq 16(%rsp), %rsi
           __stringlit_12(%rip), %rdi
262
263
     movl $0, %eax
264
    call __isoc99_scanf
265
    leaq 16(%rsp), %rsi
movq %rbx, %rdi
266
267
           getUser
    call
268
269 movq %rax, %rbx
```

```
cmpq $0, %rbx
270
     jne .L108
     leaq __stringlit_13(%rip), %rdi
272
     leaq 16(%rsp), %rsi
273
     movl $0, %eax
274
     call printf
xorl %eax, %eax
275
276
     jmp .L109
277
278 .L108:
    leaq __stringlit_14(%rip), %rdi
movl $0, %eax
279
280
281
     call
           printf
           __stringlit_12(%rip), %rdi
     leaq
282
     leaq 272(%rsp), %rsi
     movl $0, %eax
284
     call
           __isoc99_scanf
285
     leaq 264(%rbx), %rdi
286
     leaq 272(%rsp), %rsi
287
288
     call strcmp
     testl %eax, %eax
289
     je .L110
     leaq __stringlit_15(%rip), %rdi
291
     movl $0, %eax
292
    call printf
xorl %eax, %eax
293
294
295
     jmp .L109
296 .L110:
           __stringlit_16(%rip), %rdi
297
    leaq
     leaq 8(%rbx), %rsi
298
     movl $0, %eax
299
     call
           printf
     leaq
           __stringlit_9(%rip), %rdi
301
     movl $0, %eax
302
303
     call printf
     leaq __stringlit_17(%rip), %rdi
leaq 8(%rbx), %rsi
304
305
     mov1 $0, %eax
306
     call printf
     leaq __stringlit_18(%rip), %rdi
308
     movq 520(%rbx
movl $0, %eax
309
           520(%rbx), %rsi
310
     call printf
311
312
    xorl %eax, %eax
313 .L109:
    movq 8(%rsp), %rbx
314
     addq $536, %rsp
315
316
     ret
317
     .cfi_endproc
     .type main, @function
318
     .size main, . - main \,
319
     .section "__compcert_ais_annotations","",@note
320
     .ascii "# file:password.c line:20 function:setup_users\n"
321
322
     .byte 7,8
     .quad .L100
323
     .ascii " L((reg(\"rbp\") + 264)) = high\n"
     .ascii "# file:password.c line:26 function:setup_users\n"
325
326 .byte 7,8
```

```
327    .quad    .L101
328    .ascii    " L((reg(\"r12\") + 264)) = high\n"
329    .ascii    "# file:password.c line:32 function:setup_users\n"
330    .byte 7,8
331    .quad    .L102
332    .ascii    " L((reg(\"rbx\") + 264)) = high\n"
```

# .4.12 password-O3.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -03 -o annotated/password-03.s
3 .section .rodata
   .align 1
5 __stringlit_7:
   .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
9 .section .rodata
10 .align 1
11 __stringlit_6:
12 .ascii "passw0rd123\000"
.type __stringlit_6, @object
   .size __stringlit_6, . - __stringlit_6
14
   .section .rodata
15
   .align 1
16
17 __stringlit_4:
.ascii "!alice12!_veuje@@hak\000"
   .type __stringlit_4, @object
.size __stringlit_4, . - __stringlit_4
19
20
   .section .rodata
21
   .align 1
23 __stringlit_14:
   .ascii "Password: \000"
   .type __stringlit_14, @object
.size __stringlit_14, . - __stringlit_14
.section .rodata
28 .align 1
29 __stringlit_18:
   .ascii "Your balance: $%ld\012\000"
30
31
   .type __stringlit_18, @object
   .size __stringlit_18, . - __stringlit_18
32
   .section .rodata
33
   .align 1
35 __stringlit_13:
.ascii "User < %s > does not exist.\012\000"
   .type __stringlit_13, @object
37
   .size __stringlit_13, . - __stringlit_13
38
   .section .rodata
39
   .align 1
40
41 __stringlit_8:
.ascii " %021d. %s\012\000"
   .type __stringlit_8, @object
43
44
   .size __stringlit_8, . - __stringlit_8
45 .section .rodata
46 .align 1
47 __stringlit_1:
48 .ascii "admin\000"
```

```
.type __stringlit_1, @object
    .size __stringlit_1, . - __stringlit_1
    .section .rodata
51
   .align 1
53 __stringlit_2:
    .ascii "4dm1n_4eva\\000"
54
    .type __stringlit_2, @object
   .size __stringlit_2, . - __stringlit_2
56
.section .rodata
58
   .align 1
59 __stringlit_3:
    .ascii "alice\000"
60
    .type __stringlit_3, @object
61
    .size __stringlit_3, . - __stringlit_3
63
    .section .rodata
    .align 1
65 __stringlit_11:
66 .ascii "Username: \000"
    .type __stringlit_11, @object
    .size __stringlit_11, . - __stringlit_11
68
    .section .rodata
    .align 1
70
71 __stringlit_5:
^{72} .ascii "abdul\000"
    .type __stringlit_5, @object
73
74
    .size __stringlit_5, . - __stringlit_5
    .section .rodata
75
76
    .align 1
77 __stringlit_17:
    .ascii "Welcome, %s!\012\000"
78
    .type __stringlit_17, @object
.size __stringlit_17, . - __stringlit_17
79
80
    .section .rodata
82
    .align 1
83 __stringlit_12:
    .ascii "%255s\000"
    .type __stringlit_12, @object
85
    .size __stringlit_12, . - __stringlit_12
    .section .rodata
87
88
    .align 1
89 __stringlit_9:
90 .ascii "\012\000"
    .type __stringlit_9, @object
    .size __stringlit_9, . - __stringlit_9
92
    .section .rodata
    .align 1
94
95 __stringlit_15:
96 .ascii "ERROR: incorrect password\012\000"
    .type __stringlit_15, @object
97
    .size __stringlit_15, . - __stringlit_15
    .section .rodata
99
    .align 1
100
101 __stringlit_10:
.ascii "Welcome to BigBank Australia!\012\000"
.type __stringlit_10, @object
.size __stringlit_10, . - __stringlit_10
.section .rodata
```

```
106 .align 1
107 __stringlit_16:
    .ascii "Logged in as < %s >!\012\000"
    .type __stringlit_16, @object
    .size __stringlit_16, . - __stringlit_16
110
111
    .text
    .align 16
112
.globl setup_users
114 setup_users:
115
    .cfi_startproc
     subq $40, %rsp
116
     .cfi_adjust_cfa_offset 40
117
    leaq 48(%rsp), %rax
118
    movq %rax, 0(%rsp)
119
    movq %rbx, 8(%rsp)
120
    movq %rbp, 16(%rsp)
movq %r12, 24(%rsp)
121
122
    movq $528, %rdi
123
124
    call malloc
    movq %rax, %rbp
125
     leaq 8(%rbp), %rdi
    leaq __stringlit_1(%rip), %rsi
127
    call strcpy
128
    leaq 264(%rbp), %rdi
129
    leaq __stringlit_2(%rip), %rsi
call strcpy
130
131
     call
           strcpy
132 .L100:
movq $1000000, %r10
    movq %r10, 520(%rbp)
134
    movq $528, %rdi
135
136
     call
           {\tt malloc}
    movq %rax, %r12
137
    leaq 8(%r12), %rdi
138
    leaq __stringlit_3(%rip), %rsi
139
    call
           strcpy
140
    leaq 264(%r12), %rdi
141
    leaq __stringlit_4(%rip), %rsi
142
143 call strcpy
144 .L101:
    movq $783, %r9
movq %r9, 520(%r12)
145
146
147
    movq $528, %rdi
    call malloc
    movq %rax, %rbx
149
     leaq 8(%rbx), %rdi
150
    leaq __stringlit_5(%rip), %rsi
call strcpy
152
153
    leaq 264(%rbx), %rdi
    leaq __stringlit_6(%rip), %rsi
call strcpy
154
155
156 .L102:
movq $2, %r11
    movq %r11, 520(%rbx)
158
    movq %r12, 0(%rbp)
movq %rbx, 0(%r12)
159
161 xorq %r8, %r8
movq %r8, 0(%rbx)
```

```
movq %rbp, %rax
163
     movq
            8(%rsp), %rbx
     movq 16(%rsp), %rbp
165
     movq 24(%rsp), %r12
166
     addq $40, %rsp
167
     ret
168
169
     .cfi_endproc
     .type setup_users, @function
170
171
     .size setup_users, . - setup_users
172
     .text
173
     .align 16
     .globl print_users
174
175 print_users:
     .cfi_startproc
     subq $24, %rsp
177
     .cfi_adjust_cfa_offset 24
178
     leaq 32(%rsp), %rax
179
     movq %rax, 0(%rsp)
180
181
     movq %rbx, 8(%rsp)
     movq %rbp, 16(%rsp)
movq %rdi, %rbp
182
183
            __stringlit_7(%rip), %rdi
184
     leaq
     movl $0, %eax
185
186
     call printf
    xorq %rbx, %rbx
187
188 .L103:
189 cmpq $0, %rbp
     je .L104
190
    leaq 1(%rbx), %rbx
191
    leaq __stringlit_8(%rip), %rdi
leaq 8(%rbp), %rdx
movq %rbx, %rsi
192
193
194
     movl $0, %eax
195
    call printf
movq 0(%rbp), %rbp
196
197
198
     jmp .L103
199 .L104:
200 leaq __stringlit_9(%rip), %rdi
201 movl $0, %eax
    call printf
movq 8(%rsp), %rbx
202
203
204
     movq 16(%rsp), %rbp
205
     addq $24, %rsp
206
     ret
     .cfi_endproc
207
     .type print_users, @function
208
    .size print_users, . - print_users
209
210
    .text
    .align 16
211
212
    .globl getUser
213 getUser:
214 .cfi_startproc
215
     subq $24, %rsp
     .cfi_adjust_cfa_offset 24
216
leaq 32(%rsp), %rax
movq %rax, 0(%rsp)
217
219 movq %rbx, 8(%rsp)
```

```
220 movq %rbp, 16(%rsp)
221 movq %rsi, %rbp
222 movq %rdi, %rbx
223 .L105:
224 cmpq $0, %rbx
     je .L106
225
     leaq 8(%rbx), %rdi
226
movq %rbp, %rsi
call strcmp
     testl %eax, %eax
229
230
     je .L107
    movq 0(%rbx), %rbx
231
232 jmp .L105
233 .L106:
xorq %rbx, %rbx
235 .L107:
movq %rbx, %rax
     movq 8(%rsp), %rbx
237
238
     movq 16(%rsp), %rbp
     addq $24, %rsp
239
240
     ret
     .cfi_endproc
241
     .type getUser, @function
242
243
     .size getUser, . - getUser
     .text
244
245
     .align 16
    .globl main
246
247 main:
    .cfi_startproc
248
     subq $536, %rsp
249
250
     .cfi_adjust_cfa_offset 536
     leaq 544(%rsp), %rax
251
     movq %rax, 0(%rsp)
252
     movq %rbx, 8(%rsp)
253
     call
           setup_users
254
255
     movq %rax, %rbx
     leaq
           __stringlit_10(%rip), %rdi
256
257
     movl $0, %eax
     call printf
258
     leaq __stringlit_11(%rip), %rdi
movl $0, %eax
259
260
     call printf
261
     leaq __stringlit_12(%rip), %rdi
262
     leaq 16(%rsp), %rsi
movl $0, %eax
263
264
     call
265
            __isoc99_scanf
     leaq 16(%rsp), %rsi
266
267
     movq %rbx, %rdi
     call getUser
movq %rax, %rbx
268
269
     cmpq $0, %rbx
270
     jne .L108
271
     leaq __stringlit_13(%rip), %rdi
272
     leaq 16(%rsp), %rsi
movl $0, %eax
273
274
    call printf
275
276 xorl %eax, %eax
```

```
jmp .L109
278 .L108:
            __stringlit_14(%rip), %rdi
    leaq
279
     movl $0, %eax
280
     call printf
281
    leaq __stringlit_12(%rip), %rdi
leaq 272(%rsp), %rsi
282
283
    movl $0, %eax
284
     call __isoc99_scanf
286
    leaq 264(%rbx), %rdi
     leaq 272(%rsp), %rsi
287
          strcmp
288
     call
    testl %eax, %eax
289
    je .L110
    leaq __stringlit_15(%rip), %rdi
291
    movl $0, %eax call printf
292
293
    xorl %eax, %eax
294
    jmp .L109
296 .L110:
    leaq
           __stringlit_16(%rip), %rdi
    leaq 8(%rbx), %rsi
298
    movl $0, %eax
299
300
     call printf
    leaq __string
movl $0, %eax
           __stringlit_9(%rip), %rdi
301
302
     call printf
303
     leaq __stringlit_17(%rip), %rdi
304
    leaq 8(%rbx), %rsi
305
     movl $0, %eax
306
307
     call
           printf
    leaq
           __stringlit_18(%rip), %rdi
308
     movq 520(%rbx), %rsi
309
310
    movl $0, %eax
           printf
311
    call
     xorl %eax, %eax
312
313 .L109:
314
    movq 8(%rsp), %rbx
     addq $536, %rsp
315
316
317
     .cfi_endproc
    .type main, @function
318
319
    .size main, . - main
     .section "__compcert_ais_annotations","", @note
320
     .ascii "# file:password.c line:20 function:setup_users\n"
321
322
     .byte 7,8
    .quad .L100
323
     .ascii " L((reg(\"rbp\") + 264)) = high\n"
324
     .ascii "# file:password.c line:26 function:setup_users\n"
325
     .byte 7,8
327
     .quad .L101
     .ascii " L((reg(\"r12\") + 264)) = high\n"
328
     .ascii "# file:password.c line:32 function:setup_users\n"
329
     .byte 7,8
330
     .quad .L102
.ascii " L((reg(\rbx\") + 264)) = high\n"
```

## .4.13 deadStoreElimination-O0.s

```
# File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -00 -o annotated/deadStoreElimination-00.s
    .align 16
   .globl deadStore
6 deadStore:
   .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
  leaq 16(%rsp), %rax
  movq %rax, 0(%rsp)
movl $43981, %ecx
11
13 .L100:
14 nop
15 .L101:
16 cmpl %esi, %edi
    jle .L102
18 leal -1(%edi), %edi
   jmp .L101
19
20 .L102:
21
   leal 0(%edi,%esi,1), %eax
    addq $8, %rsp
22
23
    ret
   .cfi_endproc
24
   .type deadStore, @function
25
    .size deadStore, . - deadStore
26
    .text
   .align 16
28
   .globl main
29
30 main:
   .cfi_startproc
31
32
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
33
    leaq 16(%rsp), %rax
34
   movq %rax, 0(%rsp)
movl $2, %esi
call deadStore
35
36
37
    xorl %eax, %eax
38
    addq $8, %rsp
40
    ret
    .cfi_endproc
41
42
    .type main, @function
    .size main, . - main
43
   .section "__compcert_ais_annotations","", @note
    .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
45
    .byte 7,8
    .quad .L100
47
.ascii " L(reg(\rcx\")) = high\n"
```

## .4.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7

# Command line: deadStoreElimination.c -S -03 -o annotated/deadStoreElimination-03.s

text

align 16
```

```
5 .globl deadStore
6 deadStore:
   .cfi_startproc
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
10
11
12 .L100:
13 nop
14 .L101:
   cmpl %esi, %edi
15
16
    jle .L102
17 leal -1(%edi), %edi
18 jmp .L101
19 .L102:
   leal 0(%edi,%esi,1), %eax
20
   addq $8, %rsp
21
   ret
22
   .cfi_endproc
    .type deadStore, @function
24
    .size deadStore, . - deadStore
25
26
   .text
   .align 16
27
28 .globl main
29 main:
    .cfi_startproc
30
    subq $8, %rsp
31
   .cfi_adjust_cfa_offset 8
32
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
movl $2, %esi
33
34
35
    call deadStore
36
    xorl %eax, %eax
37
38
    addq $8, %rsp
39
    ret
40
    .cfi_endproc
    .type main, Ofunction
41
   .size main, . - main
   .section "__compcert_ais_annotations","", @note
43
    .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
44
45
   .byte 7,8
   .quad .L100
46
.ascii " L(43981) = high n"
```

#### .4.15 pread-O0.s

```
pread.c:6: error: access to volatile variable 'z' for parameter '%e1' is not supported in
    ais annotations
pread.c:7: error: access to volatile variable 'x' for parameter '%e1' is not supported in
    ais annotations
pread.c:7: error: access to volatile variable 'z' for parameter '%e2' is not supported in
    ais annotations
pread.c:20: error: access to volatile variable 'z' for parameter '%e2' is not supported in
    ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3' is not supported in
    ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3' is not supported in
    ais annotations
```

```
ais annotations
7 pread.c:23: error: access to volatile variable 'z' for parameter '%e2' is not supported in ais annotations
8 7 errors detected.
```

### .4.16 pread-O3.s

```
pread.c:6: error: access to volatile variable 'z' for parameter '%e1' is not supported in ais annotations

pread.c:7: error: access to volatile variable 'x' for parameter '%e1' is not supported in ais annotations

pread.c:7: error: access to volatile variable 'z' for parameter '%e2' is not supported in ais annotations

pread.c:20: error: access to volatile variable 'z' for parameter '%e2' is not supported in ais annotations

pread.c:21: error: access to volatile variable 'z' for parameter '%e3' is not supported in ais annotations

pread.c:21: error: access to volatile variable 'z' for parameter '%e3' is not supported in ais annotations

pread.c:21: error: access to volatile variable 'z' for parameter '%e3' is not supported in ais annotations

pread.c:23: error: access to volatile variable 'z' for parameter '%e2' is not supported in ais annotations

7 errors detected.
```

# .5 Inline Assembly Annotated C Programs

#### .5.1 comment.c

```
int main() {
    asm("# CRITICAL COMMENT");
    return 0;
4 }
```

# .5.2 variable.c

```
int main(int argc, char* argv[]) {
     // a = int
// b = char
     // c = unsigned int
     // d = short
     // e = long
6
     // x = float
7
     // y = double
     int a = -10:
9
     asm("# annotation: %0 = int" : : "X"(a));
10
     char b = 'b';
11
     asm("# annotation: %0 = char" : : "X"(b));
12
     unsigned int c = -b;
13
     asm("# annotation: %0 = unsigned int" : : "X"(c));
14
     short d = 0x1;
15
     asm("# annotation: %0 = short" : : "X"(d));
16
     long e = 4294967296;
17
18
      asm("# annotation: %0 = long" : : "X"(e));
     float x = 3.141592653589793;
19
    asm("# annotation: %0 = float" : : "X"(x));
     double y = x / 2.3784;
21
asm("# annotation: %0 = double" : : "X"(y));
```

```
return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int) y + argc;
24 }
```

# .5.3 volatile.c

```
volatile int x;

int main() {
    asm("# annotation: %0 = High" : : "X"(x));
    return x + 1;

6 }
```

## .5.4 loop.c

```
1 int z;
2 int x;
_4 // security policies
5 // {L(z)=true}
6 // \{L(x)=z \% 2 == 0\}
8 // predicates on initial state
9 // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
     int r1 = 0;
13
     // {L(r2)=False}
14
     int r2 = 0;
15
16
17
     while(1) {
     do {
18
         // {_invariant: r1 % 2 == 0 /\ r1 <= z}
19
          // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
20
21
              // {_invariant: r1 <= z}
22
              // {_Gamma: r1 -> LOW}
23
             r1 = z;
          } while (r1 %2 != 0);
25
             r2 = x;
26
          } while (z != r1);
27
28
    return r2;
30 }
```

### .5.5 rooster.c

```
int rooster;
int drake;
// MEDIUM
int goose;

int fun(int a, int b, int c) {
    // CRITICAL COMMENT
    static int count = 0;
    int sum = a + b + c;
    if (sum < 0) {</pre>
```

```
return sum;
11
12
      if (a < b && b < c) {
13
          while (a != b) {
14
              a++;
15
               count++;
16
               while (b != c) {
17
                  c--;
18
19
                   count++;
               }
20
21
22
      return count;
23
24 }
25
26 int main(void) {
     // EXCEPTIONAL
27
     rooster = 1;
28
29
    drake = 5;
     goose = 10;
30
      int result;
      result = fun(rooster, drake, goose);
32
      return 0;
33
34 }
```

# .5.6 password.c

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
   user_t* next;
10
11
      char name[BUFF_LEN];
     // L(password) = High
12
13
     char password[BUFF_LEN];
      size_t balance;
14
15 };
17 user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
18
      strcpy(user_admin->name, "admin");
19
     strcpy(user_admin->password, "4dm1n__4eva");
20
21
     user_admin->balance = 1000000;
22
     user_t* user_alice = malloc(sizeof(user_t));
24
      strcpy(user_alice->name, "alice");
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
      user_alice->balance = 783;
26
27
      user_t* user_abdul = malloc(sizeof(user_t));
      strcpy(user_abdul->name, "abdul");
29
     strcpy(user_abdul->password, "passw0rd123");
```

```
user_abdul->balance = 2;
31
32
      user_admin->next = user_alice;
33
      user_alice->next = user_abdul;
34
      user_abdul ->next = NULL;
35
36
37
      return user_admin;
38 }
40 void print_users(user_t* users) {
      printf("--- USERS ---\n");
41
      size_t count = 0;
42
      while (users != NULL) {
43
          printf(" %02ld. %s\n", ++count, users->name);
          users = users->next;
45
46
47
      printf("\n");
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
      while (user_list != NULL) {
51
          if (strcmp(user_list->name, name) == 0) {
52
              return user_list;
53
54
          user_list = user_list->next;
55
56
      return NULL;
57
58 }
59
60 int main() {
61
      user_t* users = setup_users();
62
      printf("Welcome to BigBank Australia!\n");
63
64
      char username[BUFF_LEN];
65
      printf("Username: ");
66
      scanf("%255s", username);
67
68
      user_t* user = getUser(users, username);
69
70
      if (user == NULL) {
          printf("User < %s > does not exist.\n", username);
71
72
          return 0;
73
74
      char password[BUFF_LEN];
75
      printf("Password: ");
76
      scanf("%255s", password);
77
78
      if (strcmp(user->password, password) != 0) {
          printf("ERROR: incorrect password\n");
79
          return 0;
80
81
82
83
      printf("Logged in as < %s >!\n", user->name);
      printf("\n");
84
      printf("Welcome, %s!\n", user->name);
85
      printf("Your balance: $%ld\n", user->balance);
86
87 }
```

# .5.7 deadStoreElimination.c

```
int deadStore(int i, int n) {
      int key = 0xabcd;
// L(key) = high
2
      // do some work
      int result = 0;
6
7
      while (i > n) {
          result += key;
          i--;
9
10
11
      // clear out our secret key
12
      key = 0;
13
      return i + n;
14
15 }
16
int main(int argc, char *argv[]) {
      deadStore(argc, 2);
18
19 }
```

# .5.8 pread.c

```
volatile int z;
volatile int x;
4 // security policies
5 // {L(z)=true}
6 // \{L(x)=z \% 2 == 0\}
_{8} // predicates on initial state _{9} // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
    int r1 = 0;
// {L(r2)=False}
13
14
      int r2 = 0;
15
16
      while(1) {
17
18
           // {_invariant: r1 % 2 == 0 /\ r1 <= z}
19
          // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
20
21
               // {_invariant: r1 <= z}
22
23
               // {_Gamma: r1 -> LOW}
               r1 = z;
24
           } while (r1 %2 != 0);
25
               r2 = x;
           } while (z != r1);
27
28
29
      return r2;
30 }
```

# .6 Inline Assembly Annotated Assembly Output

### .6.1 comment-O0.s

```
1 .text
  .file "comment.c"
.globl main
                                   # -- Begin function main
p2align 4, 0x90 type main, @function
                                          # @main
6 main:
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register %rbp
movl $0, -4(\%rbp)
15 #APP
   # CRITICAL COMMENT
16
   #NO_APP
18 xorl %eax, %eax
popq %rbp
cfi_def_cfa %rsp, 8
21
   retq
^{22} .Lfunc_end0:
.size main, .Lfunc_end0-main
24 .cfi_endproc
                                          # -- End function
   .ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","",@progbits
28 .addrsig
```

#### .6.2 comment-O3.s

```
1 .text
   .file "comment.c"
   .globl main
                                 # -- Begin function main
4 .p2align 4, 0x90
5 .type main, @function
6 main:
                                      # @main
  .cfi_startproc
8 # %bb.0:
9 #APP
# CRITICAL COMMENT
   #NO_APP
xorl %eax, %eax
13 retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
                                      # -- End function
.ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","",@progbits
20 .addrsig
```

## .6.3 variable-O0.s

```
1 .text
   .file "variable.c"
   .section .rodata.cst8,"aM",@progbits,8
    .p2align 3
                              # -- Begin function main
5 .LCPI0_0:
.quad 4612538099476886402
                                   # double 2.378400000000001
   .section .rodata.cst4, "aM", @progbits, 4
.p2align 2
9 .LCPI0_1:
.long 1078530011
                                   # float 3.14159274
   .text
12
    .globl main
.p2align 4, 0x90
14 .type main, @function
                                            # @main
15 main:
.cfi_startproc
17 # %bb.0:
pushq %rbp
   .cfi_def_cfa_offset 16
   .cfi_offset %rbp, -16
20
   movq %rsp, %rbp
21
    .cfi_def_cfa_register %rbp
22
23
   movl $0, -4(\%rbp)
   movl %edi, -8(%rbp)
   movq %rsi, -16(%rbp)
movl $-10, -20(%rbp)
movl -20(%rbp), %eax
25
26
27
    #APP
28
    # annotation: %eax = int
    #NO_APP
30
    movb $98, -21(%rbp)
    movb -21(%rbp), %cl
32
    #APP
33
34
    # annotation: %cl = char
    #NO_APP
35
    xorl %eax, %eax
    movsbl -21(%rbp), %edx
37
    subl %edx, %eax
38
    mov1 \%eax, -28(\%rbp)
39
    movl
          -28(%rbp), %eax
40
41
    #APP
    # annotation: %eax = unsigned int
42
    #NO_APP
43
    movw $1, -30(%rbp)
44
    movw -30(%rbp), %r8w
45
46
    #APP
    # annotation: %r8w = short
47
    #NO_APP
    movabsq $4294967296, %rsi
49
                                    # imm = 0x100000000
    movq %rsi, -40(%rbp)
movq -40(%rbp), %rsi
50
51
    #APP
52
# annotation: %rsi = long
54 #NO_APP
55 movss .LCPIO_1(%rip), %xmm0 # xmm0 = mem[0],zero,zero,zero
```

```
movss %xmm0, -44(%rbp)
57
    movss -44(%rbp), %xmm0
                                  # xmm0 = mem[0],zero,zero,zero
   #APP
58
    # annotation: %xmm0 = float
59
   #NO_APP
60
   movsd .LCPI0_0(%rip), %xmm0
movss -44(%rbp), %xmm1
                                  # xmm0 = mem[0], zero
61
62
                                   # xmm1 = mem[0],zero,zero,zero
   cvtss2sd %xmm1, %xmm1
63
   divsd %xmm0, %xmm1
   movsd %xmm1, -56(%rbp)
65
    movsd -56(%rbp), %xmm0
                                   # xmm0 = mem[0],zero
66
    #APP
67
    # annotation: %xmm0 = double
68
    #NO_APP
69
   movq -40(\%rbp), \%rax
70
    cqto
71
    movl $32, %esi
72
    idivq %rsi
73
74
                                           # kill: def $eax killed $eax killed $rax
    addl -20(%rbp), %eax
addl -28(%rbp), %eax
75
76
    movswl -30(%rbp), %edi
77
    addl %edi, %eax
78
79
    cvttss2si -44(%rbp), %edi
    addl %edi, %eax
80
81
    cvttsd2si -56(%rbp), %edi
82
   addl %edi, %eax
83 addl -8(%rbp), %eax
84 popq %rbp
    .cfi_def_cfa %rsp, 8
85
86
87 .Lfunc_end0:
ss .size main, .Lfunc_end0-main
   .cfi_endproc
89
                                           # -- End function
   .ident "clang version 10.0.0-4ubuntu1"
92 .section ".note.GNU-stack","", @progbits
93 .addrsig
```

### .6.4 variable-O3.s

```
1 .text
   .file "variable.c"
section .rodata.cst4, "aM", @progbits,4
4 .p2align 2
                # -- Begin function main
5 .LCPI0_0:
6 .long 1078530011
                               # float 3.14159274
   .section .rodata.cst8, "aM", @progbits,8
   .p2align 3
9 .LCPIO_1:
.quad 4608627556095693531  # double 1.3208849398808329
11
   .text
12
   .globl main
.p2align 4, 0x90
.type main, @function
                                      # @main
15 main:
.cfi_startproc
```

```
17 # %bb.0:
                                          # kill: def $edi killed $edi def $rdi
   #APP
19
   # annotation: \$-10 = int
20
   #NO_APP
21
22
   #APP
   # annotation: $98 = char
23
   #NO_APP
24
   #APP
   # annotation: $-98 = unsigned int
26
   #NO_APP
27
   #APP
28
   # annotation: $1 = short
29
   #NO_APP
30
   #APP
31
   # annotation: $4294967296 = long
32
33
   #NO_APP
   movss .LCPIO_O(%rip), %xmm0
                                # xmm0 = mem[0],zero,zero,zero
34
35
   #APP
   # annotation: %xmm0 = float
36
   #NO_APP
   movsd .LCPIO_1(%rip), %xmm0
                                 # xmm0 = mem[0],zero
38
39
   # annotation: %xmm0 = double
40
   #NO_APP
41
12 leal 134217625(%rdi), %eax
43 retq
44 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
46
                                          # -- End function
.ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","", @progbits
o .addrsig
```

#### .6.5 volatile-O0.s

```
1 .text
   .file "volatile.c"
   .globl main
                                     # -- Begin function main
4 .p2align 4, 0x90
5 .type main, @function
6 main:
                                            # @main
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
10
    .cfi_offset %rbp, -16
11
movq %rsp, %rbp
   .cfi_def_cfa_register %rbp
   movl $0, -4(%rbp)
movl x, %eax
14
15
16
   #APP
   # annotation: %eax = High
17
18 #NO_APP
movl x, %eax 20 addl $1, %eax
```

```
popq %rbp
22
   .cfi_def_cfa %rsp, 8
23 retq
24 .Lfunc_end0:
.size main, .Lfunc_end0-main
26
   .cfi_endproc
                                       # -- End function
27
   .type x,@object
                               # 0x
28
29 .comm x,4,4
.ident "clang version 10.0.0-4ubuntu1 "
   .section ".note.GNU-stack","",@progbits
31
32
   .addrsig
33 .addrsig_sym x
```

# .6.6 volatile-O3.s

```
1 .text
2 .file "volatile.c"
3 .globl main
                                         # -- Begin function main
2 .p2align 4, 0x90 5 .type main,@function
 6 main:
                                               # @main
7 .cfi_startproc
 8 # %bb.0:
9 movl x(%rip), %eax
10 #APP
# annotation: %eax = High
12 #NO_APP
13 movl x(%rip), %eax
14 addl $1, %eax
15 retq
16 .Lfunc_end0:
.size main, .Lfunc_end0-main
18
    .cfi_endproc
                                             # -- End function
19
20 .type x,@object
                                      # @x
21 .comm x, 4, 4
    .ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","", @progbits
24 .addrsig
25 .addrsig_sym x
```

# .6.7 loop-O0.s

```
14 movl $0, -4(%rbp)
15 movl $0, -8(%rbp)
16 movl $0, -12(%rbp)
17 .LBB0_1:
                                            # =>This Loop Header: Depth=1
                                               Child Loop BBO_2 Depth 2
                                                   Child Loop BBO_3 Depth 3
19
    jmp .LBB0_2
20
21 .LBB0_2:
                                            # Parent Loop BBO_1 Depth=1
                                            # => This Loop Header: Depth=2
                                                  Child Loop BBO_3 Depth 3
   jmp .LBB0_3
24
25 .LBB0_3:
                                             Parent Loop BBO_1 Depth=1
                                                Parent Loop BBO_2 Depth=2
                                            # => This Inner Loop Header: Depth=3
28 movl z, %eax
29 movl %eax, -8(%rbp)
30 # %bb.4:
                                           # in Loop: Header=BB0_3 Depth=3
movl -8(\%rbp), \%eax
   cltd
   movl $2, %ecx
33
   idivl %ecx
35 cmpl $0, %edx
36 jne .LBBO_3
37 # %bb.5:
                                              in Loop: Header=BBO_2 Depth=2
38 movl x, %eax
39 movl %eax, -12(%rbp)
40 # %bb.6:
                                            # in Loop: Header=BBO_2 Depth=2
movl z, %eax
42 cmpl -8(%rbp), %eax
jne .LBB0_2
44 # %bb.7:
                                              in Loop: Header=BBO_1 Depth=1
jmp .LBB0_1
46 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
                                           # -- End function
                                 # @z
50 .type z,@object
   .comm z,4,4
   .type x,@object
52
53
    .comm x,4,4
    .ident "clang version 10.0.0-4ubuntu1"
55
   .section ".note.GNU-stack","", @progbits
56 .addrsig
   .addrsig_sym z
.addrsig_sym x
```

#### .6.8 loop-O3.s

```
jne .LBB0_2
   .p2align 4, 0x90
12 .LBB0_1:
                                        # =>This Inner Loop Header: Depth=1
jmp .LBB0_1
.p2align 4, 0x90
15 .LBB0_2:
                                        # =>This Inner Loop Header: Depth=1
jmp .LBB0_2
17 .Lfunc_end0:
.size main, .Lfunc_end0-main
19
   .cfi_endproc
                                        # -- End function
20
                                # @z
21
   .type z,@object
   .comm z,4,4
22
   .type x,@object
                                # @x
   .comm x,4,4
24
   .ident "clang version 10.0.0-4ubuntu1 "
  .section ".note.GNU-stack","", @progbits
27 .addrsig
```

## .6.9 rooster-O0.s

```
1 .text
   .file "rooster.c"
    .globl fun
                                     # -- Begin function fun
   .p2align 4, 0x90
 5 .type fun, @function
                                          # @fun
6 fun:
 7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
11
12
   .cfi_offset %rbp, -16
   movq %rsp, %rbp
   .cfi_def_cfa_register %rbp
13
14 movl %edi, -8(%rbp)
movl %esi, -12(%rbp)
movl %edx, -16(%rbp)
movl -8(%rbp), %eax
   addl -12(%rbp), %eax
18
19
   addl -16(%rbp), %eax
20 movl %eax, -20(%rbp)
21 cmpl $0, -20(%rbp)
    jge .LBB0_2
22
23 # %bb.1:
movl -20(%rbp), %eax
25 movl %eax, -4(%rbp)
26
   jmp .LBB0_12
27 .LBB0_2:
movl -8(\%rbp), \%eax
29 cmpl -12(%rbp), %eax
31 # %bb.3:
_{32} movl -12(%rbp), %eax
33 cmpl -16(%rbp), %eax
35 # %bb.4:
36 jmp .LBB0_5
```

```
37 .LBB0_5:
                                                # =>This Loop Header: Depth=1
                                                     Child Loop BBO_7 Depth 2
39 movl -8(%rbp), %eax
40 cmpl -12(%rbp), %eax
je .LBB0_10
42 # %bb.6:
                                                   in Loop: Header=BBO_5 Depth=1
    movl -8(\%rbp), \%eax
43
    addl $1, %eax
44
movl \%eax, -8(\%rbp)
46 movl fun.count, %eax
47 addl $1, %eax
48 movl %eax, fun.count
49 .LBB0_7:
                                                # Parent Loop BB0_5 Depth=1
50
                                                # => This Inner Loop Header: Depth=2
51 movl -12(%rbp), %eax
52 cmpl -16(%rbp), %eax
53 je .LBBO_9
54 # %bb.8:
                                                   in Loop: Header=BBO_7 Depth=2
55 movl -16(%rbp), %eax

56 addl $-1, %eax

57 movl %eax, -16(%rbp)

58 movl fun.count, %eax
addl $1, %eax
60 movl %eax, fun.count
61 jmp .LBB0_7
                                                   in Loop: Header=BB0_5 Depth=1
62 .LBB0_9:
63 jmp .LBB0_5
64 .LBB0_10:
65 jmp .LBB0_11
66 .LBB0_11:
movl fun.count, %eax movl %eax, -4(%rbp)
69 .LBB0_12:
70 movl -4(%rbp), %eax
71 popq %rbp
72
    .cfi_def_cfa %rsp, 8
73 retq
74 .Lfunc_end0:
.size fun, .Lfunc_end0-fun
76
    .cfi_endproc
                                               # -- End function
78 .globl main
                                         # -- Begin function main
79 .p2align 4, 0x90
80 .type main, @function
81 main:
                                                # @main
82 .cfi_startproc
83 # %bb.0:
84 pushq %rbp
    .cfi_def_cfa_offset 16
85
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
87
    .cfi_def_cfa_register %rbp
88
    subq $16, %rsp
89
    movl $0, -4(%rbp)
movl $1, rooster
90
92 movl $5, drake
movl $10, goose
```

```
movl rooster, %edi
movl drake, %esi
movl goose, %edx
     callq fun
98 xorl %ecx, %ecx
99 movl %eax, -8(%rbp)
100 movl %ecx, %eax
101 addq $16, %rsp
102 popq %rbp
.cfi_def_cfa %rsp, 8
104
     retq
105 .Lfunc_end1:
.size main, .Lfunc_end1-main
107
     .cfi_endproc
108
                                                # -- End function
    .type fun.count,@object # @fun.count
.local fun.count
109
110
.comm fun.count,4,4
.type rooster,@object
                                      # @rooster
     .comm rooster,4,4
113
114
     .type drake, @object
                                       # @drake
     .comm drake,4,4
115
     .type goose,@object
                                       # @goose
116
117
     .comm goose,4,4
     .ident "clang version 10.0.0-4ubuntu1"
118
     .section ".note.GNU-stack","",@progbits
119
     .addrsig
120
     .addrsig_sym fun
121
     .addrsig_sym fun.count
122
123
     .addrsig_sym rooster
.addrsig_sym drake
.addrsig_sym goose
```

#### .6.10 rooster-O3.s

```
1 .text
    .file "rooster.c"
.globl fun
                                          # -- Begin function fun
    .p2align 4, 0x90
 5 .type fun, @function
 6 fun:
                                                 # @fun
 7 .cfi_startproc
 8 # %bb.0:
                                                 # kill: def $edx killed $edx def $rdx
                                                 # kill: def $esi killed $esi def $rsi
                                                 # kill: def $edi killed $edi def $rdi
11
12 leal (%rsi,%rdi), %eax
13 addl %edx, %eax
14 js .LBB0_9
15 # %bb.1:
movl fun.count(%rip), %eax cmpl %esi, %edi jge .LBBO_9
19 # %bb.2:
cmpl %edx, %esi
jge .LBB0_9
22 # %bb.3:
```

```
23 leal 1(%rdi), %ecx
24 cmpl %esi, %ecx
25 jne .LBBO_5
26 # %bb.4:
30 jmp .LBB0_8
31 .LBB0_5:
32 addl $-1, %esi
    .p2align 4, 0x90
33
                                            # =>This Inner Loop Header: Depth=1
34 .LBB0_6:
35 addl $-1, %esi
36 cmpl %esi, %edi
jne .LBB0_6
38 # %bb.7:
39 addl %edx, %eax
subl %esi, %eax
41 .LBB0_8:
42 movl %eax, fun.count(%rip)
43 .LBB0_9:
                                            # kill: def $eax killed $eax killed $rax
44
45
    retq
46 .Lfunc_end0:
.size fun, .Lfunc_end0-fun
    .cfi_endproc
                                           # -- End function
49
50 .globl main
                                      # -- Begin function main
.p2align 4, 0x90
.type main, @function
53 main:
                                            # @main
.cfi_startproc
55 # %bb.0:
movl $1, rooster(%rip)
movl $5, drake(%rip)
movl $10, goose(%rip)
addl $9, fun.count(%rip)
60 xorl %eax, %eax
61 retq
62 .Lfunc_end1:
    .size main, .Lfunc_end1-main
63
64
    .cfi_endproc
                                           # -- End function
    .type fun.count,@object
                                 # @fun.count
66
    .local fun.count
67
    .comm fun.count,4,4
68
    .type rooster, @object
                                   # @rooster
69
70
    .comm rooster,4,4
    .type drake, @object
                                   # @drake
71
72
    .comm drake,4,4
73
    .type goose,@object
                                   # @goose
74
    .comm goose,4,4
    .ident "clang version 10.0.0-4ubuntu1"
75
    .section ".note.GNU-stack","", @progbits
76
77 .addrsig
```

## .6.11 password-O0.s

```
1 .text
    .file "password.c"
.globl setup_users
                                          # -- Begin function setup_users
    .p2align 4, 0x90
 .type setup_users,@function
 6 setup_users:
                                                 # @setup_users
   .cfi_startproc
 8 # %bb.0:
9 pushq %rbp
10
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
11
    movq %rsp, %rbp
12
    .cfi_def_cfa_register %rbp
13
14
    subq $80, %rsp
    movl $528, %edi
                                       # imm = 0x210
15
    callq malloc
16
    movq %rax, -8(%rbp)
17
    movq -8(\%rbp), \%rax
18
    addq $8, %rax
movl $.L.str, %esi
movq %rax, %rdi
callq strcpy
19
20
21
22
    movq -8(\%rbp), \%rcx
23
    addq $264, %rcx
24
                                       # imm = 0x108
    movl $.L.str.1, %esi
movq %rcx, %rdi
movq %rax, -32(%rbp)
25
26
27
                                       # 8-byte Spill
    callq strcpy
28
    movq -8(%rbp), %rcx
movq $1000000, 520(%rcx)
mov1 $528, %edi
movq %rax, -40(%rbp)
29
                                       # imm = 0xF4240
30
31
                                       # imm = 0x210
                                       # 8-byte Spill
32
    callq malloc
33
    movq %rax, -16(%rbp)
34
    movq -16(%rbp), %rax addq $8, %rax
35
36
    movl $.L.str.2, %esi
37
    movq %rax, %rdi
    callq strcpy
39
    movq -16(%rbp), %rcx addq $264, %rcx
40
                                       # imm = 0x108
41
    movl $.L.str.3, %esi
42
    movq %rcx, %rdi
    movq %rax, -48(%rbp)
                                       # 8-byte Spill
44
    callq strcpy
45
    movq -16(%rbp), %rcx
46
    movq $783, 520(%rcx)
                                       # imm = 0x30F
47
    movl $528, %edi
                                       # imm = 0x210
48
    movq %rax, -56(%rbp) callq malloc
                                       # 8-byte Spill
49
    movq %rax, -24(%rbp)
51
    movq -24(%rbp), %rax
52
    addq $8, %rax
53
    movl $.L.str.4, %esi
movq %rax, %rdi
54
55
56 callq strcpy
57 movq -24(%rbp), %rcx
```

```
addq $264, %rcx
movl $.L.str.5, %esi
                               # imm = 0x108
58
     movq %rcx, %rdi
60
     movq %rax, -64(%rbp)
                                     # 8-byte Spill
61
     callq strcpy
62
    movq -24(%rbp), %rcx
movq $2, 520(%rcx)
63
64
     movq -16(%rbp), %rcx
65
     movq -8(%rbp), %rdx
     movq %rcx, (%rdx)
67
    movq -24(%rbp), %rcx
movq -16(%rbp), %rdx
68
69
     movq %rcx, (%rdx)
70
     movq -24(%rbp), %rcx
71
    movq $0, (%rcx)
72
    movq -8(%rbp), %rcx
movq %rax, -72(%rbp)
73
                                    # 8-byte Spill
74
     movq %rcx, %rax
75
76
     addq $80, %rsp
     popq %rbp
77
78
     .cfi_def_cfa %rsp, 8
79
    retq
80 .Lfunc_end0:
size setup_users, .Lfunc_end0-setup_users
    .cfi_endproc
82
                                             # -- End function
    .globl print_users
                                        # -- Begin function print_users
84
85 .p2align 4, 0x90
.type print_users,@function
87 print_users:
                                              # @print_users
88 .cfi_startproc
89 # %bb.0:
90 pushq %rbp
    .cfi_def_cfa_offset 16
91
    .cfi_offset %rbp, -16
92
    movq %rsp, %rbp
93
    .cfi_def_cfa_register %rbp
94
     subq $16, %rsp
    movq %rdi, -8(%rbp)
96
97
     movabsq $.L.str.6, %rdi
     movb $0, %al
98
99
    callq printf
movq $0, -16(%rbp)
101 .LBB1_1:
                                              # =>This Inner Loop Header: Depth=1
102 cmpq $0, -8(%rbp)
     je .LBB1_3
103
104 # %bb.2:
                                              # in Loop: Header=BB1_1 Depth=1
movq -16(%rbp), %rax
    addq $1, %rax
movq %rax, -16(%rbp)
movq -8(%rbp), %rcx
106
107
108
     addq $8, %rcx
109
     movabsq $.L.str.7, %rdi
110
movq %rax, %rsi
movq %rcx, %rdx
movb $0, %al
114 callq printf
```

```
movq -8(%rbp), %rcx
movq (%rcx), %rcx
movq %rcx, -8(%rbp)
119 .LBB1_3:
    movabsq $.L.str.8, %rdi
120
     movb $0, %al
121
122 callq printf
addq $16, %rsp
popq %rbp
     .cfi_def_cfa %rsp, 8
125
126
     retq
127 .Lfunc_end1:
.size print_users, .Lfunc_end1-print_users
    .cfi_endproc
129
                                               # -- End function
130
                                        # -- Begin function getUser
    .globl getUser
131
.p2align 4, 0x90
.type getUser, @function
134 getUser:
                                               # @getUser
.cfi_startproc
136 # %bb.0:
pushq %rbp
    .cfi_def_cfa_offset 16
138
     .cfi_offset %rbp, -16
139
    movq %rsp, %rbp
    .cfi_def_cfa_register %rbp
141
subq $32, %rsp
movq %rdi, -16(%rbp)
movq %rsi, -24(%rbp)
145 .LBB2_1:
                                               # =>This Inner Loop Header: Depth=1
146 cmpq $0, -16(%rbp)
147 je .LBB2_5
148 # %bb.2:
                                               # in Loop: Header=BB2_1 Depth=1
movq -16(%rbp), %rax

addq $8, %rax

movq -24(%rbp), %rsi
movq %rax, %rdi
^{153} callq strcmp
154
    cmpl $0, %eax
jne .LBB2_4
156 # %bb.3:
movq -16(%rbp), %rax
movq %rax, -8(%rbp)
     jmp .LBB2_6
159
                                                  in Loop: Header=BB2_1 Depth=1
160 .LBB2_4:
movq -16(%rbp), %rax
162 movq (%rax), %rax
163 movq %rax, -16(%rbp)
164 jmp .LBB2_1
165 .LBB2_5:
166 movq $0, -8(%rbp)
167 .LBB2_6:
168 movq -8(%rbp), %rax
169 addq $32, %rsp
170 popq %rbp
.cfi_def_cfa %rsp, 8
```

```
172 retq
173 .Lfunc_end2:
.size getUser, .Lfunc_end2-getUser
    .cfi_endproc
175
                                            # -- End function
176
177
    .globl main
                                      # -- Begin function main
    .p2align 4, 0x90
178
    .type main, @function
179
                                            # @main
180 main:
.cfi_startproc
182 # %bb.0:
pushq %rbp
    .cfi_def_cfa_offset 16
184
185
    .cfi_offset %rbp, -16
    movq %rsp, %rbp
186
    .cfi_def_cfa_register %rbp
187
    subq $576, %rsp
movl $0, -4(%rbp)
188
                                   # imm = 0x240
189
190
    callq setup_users
    movq %rax, -16(%rbp)
191
    movabsq $.L.str.9, %rdi
192
    movb $0, %al
193
    callq printf
194
    movabsq $.L.str.10, %rdi
195
    movl %eax, -548(%rbp)
movb $0, %al
                                  # 4-byte Spill
196
197
    callq printf
198
    leaq -272(%rbp), %rsi
199
    movabsq $.L.str.11, %rdi
200
    movl %eax, -552(%rbp)
movb $0, %al
                                   # 4-byte Spill
201
202
    callq __isoc99_scanf
203
    leaq -272(%rbp), %rsi
204
    movq -16(%rbp), %rdi
movl %eax, -556(%rbp)
205
                                   # 4-byte Spill
206
207
    callq getUser
    movq %rax, -280(%rbp)
208
    cmpq $0, -280(%rbp)
209
    jne .LBB3_2
210
211 # %bb.1:
leaq -272(%rbp), %rsi
    movabsq $.L.str.12, %rdi
213
214 movb $0, %al
215
    callq printf
    mov1 $0, -4(%rbp)
jmp .LBB3_5
216
217
218 .LBB3_2:
movabsq $.L.str.13, %rdi
    movb $0, %al
220
221
    callq printf
222
    leaq -544(%rbp), %rsi
    movabsq $.L.str.11, %rdi
223
    mov1 \%eax, -560(\%rbp)
224
                                   # 4-byte Spill
225
    movb $0, %al
226
    callq __isoc99_scanf
227 leaq -544(%rbp), %rsi
228 movq -280(%rbp), %rcx
```

```
229 addq $264, %rcx
230 movq %rcx, %rdi
231 movl %eax, -564(%rbp)
                          # imm = 0x108
                             # 4-byte Spill
     callq strcmp
232
233 cmpl $0, % eax
234
     je .LBB3_4
235 # %bb.3:
movabsq $.L.str.14, %rdi
237 movb $0, %al
238 callq printf
    movl \$0, -4(\%rbp)
239
241 .LBB3_4:
_{242} movq -280(%rbp), %rax
    addq $8, %rax
243
    movabsq $.L.str.15, %rdi
244
245 movq %rax, %rsi
246 movb $0, %al
247
    callq printf
    movabsq $.L.str.8, %rdi
248
     movl %eax, -568(%rbp)
                               # 4-byte Spill
249
    movb $0, %al
250
    callq printf
251
    movq -280(%rbp), %rcx addq $8, %rcx
252
253
254
     movabsq $.L.str.16, %rdi
    movq %rcx, %rsi
255
    movl %eax, -572(%rbp)
movb $0, %al
                                   # 4-byte Spill
256
257
258
     callq printf
    movq -280(%rbp), %rcx
movq 520(%rcx), %rsi
259
260
     movabsq $.L.str.17, %rdi
261
    movl %eax, -576(%rbp)
movb $0, %al
                                  # 4-byte Spill
262
263
264
     callq printf
265 .LBB3_5:
movl -4(\%rbp), \%eax
    addq $576, %rsp
                                    # imm = 0x240
267
268
     .cfi_def_cfa %rsp, 8
269
270 retq
271 .Lfunc_end3:
.size main, .Lfunc_end3-main
273
    .cfi_endproc
                                            # -- End function
274
.type .L.str,@object
                                  # @.str
.section .rodata.str1.1, "aMS", @progbits,1
277 .L.str:
    .asciz "admin"
278
    .size .L.str, 6
279
280
.type .L.str.1,@object
                                  # @.str.1
282 .L.str.1:
    .asciz "4dm1n__4eva"
284 .size .L.str.1, 12
```

```
.type .L.str.2,@object # @.str.2
287 .L.str.2:
288 .asciz "alice"
    .size .L.str.2, 6
290
291
    .type .L.str.3,@object
                               # @.str.3
292 .L.str.3:
.asciz "!alice12!_veuje@@hak"
294 .size .L.str.3, 21
295
    .type .L.str.4,@object
                            # @.str.4
296
297 .L.str.4:
298 .asciz "abdul"
299 .size .L.str.4, 6
300
    .type .L.str.5,@object
                                 # @.str.5
301
302 .L.str.5:
303 .asciz "passw0rd123"
304
    .size .L.str.5, 12
305
    .type .L.str.6,@object
                                # @.str.6
307 .L.str.6:
308 .asciz "--- USERS ---\n"
    .size .L.str.6, 15
309
310
    .type .L.str.7,@object
                                 # @.str.7
311
312 .L.str.7:
313 .asciz " %02ld. %s\n"
    .size .L.str.7, 12
314
315
316
   .type .L.str.8,@object
                                # @.str.8
317 .L.str.8:
318 .asciz "\n"
    .size .L.str.8, 2
319
320
    .type .L.str.9,@object
321
                                 # @.str.9
322 .L.str.9:
.asciz "Welcome to BigBank Australia!\n"
    .size .L.str.9, 31
324
325
   .type .L.str.10,@object # @.str.10
326
327 .L.str.10:
328 .asciz "Username: "
    .size .L.str.10, 11
329
330
                                # @.str.11
331
    .type .L.str.11,@object
332 .L.str.11:
333 .asciz "%255s"
334
    .size .L.str.11, 6
335
.type .L.str.12,@object
                               # @.str.12
337 .L.str.12:
.asciz "User < %s > does not exist.\n"
    .size .L.str.12, 29
339
340
341 .type .L.str.13,@object
                               # @.str.13
342 .L.str.13:
```

```
343 .asciz "Password: "
344
    .size .L.str.13, 11
345
    .type .L.str.14,@object
                                   # @.str.14
346
347 .L.str.14:
348 .asciz "ERROR: incorrect password\n"
    .size .L.str.14, 27
349
350
351 .type .L.str.15,@object
                                  # @.str.15
352 .L.str.15:
.asciz "Logged in as < %s >!\n"
    .size .L.str.15, 22
354
355
356
    .type .L.str.16,@object
                                  # @.str.16
357 .L.str.16:
    .asciz "Welcome, %s!\n"
.size .L.str.16, 14
358
359
360
    .type .L.str.17,@object # @.str.17
362 .L.str.17:
363 .asciz "Your balance: $%ld\n"
    .size .L.str.17, 20
364
365
366
    .ident "clang version 10.0.0-4ubuntu1"
    .section ".note.GNU-stack","", @progbits
367
368
     .addrsig
    .addrsig_sym setup_users
369
    .addrsig_sym malloc
370
371
    .addrsig_sym strcpy
372
    .addrsig_sym printf
373
    .addrsig_sym getUser
.addrsig_sym strcmp
.addrsig_sym __isoc99_scanf
```

## .6.12 password-O3.s

```
1 .text
    .file "password.c"
    .globl setup_users
                                      # -- Begin function setup_users
    .p2align 4, 0x90
 .type setup_users,@function
 6 setup_users:
                                             # @setup_users
    .cfi_startproc
8 # %bb.0:
9 pushq %r14
    .cfi_def_cfa_offset 16
10
    pushq %rbx
11
12
    .cfi_def_cfa_offset 24
    pushq %rax
13
    .cfi_def_cfa_offset 32
15
    .cfi_offset %rbx, -24
    .cfi_offset %r14, -16
16
    movl $528, %edi
17
                                    # imm = 0x210
   callq malloc
18
movq %rax, %r14
movl $1768776801, 8(%rax)
movw $110, 12(%rax)
                                   # imm = 0x696D6461
```

```
movabsq \$3773839939640058932, \%rax # imm = 0x345F5F6E316D6434
22
    movq %rax, 264(%r14)
movl $6387301, 272(%r14)
23
                                    # imm = 0x617665
24
    movq $1000000, 520(%r14)
movl $528, %edi
                                     \# imm = 0xF4240
25
                                     # imm = 0x210
26
    callq malloc
27
    movq %rax, %rbx
movl $1667853409, 8(%rax)
28
                                    # imm = 0x63696C61
29
    movw $101, 12(\%rax)
    movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
31
32
    movq %rax, 277(%rbx)
    movups .L.str.3(%rip), %xmm0 movups %xmm0, 264(%rbx)
33
34
    movq $783, 520(%rbx)
                                     # imm = 0x30F
    movl $528, %edi
                                     # imm = 0x210
36
    callq malloc
37
                                    # imm = 0x75646261
38
    movl $1969513057, 8(%rax)
    movw $108, 12(%rax)
39
    movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
    movq %rcx, 264(%rax)
movl $3355185, 272(%rax)
41
                                   # imm = 0x333231
    movq $2, 520(%rax)
43
    movq %rbx, (%r14)
44
45
    movq %rax, (%rbx)
    movq $0, (%rax)
movq %r14, %rax
addq $8, %rsp
46
47
48
    .cfi_def_cfa_offset 24
49
50
    popq %rbx
    .cfi_def_cfa_offset 16
51
52
    popq %r14
    .cfi_def_cfa_offset 8
53
   retq
55 .Lfunc_end0:
   .size setup_users, .Lfunc_end0-setup_users
56
57
    .cfi_endproc
                                             # -- End function
58
.globl print_users
                                       # -- Begin function print_users
60 .p2align 4, 0x90
61 .type print_users,@function
62 print_users:
                                              # @print_users
63 .cfi_startproc
64 # %bb.0:
pushq %r14
    .cfi_def_cfa_offset 16
66
67
    pushq %rbx
    .cfi_def_cfa_offset 24
68
69
    pushq %rax
    .cfi_def_cfa_offset 32
70
    .cfi_offset %rbx, -24
71
    .cfi_offset %r14, -16
72
73
   movq %rdi, %r14
74
    movl $.Lstr, %edi
    callq puts
75
76
    testq %r14, %r14
77 je .LBB1_3
78 # %bb.1:
```

```
_{79} movl $1, \%ebx
    .p2align 4, 0x90
81 .LBB1_2:
                                           # =>This Inner Loop Header: Depth=1
82 leaq 8(%r14), %rdx
83 movl $.L.str.7, %edi
    movq %rbx, %rsi
xorl %eax, %eax
84
85
    callq printf
86
87 movq (%r14), %r14
88 addq $1, %rbx
    testq %r14, %r14
89
    jne .LBB1_2
90
91 .LBB1_3:
92 movl $10, %edi
93 addq $8, %rsp
    .cfi_def_cfa_offset 24
95
    popq %rbx
    .cfi_def_cfa_offset 16
96
97 popq %r14
.cfi_def_cfa_offset 8
    jmp putchar
                                 # TAILCALL
100 .Lfunc_end1:
.size print_users, .Lfunc_end1-print_users
102
    .cfi_endproc
                                          # -- End function
103
                                     # -- Begin function getUser
    .globl getUser
104
.p2align 4, 0x90
.type getUser, @function
                                           # @getUser
107 getUser:
108 .cfi_startproc
109 # %bb.0:
pushq %r14
    .cfi_def_cfa_offset 16
111
pushq %rbx
    .cfi_def_cfa_offset 24
113
114
    pushq %rax
    .cfi_def_cfa_offset 32
115
.cfi_offset %rbx, -24
.cfi_offset %r14, -16
testq %rdi, %rdi
je .LBB2_4
120 # %bb.1:
movq %rsi, %r14
movq %rdi, %rbx
    .p2align 4, 0x90
123
                                           # =>This Inner Loop Header: Depth=1
124 .LBB2_2:
125 leaq 8(%rbx), %rdi
movq %r14, %rsi
callq strcmp
128
    testl %eax, %eax
129 je .LBB2_5
130 # %bb.3:
                                           # in Loop: Header=BB2_2 Depth=1
movq (%rbx), %rbx
testq %rbx, %rbx
jne .LBB2_2
134 .LBB2_4:
xorl %ebx, %ebx
```

```
136 .LBB2_5:
    movq %rbx, %rax
addq $8, %rsp
137
138
    .cfi_def_cfa_offset 24
139
    popq %rbx
140
    .cfi_def_cfa_offset 16
141
142
    popq %r14
    .cfi_def_cfa_offset 8
143
144 retq
145 .Lfunc_end2:
    .size getUser, .Lfunc_end2-getUser
146
147
     .cfi_endproc
                                            # -- End function
148
                                      # -- Begin function main
149
    .globl main
.p2align 4, 0x90
    .type main, @function
151
                                            # @main
152 main:
.cfi_startproc
154 # %bb.0:
pushq %r15
    .cfi_def_cfa_offset 16
156
    pushq %r14
157
    .cfi_def_cfa_offset 24
158
159
    pushq %rbx
    .cfi_def_cfa_offset 32
160
                                    # imm = 0x200
161
    subq $512, %rsp
    .cfi_def_cfa_offset 544
162
    .cfi_offset %rbx, -32
163
    .cfi_offset %r14, -24
164
    .cfi_offset %r15, -16
165
    movl $528, %edi
                                    # imm = 0x210
166
    callq malloc
167
    movq %rax, %rbx
168
    movl $1768776801, 8(%rax)
                                   # imm = 0x696D6461
169
    movw $110, 12(%rax)
170
    movabsq 3773839939640058932, %rax # imm = 0x345F5F6E316D6434
171
    movq %rax, 264(%rbx)
172
    movl $6387301, 272(%rbx)
173
                                   # imm = 0x617665
    movq $1000000, 520(%rbx)
movl $528, %edi
                                   # imm = 0xF4240
174
175
                                    # imm = 0x210
    callq malloc
176
177
    movq %rax, %r14
    movl $1667853409, 8(%rax) # imm = 0x63696C61
178
    movw $101, 12(%rax)
179
    movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
180
    movq %rax, 277(%r14)
181
    movups .L.str.3(%rip), %xmm0
182
    movups %xmm0, 264(%r14)
183
    movq $783, 520(%r14)
movl $528, %edi
                                   # imm = 0x30F
184
                                   # imm = 0x210
185
    callq malloc
186
    movl $1969513057, 8(%rax)
                                   # imm = 0x75646261
187
188
    movw $108, 12(%rax)
    movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
189
movq %rcx, 264(%rax)
movl $3355185, 272(%rax)
                                  # imm = 0x333231
movq $2, 520(%rax)
```

```
movq %r14, (%rbx)
movq %rax, (%r14)
movq $0, (%rax)
193
194
195
     movl $.Lstr.18, %edi
196
     callq puts
197
     movl $.L.str.10, %edi
xorl %eax, %eax
198
199
     callq printf
200
movq %rsp, %rsi
202 movl $.L.str.11, %edi
203 xorl %eax, %eax
     callq __isoc99_scanf
204
testq %rbx, %rbx
206  je .LBB3_4
207 # %bb.1:
208 movq %rsp, %r15
.p2align 4, 0x90
210 .LBB3_2:
                                                 # =>This Inner Loop Header: Depth=1
leaq 8(\%rbx), \%r14
212 movq %r14, %rdi
213 movq %r15, %rsi
214 callq strcmp
testl %eax, %eax
je .LBB3_5
217 # %bb.3:
                                                    in Loop: Header=BB3_2 Depth=1
218 movq (%rbx), %rbx
testq %rbx, %rbx
220   jne .LBB3_2
221 .LBB3_4:
222 movq %rsp, %rsi
223 movl $.L.str.12, %edi
224 .LBB3_8:
225 xorl %eax, %eax
226 callq printf
     jmp .LBB3_9
227
228 .LBB3_5:
229 movl $.L.str.13, %edi
230 xorl %eax, %eax
    callq printf
231
     leaq 256(%rsp), %r15
movl $.L.str.11, %edi
232
233
234 movq %r15, %rsi
235 xorl %eax, %eax
     callq __isoc99_scanf
236
237 leaq 264(%rbx), %rdi
238 movq %r15, %rsi
239 callq strcmp
testl %eax, %eax
je .LBB3_7
242 # %bb.6:
243 movl $.Lstr.19, %edi
244 callq puts
245 .LBB3_9:
246 xorl %eax, %eax
247 addq $512, %rsp
                                        # imm = 0x200
.cfi_def_cfa_offset 32
249 popq %rbx
```

```
.cfi_def_cfa_offset 24
    popq %r14
.cfi_def_cfa_offset 16
251
252
253 popq %r15
    .cfi_def_cfa_offset 8
254
255
    retq
256 .LBB3_7:
.cfi_def_cfa_offset 544
258
    movl $.L.str.15, %edi
259 movq %r14, %rsi
260 xorl %eax, %eax
    callq printf
261
262 movl $10, %edi
263 callq putchar
    movl $.L.str.16, %edi
movq %r14, %rsi
xorl %eax, %eax
264
265
266
callq printf
268 movq 520(%rbx), %rsi
269 movl $.L.str.17, %edi
270 jmp .LBB3_8
271 .Lfunc_end3:
.size main, .Lfunc_end3-main
273
    .cfi_endproc
                                            # -- End function
274
                            # @.str
    .type .L.str,@object
275
.section .rodata.str1.1, "aMS", @progbits,1
277 .L.str:
278 .asciz "admin"
    .size .L.str, 6
279
280
    .type .L.str.1,@object
                                # @.str.1
281
282 .L.str.1:
283 .asciz "4dm1n__4eva"
    .size .L.str.1, 12
284
285
    .type .L.str.2,@object
                                   # @.str.2
286
287 .L.str.2:
288 .asciz "alice"
289
    .size .L.str.2, 6
290
291
    .type .L.str.3,@object
292 .L.str.3:
.asciz "!alice12!_veuje@@hak"
    .size .L.str.3, 21
294
295
.type .L.str.4,@object
                                  # @.str.4
297 .L.str.4:
298 .asciz "abdul"
299
     .size .L.str.4, 6
300
    .type .L.str.5,@object
                                  # @.str.5
301
302 .L.str.5:
303 .asciz "passw0rd123"
    .size .L.str.5, 12
304
305
.type .L.str.7,@object # @.str.7
```

```
307 .L.str.7:
308 .asciz " %021d. %s\n"
    .size .L.str.7, 12
309
310
                               # @.str.10
    .type .L.str.10,@object
311
312 .L.str.10:
   .asciz "Username: "
313
    .size .L.str.10, 11
314
315
316
    .type .L.str.11,@object
                                # @.str.11
317 .L.str.11:
318 .asciz "%255s"
    .size .L.str.11, 6
319
320
    .type .L.str.12,@object
                                # @.str.12
321
322 .L.str.12:
.asciz "User < %s > does not exist.\n"
    .size .L.str.12, 29
324
325
    .type .L.str.13,@object
                               # @.str.13
326
327 .L.str.13:
328 .asciz "Password: "
   .size .L.str.13, 11
329
330
    .type .L.str.15,@object
                               # @.str.15
331
332 .L.str.15:
.asciz "Logged in as < s > 1 n"
334 .size .L.str.15, 22
335
    .type .L.str.16,@object
                             # @.str.16
336
337 .L.str.16:
338 .asciz "Welcome, %s!\n"
    .size .L.str.16, 14
339
340
    .type .L.str.17,@object
341
342 .L.str.17:
.asciz "Your balance: $%ld\n"
344
   .size .L.str.17, 20
345
346
    .type .Lstr,@object
347 .Lstr:
348 .asciz "--- USERS ---"
349
    .size .Lstr, 14
350
    .type .Lstr.18,@object
                                 # @str.18
351
352 .Lstr.18:
.asciz "Welcome to BigBank Australia!"
    .size .Lstr.18, 30
354
355
356
    .type .Lstr.19,@object # @str.19
357 .Lstr.19:
.asciz "ERROR: incorrect password"
359
    .size .Lstr.19, 26
360
    .ident "clang version 10.0.0-4ubuntu1 "
361
   .section ".note.GNU-stack","", @progbits
362
363 .addrsig
```

## .6.13 deadStoreElimination-O0.s

```
1 .text
    .file "deadStoreElimination.c"
    .globl deadStore
                                         # -- Begin function deadStore
 p2align 4, 0x90
type deadStore,@function
                                               # @deadStore
 6 deadStore:
 7 .cfi_startproc
 8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
    .cfi_def_cfa_register %rbp
13
_{14} movl %edi, -4(%rbp)
movl %esi, -8(%rbp)

movl $43981, -12(%rbp)

movl $0, -16(%rbp)
                                      # imm = OxABCD
18 .LBB0_1:
                                                # =>This Inner Loop Header: Depth=1
movl -4(\%rbp), \%eax
cmpl -8(\%rbp), \%eax
     ile .LBB0_3
21
22 # %bb.2:
                                                  in Loop: Header=BBO_1 Depth=1
movl -12(%rbp), %eax
24 addl -16(%rbp), %eax
25 movl %eax, -16(%rbp)
26 movl -4(%rbp), %eax
27 addl $-1, %eax
movl \%eax, -4(\%rbp)
30 .LBB0_3:
31 movl $0, -12(%rbp)
32 movl -4(%rbp), %eax
33 addl -8(%rbp), %eax
34 popq %rbp
    .cfi_def_cfa %rsp, 8
35
36
    retq
37 .Lfunc_end0:
.size deadStore, .Lfunc_endO-deadStore
39
    .cfi_endproc
                                               # -- End function
40
    .globl main
                                         # -- Begin function main
41
p2align 4, 0x90
.type main, @function
                                               # @main
44 main:
45 .cfi_startproc
46 # %bb.0:
47 pushq %rbp
    .cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
   movq %rsp, %rbp
50
51
    .cfi_def_cfa_register %rbp
52 subq $32, %rsp
53 movl %edi, -4(%rbp)
54 movq %rsi, -16(%rbp)
55 movl -4(%rbp), %edi
```

```
56 movl $2, %esi
    callq deadStore
   xorl %ecx, %ecx
58
59 movl %eax, -20(%rbp)
                                 # 4-byte Spill
60 movl %ecx, %eax
61 addq $32, %rsp
62 popq %rbp
    .cfi_def_cfa %rsp, 8
63
64 retq
65 .Lfunc_end1:
.size main, .Lfunc_end1-main
67
    .cfi_endproc
                                          # -- End function
68
.ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","", @progbits
.addrsig.addrsig_sym deadStore
```

# .6.14 deadStoreElimination-O3.s

```
1 .text
.file "deadStoreElimination.c"
3 .globl deadStore
                                   # -- Begin function deadStore
.p2align 4, 0x90.type deadStore, @function
                                         # @deadStore
6 deadStore:
7 .cfi_startproc
8 # %bb.0:
                                         # kill: def $esi killed $esi def $rsi
9
                                         # kill: def $edi killed $edi def $rdi
10
cmpl %edi, %esi
cmovlel %esi, %edi
13 leal (%rdi,%rsi), %eax
14 retq
15 .Lfunc_end0:
.size deadStore, .Lfunc_end0-deadStore
.cfi_endproc
                                        # -- End function
18
.globl main
.p2align 4, 0x90
                                   # -- Begin function main
21 .type main, @function
                                       # @main
22 main:
.cfi_startproc
24 # %bb.0:
25 xorl %eax, %eax
26 retq
27 .Lfunc_end1:
.size main, .Lfunc_end1-main
29
   .cfi_endproc
                                        # -- End function
30
.ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","",@progbits
33 .addrsig
```

## .6.15 pread-O0.s

```
1 .text
```

```
2 .file "pread.c"
   .globl main
                                    # -- Begin function main
   .p2align 4, 0x90
5 .type main, @function
                                          # @main
6 main:
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
   .cfi_offset %rbp, -16
11
   movq %rsp, %rbp
12
   .cfi_def_cfa_register %rbp
13
14 \quad movl \quad $0, -4(%rbp)
15 movl $0, -8(%rbp)
16 movl $0, -12(%rbp)
17 .LBB0_1:
                                          # =>This Loop Header: Depth=1
                                             Child Loop BBO_2 Depth 2
18
                                                 Child Loop BBO_3 Depth 3
19
21 .LBB0_2:
                                          # Parent Loop BB0_1 Depth=1
                                          # => This Loop Header: Depth=2
                                                 Child Loop BBO_3 Depth 3
   jmp .LBB0_3
24
25 .LBB0_3:
                                          #
                                            Parent Loop BBO_1 Depth=1
                                               Parent Loop BBO_2 Depth=2
26
27
                                                 This Inner Loop Header: Depth=3
movl z, %eax
29 movl %eax, -8(%rbp)
30 # %bb.4:
                                          # in Loop: Header=BB0_3 Depth=3
   movl -8(\%rbp), \%eax
31
32
    cltd
   movl $2, %ecx
33
   idivl %ecx
35 cmpl \$0, \%edx
    jne .LBB0_3
36
37 # %bb.5:
                                          # in Loop: Header=BBO_2 Depth=2
movl x, %eax
39 movl %eax, -12(%rbp)
40 # %bb.6:
                                             in Loop: Header=BBO_2 Depth=2
41 movl z, %eax
42 cmpl -8(%rbp), %eax
jne .LBB0_2
44 # %bb.7:
                                          # in Loop: Header=BBO_1 Depth=1
45 jmp .LBB0_1
46 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
48
                                          # -- End function
49
   .type z,@object
                                 # @z
50
51
    .comm z,4,4
   .type x,@object
                                  # @x
52
   .comm x,4,4
53
   .ident "clang version 10.0.0-4ubuntu1"
    .section ".note.GNU-stack","", @progbits
55
    .addrsig
   .addrsig_sym z
57
.addrsig_sym x
```

# .6.16 pread-O3.s

```
1 .text
.file "pread.c"
.glob1 main
.p2align 4, 0x90
                                         # -- Begin function main
 5 .type main, @function
                                                # @main
6 main:
 7 .cfi_startproc
 8 # %bb.0:
9 .p2align 4, 0x90
10 .LBB0_1:
                                                # =>This Inner Loop Header: Depth=1
movl z(%rip), %eax
testb $1, %al
jne .LBBO_1
14 # %bb.2:
                                              # in Loop: Header=BBO_1 Depth=1
movl x(%rip), %eax movl z(%rip), %eax
    jmp .LBB0_1
18 .Lfunc_end0:
size main, .Lfunc_end0-main
cfi_endproc
                                              # -- End function
21
.type z,@object
.comm z,4,4
                                    # @z
                                      # @x
24 .type x,@object
.comm x,4,4
    .ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","",@progbits
26
27
28 .addrsig
29 .addrsig_sym z
30 .addrsig_sym x
```