

# C Annotations for Concurrent Information Flow Security

Alexander Blyth

June 2021

**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.

## 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 Winter 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*.

## 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

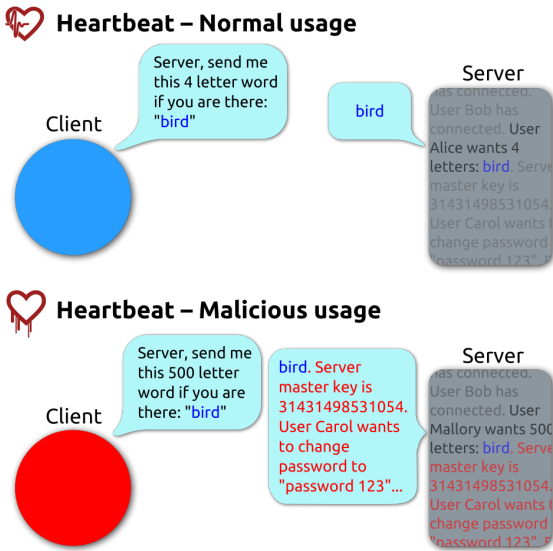


Figure 1: The Heartbleed bug. [13]

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].

## 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, and that the system can perform tasks without interference from outside sources. Finally, availability of a system requires that service is not denied to au-

thorised 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 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,

```

1 secret := 0xC0DE mod 2
2 public := 1
3 if secret = 1
4     public := 0
5

```

Figure 2: Implicit flow of data to a public variable

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].

## 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 or 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

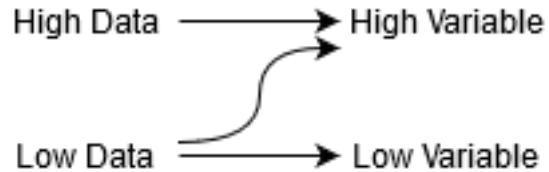


Figure 3: Permitted flow of data

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.

## 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 pos-

sibilities 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 non-blocking concurrent threads, Winter et al. [31] explores verifying security properties such as non-interference 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.

## 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 ex-

```

1 crypt() {
2     key := 0xC0DE // Read key
3     ... // Work with the key
4     key := 0x0 // Clear memory
5 }
6

```

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

ecuted. 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 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 counter-initiative. 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.

## 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 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 higher-level 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.

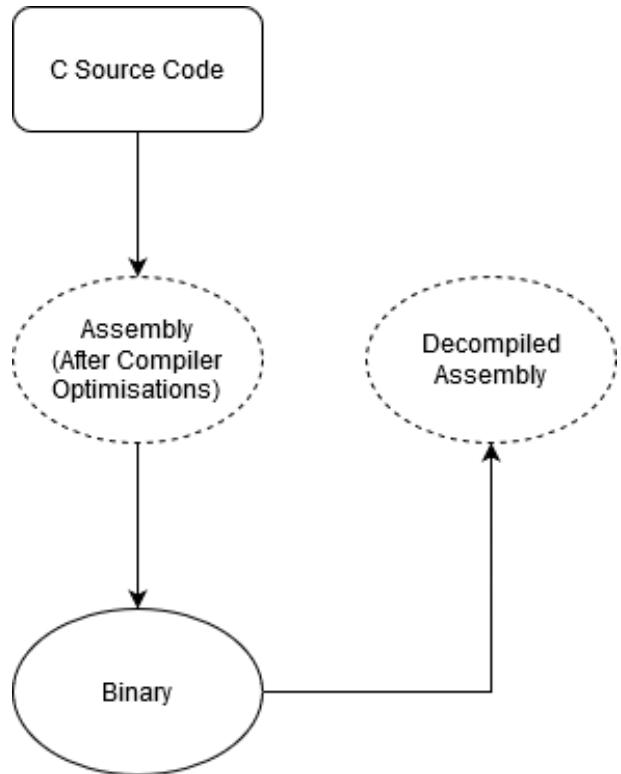


Figure 5: The static analysis options after compilation.

## 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 annotations are not limited to WCET analysis. A 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 through 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 properties 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 and memory locations. If an optimisation pass had caused the verification to fail the programmer would then be notified, to which they could annotate differently or disable the optimisation.

### 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 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 3.2 and 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 3.3.

#### 3.1 Test Cases

A suite of test C programs (See Appendix A) 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 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.

### 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.

### 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

$$h(n) \in O(g(n)), \text{ and}$$

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

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.

### 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.

## 4 Execution

Experimentation began with the CompCert compiler and the provided assembly annotation tools, outlined in section 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 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.

### 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

OS Name	Ubuntu 20.04.2 LTS
OS Type	64-bit
Processor	Intel® Core™ i7-6700K CPU @ 4.00GHz × 8
Instruction Set	x86-64
CompCert Version	The CompCert C verified compiler, version 3.7

Table 1: CompCert install specifications

builtin. This assembly annotation is created through the use of the `__builtin_annot` function described in 2.6. The following builtin annotation was placed in line 2, within the main function in *comment.c*.

```
2  __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 -O0 -S
```

Here -O0 is used to specify to perform no optimisations during compilation. The full compiled output can be seen in Appendix D. 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 annotations are unable to handle volatile variables, generating the follow error upon compiling *volatile.c*.

```
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.

```
84 .cfi_endproc
85 .type main, @function
86 .size main, . - main
87 .section
    "__compcert_ais_annotations","",@note
88 .ascii "# file:rooster.c line:6
    function:fun\n"
89 .byte 7,8
90 .quad .L100
91 .ascii " CRITICAL COMMENT\n"
92 .ascii "# file:rooster.c line:26
    function:main\n"
93 .byte 7,8
94 .quad .L107
95 .ascii " L(mem("
96 .byte 7,8
97 .quad goose
98 .ascii ", 4)) = medium\n"
99 .ascii "# file:rooster.c line:27
    function:main\n"
100 .byte 7,8
101 .quad .L108
```

```
102 .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, 4)) = medium` has been preserved. Thus, the CompCert annotations can successfully preserve annotations on global variables.

Another interesting problem faced when working with CompCert AIS annotations is found when working with structs. If the programmer wants to annotate a member of a struct for all structs of that type, each instance of that type of struct must be annotated when using CompCert AIS annotations. This is because CompCert treats `__builtin_ais_annot()` as a call to an external function. 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.

```
17 user_t* user_admin =
   malloc(sizeof(user_t));
18 strcpy(user_admin->name, "admin");
19 strcpy(user_admin->password,
   "4dmin__4eva");
20 __builtin_ais_annot("%here L(%e1) =
   high", user_admin->password);
21 user_admin->balance = 1000000;
22
23 user_t* user_alice =
   malloc(sizeof(user_t));
24 strcpy(user_alice->name, "alice");
25 strcpy(user_alice->password,
   "!alice12!_veuje@0hak");
26 __builtin_ais_annot("%here L(%e1) =
   high", user_alice->password);
27 user_alice->balance = 783;
28
29 user_t* user_abdul =
   malloc(sizeof(user_t));
30 strcpy(user_abdul->name, "abdul");
31 strcpy(user_abdul->password,
   "passw0rd123");
32 __builtin_ais_annot("%here L(%e1) =
   high", user_abdul->password);
```

```
33 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.

```
320 .section
   "__compcert_ais_annotations","",@note
321 .ascii "# file:password.c line:20
   function:setup_users\n"
322 .byte 7,8
323 .quad .L100
324 .ascii " L((reg(\"rbp\") + 264)) = high\n"
325 .ascii "# file:password.c line:26
   function:setup_users\n"
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"
```

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.

#### 4.1.1 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 C. 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:

```
1 2c2
2 < # Command line: comment.c -S -O3 -o
   compCert/out/comment-03.s
3 ---
4 > # Command line: comment.c -O3 -S
```

```

5 11a12
6 > .L100:
7 17a19,23
8 > .section
   "__compcert_ais_annotaions","",@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.

```

1 2c2
2 < # Command line: variable.c -S -O3 -o
   compCert/out/variable-O3.s
3 ---
4 > # Command line: variable.c -S -O3
5 11a12,16
6 > .L100:
7 > .L101:
8 > .L102:
9 > .L103:
10 > .L104:
11 17a23,43
12 > .section
   "__compcert_ais_annotaions","",@note
13 > .ascii "# file:variable.c line:3
   function:main\n"
14 > .byte 7,8
15 > .quad .L100
16 > .ascii " -10 = int\n"
17 > .ascii "# file:variable.c line:5
   function:main\n"
18 > .byte 7,8
19 > .quad .L101
20 > .ascii " 98 = char\n"
21 > .ascii "# file:variable.c line:7
   function:main\n"
22 > .byte 7,8
23 > .quad .L102
24 > .ascii " -98 = unsigned int\n"
25 > .ascii "# file:variable.c line:9
   function:main\n"
26 > .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*

```

1 int main() {
2     int count = 0;
3     // here count is always zero
4     for(int i = 0; i < count; i++) {
5         __builtin_ais_annot("try loop
   %here bound: %e1;", count);
6     }
7
8     return 0;
9 }

```

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.

```

1 # File generated by CompCert 3.7
2 # Command line: count.c -S -O0 -o
   annotated/count-O0.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 xorl %ecx, %ecx
13 xorl %eax, %eax
14 .L100:
15 cmpl %ecx, %eax
16 jge .L101
17 .L102:
18 leal 1(%eax), %eax
19 jmp .L100
20 .L101:
21 xorl %eax, %eax
22 addq $8, %rsp
23 ret
24 .cfi_endproc
25 .type main, @function
26 .size main, . - main
27 .section
   "__compcert_ais_annotations","",@note
28 .ascii "# file:count.c line:5
   function:main\ntry loop "
29 .byte 7,8
30 .quad .L102
31 .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*.

```

1 # File generated by CompCert 3.7
2 # Command line: count.c -S -O3 -o
   annotated/count-O3.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)

```

```

12 xorl %eax, %eax
13 addq $8, %rsp
14 ret
15 .cfi_endproc
16 .type main, @function
17 .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.

## 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 E.

```

2 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 F.

```

6 main:                                     #
    @main
7   .cfi_startproc
8   # %bb.0:
9   pushq %rbp
10  .cfi_def_cfa_offset 16
11  .cfi_offset %rbp, -16
12  movq %rsp, %rbp
13  .cfi_def_cfa_register %rbp
14  movl $0, -4(%rbp)
15  #APP
16  # CRITICAL COMMENT
17  #NO_APP
18  xorl %eax, %eax
19  popq %rbp
20  .cfi_def_cfa %rsp, 8
21  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:

```

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

```

Following, the compiled output would be:

```

15  #APP
16  # annotation: CRITICAL COMMENT
17  #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 `asm`

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:

```



```

9      kill: def $edi killed $edi def $rdi #
10     movl  $-10, -4(%rsp)
11     #APP
12     # annotation: -4(%rsp) = int
13     #NO_APP
14     movl  -4(%rsp), %eax
15     addl  %edi, %eax
16     addl  $134217635, %eax      # imm =
17     retq  0x7FFFFFFA3

```

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 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 16. 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.

```

23     movl  $0, -4(%rbp)
24     movl  %edi, -8(%rbp)
25     movq  %rsi, -16(%rbp)
26     movl  $-10, -20(%rbp)
27     #APP
28     # annotation: -20(%rbp) = int
29     #NO_APP
30     movsd .LCPI0_0(%rip), %xmm0 # xmm0 =
31     mem[0],zero
32     movss .LCPI0_1(%rip), %xmm1 # xmm1 =
33     mem[0],zero,zero,zero
34     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 16.

```

6     main:
7     # @main
8     .cfi_startproc
9     # %bb.0:
10
11     # kill: def $edi killed $edi def $rdi
12     movl  $-10, -4(%rsp)
13     #APP
14     # annotation: -4(%rsp) = int
15     #NO_APP
16     movl  -4(%rsp), %eax
17     addl  %edi, %eax
18     addl  $134217635, %eax      # imm
19     = 0x7FFFFFFA3
20     retq

```

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  $\mathbf{X}(\mathbf{a})$  specifies that any operand whatsoever is allowed. The resulting optimised assembly is as follows.



1. Install specifications
2. Insert variable inline assembly
3. Volatile variable
4. Floats
5. Discuss implications of additional move statements
6. Quality Analysis -
7. speed comparison

Tool building:

1. af
2. escaping special strings
3. Source Code documentation

### 4.3 CompCert Builtin Annotations

TODO: Revisit compcert using builtin annotations rather than AIS annotations

### 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 4.2. Therefore, evaluating compiler modification for static analysis purposes was not performed in this research.

## References

- [1] Pieter Agten et al. “Recent developments in low-level software security”. In: *IFIP International 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](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] Sundeeep 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

## A Test C Programs

### A.1 comment.c

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

### A.2 variable.c

```
1 int main(int argc, char* argv[]) {  
2     // a = int  
3     // b = char  
4     // c = unsigned int  
5     // d = short  
6     // e = long  
7     // x = float  
8     // y = double  
9     int a = -10;  
10    char b = 'b';  
11    unsigned int c = -b;  
12    short d = 0x1;  
13    long e = 4294967296;  
14    float x = 3.141592653589793;  
15    double y = x / 2.3784;  
16    return (int)(e / 32) + (int)a + (int)c + (int)d + (int)x + (int)y + argc;  
17 }
```

### A.3 volatile.c

```
1 volatile int x;  
2  
3 int main() {  
4     // L(x) = High  
5     return x + 1;  
6 }
```

### A.4 loop.c

```
1 int z;  
2 int x;  
3  
4 // security policies  
5 // {L(z)=true}  
6 // {L(x)=z % 2 == 0}  
7  
8 // predicates on initial state  
9 // {_P_0: r1 % 2 == 0}  
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}  
11  
12 int main() {
```

```

13  int r1 = 0;
14  // {L(r2)=False}
15  int r2 = 0;
16
17  while(1) {
18  do {
19      // {_invariant: r1 % 2 == 0 /\ r1 <= z}
20      // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
21      do {
22          // {_invariant: r1 <= z}
23          // {_Gamma: r1 -> LOW}
24          r1 = z;
25      } while (r1 %2 != 0);
26      r2 = x;
27      } while (z != r1);
28  }
29  return r2;
30 }

```

## A.5 rooster.c

```

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

```

## A.6 password.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define BUFF_LEN 256
6
7 typedef struct user_t user_t;
8
9 struct user_t {
10     user_t* next;
11     char name[BUFF_LEN];
12     // L(password) = High
13     char password[BUFF_LEN];
14     size_t balance;
15 };
16
17 user_t* setup_users() {
18     user_t* user_admin = malloc(sizeof(user_t));
19     strcpy(user_admin->name, "admin");
20     strcpy(user_admin->password, "4dmin__4eva");
21     user_admin->balance = 1000000;
22
23     user_t* user_alice = malloc(sizeof(user_t));
24     strcpy(user_alice->name, "alice");
25     strcpy(user_alice->password, "!alice12!_veuje@@hak");
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     user_abdul->balance = 2;
32
33     user_admin->next = user_alice;
34     user_alice->next = user_abdul;
35     user_abdul->next = NULL;
36
37     return user_admin;
38 }
39
40 void print_users(user_t* users) {
41     printf("--- USERS ---\n");
42     size_t count = 0;
43     while (users != NULL) {
44         printf(" %02ld. %s\n", ++count, users->name);
45         users = users->next;
46     }
47     printf("\n");
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
51     while (user_list != NULL) {
52         if (strcmp(user_list->name, name) == 0) {
53             return user_list;
54         }
55         user_list = user_list->next;
56     }
57     return NULL;

```

```

58 }
59
60 int main() {
61     user_t* users = setup_users();
62
63     printf("Welcome to BigBank Australia!\n");
64
65     char username[BUFF_LEN];
66     printf("Username: ");
67     scanf("%255s", username);
68
69     user_t* user = getUser(users, username);
70     if (user == NULL) {
71         printf("User < %s > does not exist.\n", username);
72         return 0;
73     }
74
75     char password[BUFF_LEN];
76     printf("Password: ");
77     scanf("%255s", password);
78     if (strcmp(user->password, password) != 0) {
79         printf("ERROR: incorrect password\n");
80         return 0;
81     }
82
83     printf("Logged in as < %s >!\n", user->name);
84     printf("\n");
85     printf("Welcome, %s!\n", user->name);
86     printf("Your balance: $%ld\n", user->balance);
87 }

```

## A.7 deadStoreElimination.c

```

1 int deadStore(int i, int n) {
2     int key = 0xabcd;
3     // L(key) = high
4
5     // do some work
6     int result = 0;
7     while (i > n) {
8         result += key;
9         i--;
10    }
11
12    // clear out our secret key
13    key = 0;
14    return i + n;
15 }
16
17 int main(int argc, char *argv[]) {
18     deadStore(argc, 2);
19 }

```

## A.8 pread.c

```

1 volatile int z;
2 volatile int x;

```

```

3
4 // security policies
5 // {L(z)=true}
6 // {L(x)=z % 2 == 0}
7
8 // predicates on initial state
9 // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
13     int r1 = 0;
14     // {L(r2)=False}
15     int r2 = 0;
16
17     while(1) {
18     do {
19         // {_invariant: r1 % 2 == 0 /\ r1 <= z}
20         // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
21         do {
22             // {_invariant: r1 <= z}
23             // {_Gamma: r1 -> LOW}
24             r1 = z;
25         } while (r1 % 2 != 0);
26         r2 = x;
27     } while (z != r1);
28     }
29     return r2;
30 }

```

## B CompCert Annotated C Programs

### B.1 comment.c

```

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

```

### B.2 variable.c

```

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

```



```
17 }
```

### B.3 volatile.c

```
1 volatile int x;
2
3 int main() {
4     __builtin_ais_annot("%here L(%e1)= false", x);
5     return x + 1;
6 }
```

### B.4 loop.c

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

### B.5 rooster.c

```
1 int rooster;
2 int drake;
3 int goose;
4
5 int fun(int a, int b, int c) {
6     __builtin_ais_annot("%here CRITICAL COMMENT");
7     static int count = 0;
8     int sum = a + b + c;
9     if (sum < 0) {
10         return sum;
11     }
```

```

11     }
12     if (a < b && b < c) {
13         while (a != b) {
14             a++;
15             count++;
16             while (b != c) {
17                 c--;
18                 count++;
19             }
20         }
21     }
22     return count;
23 }
24
25 int main(void) {
26     __builtin_ais_annot("%here L(%e1) = medium", goose);
27     __builtin_ais_annot("%here EXCEPTIONAL");
28     rooster = 1;
29     drake = 5;
30     goose = 10;
31     int result;
32     result = fun(rooster, drake, goose);
33     return 0;
34 }

```

## B.6 password.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define BUFF_LEN 256
6
7 typedef struct user_t user_t;
8
9 struct user_t {
10     user_t* next;
11     char name[BUFF_LEN];
12     char password[BUFF_LEN];           // { L(password) = High }
13     size_t balance;
14 };
15
16 user_t* setup_users() {
17     user_t* user_admin = malloc(sizeof(user_t));
18     strcpy(user_admin->name, "admin");
19     strcpy(user_admin->password, "4dmin_4eva");
20     __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
21     user_admin->balance = 1000000;
22
23     user_t* user_alice = malloc(sizeof(user_t));
24     strcpy(user_alice->name, "alice");
25     strcpy(user_alice->password, "!alice12!_veuje@@hak");
26     __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
27     user_alice->balance = 783;
28
29     user_t* user_abdul = malloc(sizeof(user_t));
30     strcpy(user_abdul->name, "abdul");

```

```

31 strcpy(user_abdul->password, "passw0rd123");
32 __builtin_ais_annot("%here L(%e1) = high", user_abdul->password);
33 user_abdul->balance = 2;
34
35 user_admin->next = user_alice;
36 user_alice->next = user_abdul;
37 user_abdul->next = NULL;
38
39 return user_admin;
40 }
41
42 void print_users(user_t* users) {
43     printf("--- USERS ---\n");
44     size_t count = 0;
45     while (users != NULL) {
46         printf(" %02ld. %s\n", ++count, users->name);
47         users = users->next;
48     }
49     printf("\n");
50 }
51
52 user_t* getUser(user_t* user_list, char* name) {
53     while (user_list != NULL) {
54         if (strcmp(user_list->name, name) == 0) {
55             return user_list;
56         }
57         user_list = user_list->next;
58     }
59     return NULL;
60 }
61
62 int main() {
63     user_t* users = setup_users();
64
65     printf("Welcome to BigBank Australia!\n");
66
67     char username[BUFF_LEN];
68     printf("Username: ");
69     scanf("%255s", username);
70
71     user_t* user = getUser(users, username);
72     if (user == NULL) {
73         printf("User < %s > does not exist.\n", username);
74         return 0;
75     }
76
77     char password[BUFF_LEN];
78     printf("Password: ");
79     scanf("%255s", password);
80     if (strcmp(user->password, password) != 0) {
81         printf("ERROR: incorrect password\n");
82         return 0;
83     }
84
85     printf("Logged in as < %s >!\n", user->name);
86     printf("\n");
87     printf("Welcome, %s!\n", user->name);

```

```

88     printf("Your balance: %ld\n", user->balance);
89 }

```

## B.7 deadStoreElimination.c

```

1  int deadStore(int i, int n) {
2      int key = 0xabcd;
3      __builtin_ais_annot("%here L(%e1) = high", key);
4
5      // do some work
6      int result = 0;
7      while (i > n) {
8          result += key;
9          i--;
10     }
11
12     // clear out our secret key
13     key = 0;
14     return i + n;
15 }
16
17 int main(int argc, char *argv[]) {
18     deadStore(argc, 2);
19 }

```

## B.8 pread.c

```

1  volatile int z;
2  volatile int x;
3
4  int main() {
5      // Security Policies
6      __builtin_ais_annot("%here L(%e1) = true", z);
7      __builtin_ais_annot("%here L(%e1)= %e2 %% 2 == 0", x, z);
8
9      int r1 = 0;
10     int r2 = 0; // {L(r2)=False}
11     __builtin_ais_annot("%here L(%e1)= false", r2);
12
13     // Predicates on initial state
14     __builtin_ais_annot("%here _P_0: %e1 %% 2 == 0", r1);
15     __builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1, r2);
16
17
18     while(1) {
19         do {
20             __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1 <= %e2", r1, z);
21             __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1 == %e3), %e3 ->
22             LOW", r1, r2, z);
23             do {
24                 __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
25                 __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
26                 r1 = z;
27             } while (r1 % 2 != 0);
28             r2 = x;
29         } while (z != r1);
30     }
31 }

```

```
30     return r2;
31 }
```

## C CompCert Assembly Output

### C.1 comment-O0.s

```
1 # File generated by CompCert 3.7
2 # Command line: comment.c -S -O0 -o compCert/out/comment-O0.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 xorl %eax, %eax
13 addq $8, %rsp
14 ret
15 .cfi_endproc
16 .type main, @function
17 .size main, . - main
```

### C.2 comment-O3.s

```
1 # File generated by CompCert 3.7
2 # Command line: comment.c -S -O3 -o compCert/out/comment-O3.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 xorl %eax, %eax
13 addq $8, %rsp
14 ret
15 .cfi_endproc
16 .type main, @function
17 .size main, . - main
```

### C.3 variable-O0.s

```
1 # File generated by CompCert 3.7
2 # Command line: variable.c -S -O0 -o compCert/out/variable-O0.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
```

```

10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 movl $-10, %esi
13 movl $98, %ecx
14 negl %ecx
15 movl $1, %r8d
16 movabsq $4294967296, %rax
17 movsd .L100(%rip), %xmm1 # 3.14159265358979312
18 cvtsd2ss %xmm1, %xmm3
19 cvtss2sd %xmm3, %xmm0
20 movsd .L101(%rip), %xmm2 # 2.37840000000000007
21 divsd %xmm2, %xmm0
22 cqto
23 shrq $59, %rdx
24 leaq 0(%rax,%rdx,1), %rax
25 sarq $5, %rax
26 leal 0(%eax,%esi,1), %r9d
27 leal 0(%r9d,%ecx,1), %r10d
28 leal 0(%r10d,%r8d,1), %r8d
29 cvttss2si %xmm3, %edx
30 leal 0(%r8d,%edx,1), %r8d
31 cvttss2si %xmm0, %eax
32 leal 0(%r8d,%eax,1), %r11d
33 leal 0(%r11d,%edi,1), %eax
34 addq $8, %rsp
35 ret
36 .cfi_endproc
37 .type main, @function
38 .size main, . - main
39 .section .rodata.cst8,"aM",@progbits,8
40 .align 8
41 .L100: .quad 0x400921fb54442d18
42 .L101: .quad 0x400306f694467382

```

## C.4 variable-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: variable.c -S -O3 -o compCert/out/variable-O3.s
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 leal 134217625(%edi), %eax
13 addq $8, %rsp
14 ret
15 .cfi_endproc
16 .type main, @function
17 .size main, . - main

```

## C.5 volatile-O0.s

```

1 # File generated by CompCert 3.7

```

```

2 # Command line: volatile.c -S -O0 -o compCert/out/volatile-00.s
3 .comm x, 4, 4
4 .text
5 .align 16
6 .globl main
7 main:
8 .cfi_startproc
9 subq $8, %rsp
10 .cfi_adjust_cfa_offset 8
11 leaq 16(%rsp), %rax
12 movq %rax, 0(%rsp)
13 movl x(%rip), %eax
14 leal 1(%eax), %eax
15 addq $8, %rsp
16 ret
17 .cfi_endproc
18 .type main, @function
19 .size main, . - main

```

## C.6 volatile-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: volatile.c -S -O3 -o compCert/out/volatile-O3.s
3 .comm x, 4, 4
4 .text
5 .align 16
6 .globl main
7 main:
8 .cfi_startproc
9 subq $8, %rsp
10 .cfi_adjust_cfa_offset 8
11 leaq 16(%rsp), %rax
12 movq %rax, 0(%rsp)
13 movl x(%rip), %eax
14 leal 1(%eax), %eax
15 addq $8, %rsp
16 ret
17 .cfi_endproc
18 .type main, @function
19 .size main, . - main

```

## C.7 loop-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: loop.c -S -O0 -o compCert/out/loop-O0.s
3 .comm z, 4, 4
4 .comm x, 4, 4
5 .text
6 .align 16
7 .globl main
8 main:
9 .cfi_startproc
10 subq $8, %rsp
11 .cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:

```

```

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

```

## C.8 loop-O3.s

```

1  # File generated by CompCert 3.7
2  # Command line: loop.c -S -O3 -o compCert/out/loop-O3.s
3  .comm z, 4, 4
4  .comm x, 4, 4
5  .text
6  .align 16
7  .globl main
8  main:
9  .cfi_startproc
10  subq  $8, %rsp
11  .cfi_adjust_cfa_offset 8
12  leaq  16(%rsp), %rax
13  movq  %rax, 0(%rsp)
14 .L100:
15  movl  z(%rip), %edx
16  movq  %rdx, %rax
17  testl %eax, %eax
18  leal  1(%eax), %ecx
19  cmovl %rcx, %rax
20  sarl  $1, %eax
21  leal  0(,%eax,2), %esi
22  movq  %rdx, %rcx
23  subl  %esi, %ecx
24  testl %ecx, %ecx
25  jne   .L100
26  movq  %rdx, %rsi
27  jmp   .L100
28  .cfi_endproc
29  .type main, @function
30  .size main, . - main

```

## C.9 rooster-O0.s

```

1  # File generated by CompCert 3.7
2  # Command line: rooster.c -S -O0 -o compCert/out/rooster-O0.s
3  .comm rooster, 4, 4
4  .comm drake, 4, 4
5  .comm goose, 4, 4

```



```

6  .data
7  .align 4
8  count:
9  .long 0
10 .type count, @object
11 .size count, . - count
12 .text
13 .align 16
14 .globl fun
15 fun:
16 .cfi_startproc
17 subq $8, %rsp
18 .cfi_adjust_cfa_offset 8
19 leaq 16(%rsp), %rax
20 movq %rax, 0(%rsp)
21 leal 0(%edi,%esi,1), %r8d
22 leal 0(%r8d,%edx,1), %eax
23 testl %eax, %eax
24 jl .L100
25 cmpl %esi, %edi
26 jl .L101
27 xorl %r8d, %r8d
28 jmp .L102
29 .L101:
30 cmpl %edx, %esi
31 setl %r8b
32 movzbl %r8b, %r8d
33 .L102:
34 cmpl $0, %r8d
35 je .L103
36 .L104:
37 cmpl %esi, %edi
38 je .L103
39 leal 1(%edi), %edi
40 movl count(%rip), %eax
41 leal 1(%eax), %ecx
42 movl %ecx, count(%rip)
43 .L105:
44 cmpl %edx, %esi
45 je .L104
46 leal -1(%edx), %edx
47 movl count(%rip), %r9d
48 leal 1(%r9d), %r8d
49 movl %r8d, count(%rip)
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
66 leaq 16(%rsp), %rax
67 movq %rax, 0(%rsp)
68 movl $1, %eax
69 movl %eax, rooster(%rip)
70 movl $5, %eax
71 movl %eax, drake(%rip)
72 movl $10, %eax
73 movl %eax, goose(%rip)
74 movl rooster(%rip), %edi
75 movl drake(%rip), %esi
76 movl goose(%rip), %edx
77 call fun
78 xorl %eax, %eax
79 addq $8, %rsp
80 ret
81 .cfi_endproc
82 .type main, @function
83 .size main, . - main

```

## C.10 rooster-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: rooster.c -S -O3 -o compCert/out/rooster-O3.s
3 .comm rooster, 4, 4
4 .comm drake, 4, 4
5 .comm goose, 4, 4
6 .data
7 .align 4
8 count:
9 .long 0
10 .type count, @object
11 .size count, . - count
12 .text
13 .align 16
14 .globl fun
15 fun:
16 .cfi_startproc
17 subq $8, %rsp
18 .cfi_adjust_cfa_offset 8
19 leaq 16(%rsp), %rax
20 movq %rax, 0(%rsp)
21 leal 0(%edi,%esi,1), %r9d
22 leal 0(%r9d,%edx,1), %eax
23 testl %eax, %eax
24 jl .L100
25 cmpl %edx, %esi
26 setl %al
27 movzbl %al, %eax
28 xorl %r8d, %r8d
29 cmpl %esi, %edi
30 cmovge %r8, %rax
31 cmpl $0, %eax
32 je .L101
33 .L102:

```

```

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

```

## C.11 password-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: password.c -S -O0 -o compCert/out/password-O0.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:
12 .ascii "passwOrd123\000"
13 .type __stringlit_6, @object
14 .size __stringlit_6, . - __stringlit_6
15 .section .rodata
16 .align 1
17 __stringlit_4:
18 .ascii "!alice12!_veuje@@hak\000"
19 .type __stringlit_4, @object
20 .size __stringlit_4, . - __stringlit_4
21 .section .rodata
22 .align 1
23 __stringlit_14:
24 .ascii "Password: \000"
25 .type __stringlit_14, @object
26 .size __stringlit_14, . - __stringlit_14
27 .section .rodata
28 .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%ld\012\000"
31 .type __stringlit_18, @object
32 .size __stringlit_18, . - __stringlit_18
33 .section .rodata
34 .align 1
35 __stringlit_13:
36 .ascii "User < %s > does not exist.\012\000"
37 .type __stringlit_13, @object
38 .size __stringlit_13, . - __stringlit_13
39 .section .rodata
40 .align 1
41 __stringlit_8:
42 .ascii " %02ld. %s\012\000"
43 .type __stringlit_8, @object
44 .size __stringlit_8, . - __stringlit_8
45 .section .rodata
46 .align 1
47 __stringlit_1:
48 .ascii "admin\000"
49 .type __stringlit_1, @object
50 .size __stringlit_1, . - __stringlit_1
51 .section .rodata
52 .align 1
53 __stringlit_2:
54 .ascii "4dmin__4eva\000"
55 .type __stringlit_2, @object
56 .size __stringlit_2, . - __stringlit_2
57 .section .rodata
58 .align 1
59 __stringlit_3:
60 .ascii "alice\000"
61 .type __stringlit_3, @object
62 .size __stringlit_3, . - __stringlit_3
63 .section .rodata
64 .align 1

```

```

65 __stringlit_11:
66     .ascii  "Username: \000"
67     .type __stringlit_11, @object
68     .size __stringlit_11, . - __stringlit_11
69     .section .rodata
70     .align 1
71 __stringlit_5:
72     .ascii  "abdul\000"
73     .type __stringlit_5, @object
74     .size __stringlit_5, . - __stringlit_5
75     .section .rodata
76     .align 1
77 __stringlit_17:
78     .ascii  "Welcome, %s!\012\000"
79     .type __stringlit_17, @object
80     .size __stringlit_17, . - __stringlit_17
81     .section .rodata
82     .align 1
83 __stringlit_12:
84     .ascii  "%255s\000"
85     .type __stringlit_12, @object
86     .size __stringlit_12, . - __stringlit_12
87     .section .rodata
88     .align 1
89 __stringlit_9:
90     .ascii  "\012\000"
91     .type __stringlit_9, @object
92     .size __stringlit_9, . - __stringlit_9
93     .section .rodata
94     .align 1
95 __stringlit_15:
96     .ascii  "ERROR: incorrect password\012\000"
97     .type __stringlit_15, @object
98     .size __stringlit_15, . - __stringlit_15
99     .section .rodata
100    .align 1
101 __stringlit_10:
102    .ascii  "Welcome to BigBank Australia!\012\000"
103    .type __stringlit_10, @object
104    .size __stringlit_10, . - __stringlit_10
105    .section .rodata
106    .align 1
107 __stringlit_16:
108    .ascii  "Logged in as < %s >!\012\000"
109    .type __stringlit_16, @object
110    .size __stringlit_16, . - __stringlit_16
111    .text
112    .align 16
113    .globl setup_users
114 setup_users:
115    .cfi_startproc
116    subq $40, %rsp
117    .cfi_adjust_cfa_offset 40
118    leaq 48(%rsp), %rax
119    movq %rax, 0(%rsp)
120    movq %rbx, 8(%rsp)
121    movq %rbp, 16(%rsp)

```

```

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

```

```

179  movq  %rbp, 16(%rsp)
180  movq  %rdi, %rbx
181  leaq  __stringlit_7(%rip), %rdi
182  movl  $0, %eax
183  call  printf
184  xorq  %rbp, %rbp
185  .L100:
186  cmpq  $0, %rbx
187  je    .L101
188  leaq  1(%rbp), %rbp
189  leaq  __stringlit_8(%rip), %rdi
190  leaq  8(%rbx), %rdx
191  movq  %rbp, %rsi
192  movl  $0, %eax
193  call  printf
194  movq  0(%rbx), %rbx
195  jmp   .L100
196  .L101:
197  leaq  __stringlit_9(%rip), %rdi
198  movl  $0, %eax
199  call  printf
200  movq  8(%rsp), %rbx
201  movq  16(%rsp), %rbp
202  addq  $24, %rsp
203  ret
204  .cfi_endproc
205  .type print_users, @function
206  .size print_users, . - print_users
207  .text
208  .align 16
209  .globl getUser
210  getUser:
211  .cfi_startproc
212  subq  $24, %rsp
213  .cfi_adjust_cfa_offset 24
214  leaq  32(%rsp), %rax
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
222  je    .L103
223  leaq  8(%rbx), %rdi
224  movq  %rbp, %rsi
225  call  strcmp
226  testl %eax, %eax
227  je    .L104
228  movq  0(%rbx), %rbx
229  jmp   .L102
230  .L103:
231  xorq  %rbx, %rbx
232  .L104:
233  movq  %rbx, %rax
234  movq  8(%rsp), %rbx
235  movq  16(%rsp), %rbp

```

```

236     addq    $24, %rsp
237     ret
238     .cfi_endproc
239     .type   getUser, @function
240     .size   getUser, . - getUser
241     .text
242     .align  16
243     .globl  main
244 main:
245     .cfi_startproc
246     subq    $536, %rsp
247     .cfi_adjust_cfa_offset 536
248     leaq    544(%rsp), %rax
249     movq    %rax, 0(%rsp)
250     movq    %rbx, 8(%rsp)
251     call    setup_users
252     movq    %rax, %rbx
253     leaq    __stringlit_10(%rip), %rdi
254     movl    $0, %eax
255     call    printf
256     leaq    __stringlit_11(%rip), %rdi
257     movl    $0, %eax
258     call    printf
259     leaq    __stringlit_12(%rip), %rdi
260     leaq    16(%rsp), %rsi
261     movl    $0, %eax
262     call    __isoc99_scanf
263     leaq    16(%rsp), %rsi
264     movq    %rbx, %rdi
265     call    getUser
266     movq    %rax, %rbx
267     cmpq    $0, %rbx
268     jne     .L105
269     leaq    __stringlit_13(%rip), %rdi
270     leaq    16(%rsp), %rsi
271     movl    $0, %eax
272     call    printf
273     xorl    %eax, %eax
274     jmp     .L106
275 .L105:
276     leaq    __stringlit_14(%rip), %rdi
277     movl    $0, %eax
278     call    printf
279     leaq    __stringlit_12(%rip), %rdi
280     leaq    272(%rsp), %rsi
281     movl    $0, %eax
282     call    __isoc99_scanf
283     leaq    264(%rbx), %rdi
284     leaq    272(%rsp), %rsi
285     call    strcmp
286     testl   %eax, %eax
287     je      .L107
288     leaq    __stringlit_15(%rip), %rdi
289     movl    $0, %eax
290     call    printf
291     xorl    %eax, %eax
292     jmp     .L106

```



```

293 .L107:
294     leaq  __stringlit_16(%rip), %rdi
295     leaq  8(%rbx), %rsi
296     movl  $0, %eax
297     call  printf
298     leaq  __stringlit_9(%rip), %rdi
299     movl  $0, %eax
300     call  printf
301     leaq  __stringlit_17(%rip), %rdi
302     leaq  8(%rbx), %rsi
303     movl  $0, %eax
304     call  printf
305     leaq  __stringlit_18(%rip), %rdi
306     movq  520(%rbx), %rsi
307     movl  $0, %eax
308     call  printf
309     xorl  %eax, %eax
310 .L106:
311     movq  8(%rsp), %rbx
312     addq  $536, %rsp
313     ret
314     .cfi_endproc
315     .type main, @function
316     .size main, . - main

```

## C.12 password-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: password.c -S -O3 -o compCert/out/password-O3.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:
12    .ascii    "passwOrd123\000"
13    .type     __stringlit_6, @object
14    .size     __stringlit_6, . - __stringlit_6
15    .section    .rodata
16    .align     1
17 __stringlit_4:
18    .ascii    "!alice12!_veuqe@@hak\000"
19    .type     __stringlit_4, @object
20    .size     __stringlit_4, . - __stringlit_4
21    .section    .rodata
22    .align     1
23 __stringlit_14:
24    .ascii    "Password: \000"
25    .type     __stringlit_14, @object
26    .size     __stringlit_14, . - __stringlit_14
27    .section    .rodata
28    .align     1
29 __stringlit_18:
30    .ascii    "Your balance: $%ld\012\000"

```

```

31 .type __stringlit_18, @object
32 .size __stringlit_18, . - __stringlit_18
33 .section .rodata
34 .align 1
35 __stringlit_13:
36 .ascii "User < %s > does not exist.\012\000"
37 .type __stringlit_13, @object
38 .size __stringlit_13, . - __stringlit_13
39 .section .rodata
40 .align 1
41 __stringlit_8:
42 .ascii " %02ld. %s\012\000"
43 .type __stringlit_8, @object
44 .size __stringlit_8, . - __stringlit_8
45 .section .rodata
46 .align 1
47 __stringlit_1:
48 .ascii "admin\000"
49 .type __stringlit_1, @object
50 .size __stringlit_1, . - __stringlit_1
51 .section .rodata
52 .align 1
53 __stringlit_2:
54 .ascii "4dmin__4eva\000"
55 .type __stringlit_2, @object
56 .size __stringlit_2, . - __stringlit_2
57 .section .rodata
58 .align 1
59 __stringlit_3:
60 .ascii "alice\000"
61 .type __stringlit_3, @object
62 .size __stringlit_3, . - __stringlit_3
63 .section .rodata
64 .align 1
65 __stringlit_11:
66 .ascii "Username: \000"
67 .type __stringlit_11, @object
68 .size __stringlit_11, . - __stringlit_11
69 .section .rodata
70 .align 1
71 __stringlit_5:
72 .ascii "abdul\000"
73 .type __stringlit_5, @object
74 .size __stringlit_5, . - __stringlit_5
75 .section .rodata
76 .align 1
77 __stringlit_17:
78 .ascii "Welcome, %s!\012\000"
79 .type __stringlit_17, @object
80 .size __stringlit_17, . - __stringlit_17
81 .section .rodata
82 .align 1
83 __stringlit_12:
84 .ascii "%255s\000"
85 .type __stringlit_12, @object
86 .size __stringlit_12, . - __stringlit_12
87 .section .rodata

```

```

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

```

```

145 movq $528, %rdi
146 call malloc
147 movq %rax, %rbx
148 leaq 8(%rbx), %rdi
149 leaq __stringlit_5(%rip), %rsi
150 call strcpy
151 leaq 264(%rbx), %rdi
152 leaq __stringlit_6(%rip), %rsi
153 call strcpy
154 movq $2, %r10
155 movq %r10, 520(%rbx)
156 movq %r12, 0(%rbp)
157 movq %rbx, 0(%r12)
158 xorq %r8, %r8
159 movq %r8, 0(%rbx)
160 movq %rbp, %rax
161 movq 8(%rsp), %rbx
162 movq 16(%rsp), %rbp
163 movq 24(%rsp), %r12
164 addq $40, %rsp
165 ret
166 .cfi_endproc
167 .type setup_users, @function
168 .size setup_users, . - setup_users
169 .text
170 .align 16
171 .globl print_users
172 print_users:
173 .cfi_startproc
174 subq $24, %rsp
175 .cfi_adjust_cfa_offset 24
176 leaq 32(%rsp), %rax
177 movq %rax, 0(%rsp)
178 movq %rbx, 8(%rsp)
179 movq %rbp, 16(%rsp)
180 movq %rdi, %rbp
181 leaq __stringlit_7(%rip), %rdi
182 movl $0, %eax
183 call printf
184 xorq %rbx, %rbx
185 .L100:
186 cmpq $0, %rbp
187 je .L101
188 leaq 1(%rbx), %rbx
189 leaq __stringlit_8(%rip), %rdi
190 leaq 8(%rbp), %rdx
191 movq %rbx, %rsi
192 movl $0, %eax
193 call printf
194 movq 0(%rbp), %rbp
195 jmp .L100
196 .L101:
197 leaq __stringlit_9(%rip), %rdi
198 movl $0, %eax
199 call printf
200 movq 8(%rsp), %rbx
201 movq 16(%rsp), %rbp

```

```

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

```

```

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

```

```
316 .size main, . - main
```

## C.13 deadStoreElimination-O0.s

```
1 # File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -O0 -o compCert/out/deadStoreElimination-O0.s
3 .text
4 .align 16
5 .globl deadStore
6 deadStore:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
13 cmpl %esi, %edi
14 jle .L101
15 leal -1(%edi), %edi
16 jmp .L100
17 .L101:
18 leal 0(%edi,%esi,1), %eax
19 addq $8, %rsp
20 ret
21 .cfi_endproc
22 .type deadStore, @function
23 .size deadStore, . - deadStore
24 .text
25 .align 16
26 .globl main
27 main:
28 .cfi_startproc
29 subq $8, %rsp
30 .cfi_adjust_cfa_offset 8
31 leaq 16(%rsp), %rax
32 movq %rax, 0(%rsp)
33 movl $2, %esi
34 call deadStore
35 xorl %eax, %eax
36 addq $8, %rsp
37 ret
38 .cfi_endproc
39 .type main, @function
40 .size main, . - main
```

## C.14 deadStoreElimination-O3.s

```
1 # File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -O3 -o compCert/out/deadStoreElimination-O3.s
3 .text
4 .align 16
5 .globl deadStore
6 deadStore:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
```

```

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

```

## C.15 pread-00.s

```

1  # File generated by CompCert 3.7
2  # Command line: pread.c -S -O0 -o compCert/out/pread-00.s
3  .comm  z, 4, 4
4  .comm  x, 4, 4
5  .text
6  .align  16
7  .globl  main
8  main:
9  .cfi_startproc
10  subq  $8, %rsp
11  .cfi_adjust_cfa_offset  8
12  leaq  16(%rsp), %rax
13  movq  %rax, 0(%rsp)
14  .L100:
15  movl  z(%rip), %edx
16  movq  %rdx, %rsi
17  movq  %rsi, %rax
18  testl  %eax, %eax
19  leal  1(%eax), %ecx
20  cmovl  %rcx, %rax
21  sarl  $1, %eax
22  leal  0(,%eax,2), %edi
23  subl  %edi, %esi
24  testl  %esi, %esi

```



```

25  jne .L100
26  movl x(%rip), %esi
27  movl z(%rip), %esi
28  jmp .L100
29  .cfi_endproc
30  .type main, @function
31  .size main, . - main

```

## C.16 pread-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: pread.c -S -O3 -o compCert/out/pread-O3.s
3  .comm z, 4, 4
4  .comm x, 4, 4
5  .text
6  .align 16
7  .globl main
8 main:
9  .cfi_startproc
10 subq $8, %rsp
11 .cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:
15 movl z(%rip), %edx
16 movq %rdx, %rsi
17 movq %rsi, %rax
18 testl %eax, %eax
19 leal 1(%eax), %ecx
20 cmovl %rcx, %rax
21 sarl $1, %eax
22 leal 0(,%eax,2), %edi
23 subl %edi, %esi
24 testl %esi, %esi
25 jne .L100
26 movl x(%rip), %esi
27 movl z(%rip), %esi
28 jmp .L100
29 .cfi_endproc
30 .type main, @function
31 .size main, . - main

```

## D CompCert Annotated Assembly Output

### D.1 comment-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: comment.c -S -O0 -o annotated/comment-O0.s
3  .text
4  .align 16
5  .globl main
6 main:
7  .cfi_startproc
8  subq $8, %rsp
9  .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax

```

```

11  movq  %rax, 0(%rsp)
12  .L100:
13  xorl  %eax, %eax
14  addq  $8, %rsp
15  ret
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"

```

## D.2 comment-O3.s

```

1  # File generated by CompCert 3.7
2  # Command line: comment.c -S -O3 -o annotated/comment-O3.s
3  .text
4  .align 16
5  .globl main
6  main:
7  .cfi_startproc
8  subq  $8, %rsp
9  .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq  %rax, 0(%rsp)
12 .L100:
13 xorl  %eax, %eax
14 addq  $8, %rsp
15 ret
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"

```

## D.3 variable-O0.s

```

1  # File generated by CompCert 3.7
2  # Command line: variable.c -S -O0
3  .text
4  .align 16
5  .globl main
6  main:
7  .cfi_startproc
8  subq  $8, %rsp
9  .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq  %rax, 0(%rsp)
12 movl  $-10, %esi
13 .L100:
14 movl  $98, %ecx
15 .L101:

```

```

16  negl  %ecx
17  .L102:
18  movl  $1, %r10d
19  .L103:
20  movabsq $4294967296, %rax
21  .L104:
22  movsd  .L105(%rip), %xmm1 # 3.14159265358979312
23  cvtsd2ss %xmm1, %xmm3
24  cvtss2sd %xmm3, %xmm0
25  movsd  .L106(%rip), %xmm2 # 2.378400000000000007
26  divsd  %xmm2, %xmm0
27  cqto
28  shrq   $59, %rdx
29  leaq   0(%rax,%rdx,1), %rax
30  sarq   $5, %rax
31  leal   0(%eax,%esi,1), %r9d
32  leal   0(%r9d,%ecx,1), %r8d
33  leal   0(%r8d,%r10d,1), %r10d
34  cvttss2si %xmm3, %edx
35  leal   0(%r10d,%edx,1), %r11d
36  cvttsd2si %xmm0, %r8d
37  leal   0(%r11d,%r8d,1), %ecx
38  leal   0(%ecx,%edi,1), %eax
39  addq   $8, %rsp
40  ret
41  .cfi_endproc
42  .type  main, @function
43  .size  main, . - main
44  .section  .rodata.cst8,"aM",@progbits,8
45  .align   8
46  .L105:  .quad 0x400921fb54442d18
47  .L106:  .quad 0x400306f694467382
48  .section  "__compert_ais_annotations","",@note
49  .ascii  "# file:variable.c line:3 function:main\n"
50  .byte  7,8
51  .quad  .L100
52  .ascii  " reg(\"rsi\") = int\n"
53  .ascii  "# file:variable.c line:5 function:main\n"
54  .byte  7,8
55  .quad  .L101
56  .ascii  " reg(\"rcx\") = char\n"
57  .ascii  "# file:variable.c line:7 function:main\n"
58  .byte  7,8
59  .quad  .L102
60  .ascii  " reg(\"rcx\") = unsigned int\n"
61  .ascii  "# file:variable.c line:9 function:main\n"
62  .byte  7,8
63  .quad  .L103
64  .ascii  " reg(\"r10\") = short\n"
65  .ascii  "# file:variable.c line:11 function:main\n"
66  .byte  7,8
67  .quad  .L104
68  .ascii  " reg(\"rax\") = long\n"

```

## D.4 variable-O3.s

```

1 # File generated by CompCert 3.7

```

```

2 # Command line: variable.c -S -O3
3 .text
4 .align 16
5 .globl main
6 main:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
13 .L101:
14 .L102:
15 .L103:
16 .L104:
17 leal 134217625(%edi), %eax
18 addq $8, %rsp
19 ret
20 .cfi_endproc
21 .type main, @function
22 .size main, . - main
23 .section "__compert_ais_annotations","",@note
24 .ascii "# file:variable.c line:3 function:main\n"
25 .byte 7,8
26 .quad .L100
27 .ascii " -10 = int\n"
28 .ascii "# file:variable.c line:5 function:main\n"
29 .byte 7,8
30 .quad .L101
31 .ascii " 98 = char\n"
32 .ascii "# file:variable.c line:7 function:main\n"
33 .byte 7,8
34 .quad .L102
35 .ascii " -98 = unsigned int\n"
36 .ascii "# file:variable.c line:9 function:main\n"
37 .byte 7,8
38 .quad .L103
39 .ascii " 1 = short\n"
40 .ascii "# file:variable.c line:11 function:main\n"
41 .byte 7,8
42 .quad .L104
43 .ascii " 4294967296 = long\n"

```

## D.5 volatile-O0.s

```

1 volatile.c:4: error: access to volatile variable 'x' for parameter '%e1' is not supported
   in ais annotations
2 1 error detected.

```

## D.6 volatile-O3.s

```

1 volatile.c:4: error: access to volatile variable 'x' for parameter '%e1' is not supported
   in ais annotations
2 1 error detected.

```

## D.7 loop-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: loop.c -S -O0 -o annotated/loop-O0.s
3 .comm z, 4, 4
4 .comm x, 4, 4
5 .text
6 .align 16
7 .globl main
8 main:
9 .cfi_startproc
10 subq $8, %rsp
11 .cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 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
30 movq %rdx, %rax
31 testl %eax, %eax
32 leal 1(%eax), %ecx
33 cmovl %rcx, %rax
34 sarl $1, %eax
35 leal 0(,%eax,2), %edi
36 movq %rdx, %rcx
37 subl %edi, %ecx
38 testl %ecx, %ecx
39 jne .L108
40 movl x(%rip), %edi
41 movl z(%rip), %esi
42 jmp .L105
43 .cfi_endproc
44 .type main, @function
45 .size main, . - main
46 .section "__compcert_ais_annotations","",@note
47 .ascii "# file:loop.c line:7 function:main\n"
48 .byte 7,8
49 .quad .L100
50 .ascii " L(mem("
51 .byte 7,8
52 .quad z
53 .ascii ", 4)) = true\n"
54 .ascii "# file:loop.c line:8 function:main\n"
55 .byte 7,8
56 .quad .L101
57 .ascii " L(mem("

```

```

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

```

## D.8 loop-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: loop.c -S -O3 -o annotated/loop-O3.s
3 .comm z, 4, 4
4 .comm x, 4, 4
5 .text
6 .align 16
7 .globl main
8 main:

```

```

 9  .cfi_startproc
10  subq $8, %rsp
11  .cfi_adjust_cfa_offset 8
12  leaq 16(%rsp), %rax
13  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
30  movq %rdx, %rax
31  testl %eax, %eax
32  leal 1(%eax), %ecx
33  cmovl %rcx, %rax
34  sarl $1, %eax
35  leal 0(,%eax,2), %edi
36  movq %rdx, %rcx
37  subl %edi, %ecx
38  testl %ecx, %ecx
39  jne .L108
40  movl x(%rip), %edi
41  movq %rdx, %rsi
42  jmp .L105
43  .cfi_endproc
44  .type main, @function
45  .size main, . - main
46  .section "__compcert_ais_annotations","",@note
47  .ascii "# file:loop.c line:7 function:main\n"
48  .byte 7,8
49  .quad .L100
50  .ascii " L(mem("
51  .byte 7,8
52  .quad z
53  .ascii ", 4)) = true\n"
54  .ascii "# file:loop.c line:8 function:main\n"
55  .byte 7,8
56  .quad .L101
57  .ascii " L(mem("
58  .byte 7,8
59  .quad x
60  .ascii ", 4))= mem("
61  .byte 7,8
62  .quad z
63  .ascii ", 4) % 2 == 0\n"
64  .ascii "# file:loop.c line:11 function:main\n"
65  .byte 7,8

```

```

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

```

## D.9 rooster-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: rooster.c -S -O0 -o annotated/rooster-O0.s
3 .comm rooster, 4, 4
4 .comm drake, 4, 4
5 .comm goose, 4, 4
6 .data
7 .align 4
8 count:
9 .long 0
10 .type count, @object
11 .size count, . - count
12 .text
13 .align 16
14 .globl fun
15 fun:
16 .cfi_startproc

```



```

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

```

```

74 movl %eax, drake(%rip)
75 movl $10, %eax
76 movl %eax, goose(%rip)
77 movl rooster(%rip), %edi
78 movl drake(%rip), %esi
79 movl goose(%rip), %edx
80 call fun
81 xorl %eax, %eax
82 addq $8, %rsp
83 ret
84 .cfi_endproc
85 .type main, @function
86 .size main, . - main
87 .section "__compcert_ais_annotations","",@note
88 .ascii "# file:rooster.c line:6 function:fun\n"
89 .byte 7,8
90 .quad .L100
91 .ascii " CRITICAL COMMENT\n"
92 .ascii "# file:rooster.c line:26 function:main\n"
93 .byte 7,8
94 .quad .L107
95 .ascii " L(mem("
96 .byte 7,8
97 .quad goose
98 .ascii ", 4)) = medium\n"
99 .ascii "# file:rooster.c line:27 function:main\n"
100 .byte 7,8
101 .quad .L108
102 .ascii " EXCEPTIONAL\n"

```

## D.10 rooster-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: rooster.c -S -O3 -o annotated/rooster-O3.s
3 .comm rooster, 4, 4
4 .comm drake, 4, 4
5 .comm goose, 4, 4
6 .data
7 .align 4
8 count:
9 .long 0
10 .type count, @object
11 .size count, . - count
12 .text
13 .align 16
14 .globl fun
15 fun:
16 .cfi_startproc
17 subq $8, %rsp
18 .cfi_adjust_cfa_offset 8
19 leaq 16(%rsp), %rax
20 movq %rax, 0(%rsp)
21 .L100:
22 leal 0(%edi,%esi,1), %r9d
23 leal 0(%r9d,%edx,1), %eax
24 testl %eax, %eax
25 jl .L101

```

```

26  cmpl  %edx, %esi
27  setl  %al
28  movzbl %al, %eax
29  xorl  %r8d, %r8d
30  cmpl  %esi, %edi
31  cmovge %r8, %rax
32  cmpl  $0, %eax
33  je    .L102
34 .L103:
35  cmpl  %esi, %edi
36  je    .L102
37  leal  1(%edi), %edi
38  movl  count(%rip), %ecx
39  leal  1(%ecx), %r8d
40  movl  %r8d, count(%rip)
41 .L104:
42  cmpl  %edx, %esi
43  je    .L103
44  leal  -1(%edx), %edx
45  movl  count(%rip), %r10d
46  leal  1(%r10d), %r8d
47  movl  %r8d, count(%rip)
48  jmp   .L104
49 .L102:
50  movl  count(%rip), %eax
51 .L101:
52  addq  $8, %rsp
53  ret
54  .cfi_endproc
55  .type fun, @function
56  .size fun, . - fun
57  .text
58  .align 16
59  .globl main
60 main:
61  .cfi_startproc
62  subq  $8, %rsp
63  .cfi_adjust_cfa_offset 8
64  leaq  16(%rsp), %rax
65  movq  %rax, 0(%rsp)
66 .L105:
67 .L106:
68  movl  $1, %eax
69  movl  %eax, rooster(%rip)
70  movl  $5, %eax
71  movl  %eax, drake(%rip)
72  movl  $10, %eax
73  movl  %eax, goose(%rip)
74  movl  $1, %edi
75  movl  $5, %esi
76  movl  $10, %edx
77  call  fun
78  xorl  %eax, %eax
79  addq  $8, %rsp
80  ret
81  .cfi_endproc
82  .type main, @function

```

```

83 .size main, . - main
84 .section "__compcert_ais_annotations","",@note
85 .ascii "# file:rooster.c line:6 function:fun\n"
86 .byte 7,8
87 .quad .L100
88 .ascii " CRITICAL COMMENT\n"
89 .ascii "# file:rooster.c line:26 function:main\n"
90 .byte 7,8
91 .quad .L105
92 .ascii " L(mem("
93 .byte 7,8
94 .quad goose
95 .ascii ", 4)) = medium\n"
96 .ascii "# file:rooster.c line:27 function:main\n"
97 .byte 7,8
98 .quad .L106
99 .ascii " EXCEPTIONAL\n"

```

## D.11 password-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: password.c -S -O0 -o annotated/password-O0.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:
12 .ascii "passwOrd123\000"
13 .type __stringlit_6, @object
14 .size __stringlit_6, . - __stringlit_6
15 .section .rodata
16 .align 1
17 __stringlit_4:
18 .ascii "!alice12!_veuje@@hak\000"
19 .type __stringlit_4, @object
20 .size __stringlit_4, . - __stringlit_4
21 .section .rodata
22 .align 1
23 __stringlit_14:
24 .ascii "Password: \000"
25 .type __stringlit_14, @object
26 .size __stringlit_14, . - __stringlit_14
27 .section .rodata
28 .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%ld\012\000"
31 .type __stringlit_18, @object
32 .size __stringlit_18, . - __stringlit_18
33 .section .rodata
34 .align 1
35 __stringlit_13:
36 .ascii "User < %s > does not exist.\012\000"
37 .type __stringlit_13, @object

```

```

38 .size __stringlit_13, . - __stringlit_13
39 .section .rodata
40 .align 1
41 __stringlit_8:
42 .ascii "%02ld. %s\012\000"
43 .type __stringlit_8, @object
44 .size __stringlit_8, . - __stringlit_8
45 .section .rodata
46 .align 1
47 __stringlit_1:
48 .ascii "admin\000"
49 .type __stringlit_1, @object
50 .size __stringlit_1, . - __stringlit_1
51 .section .rodata
52 .align 1
53 __stringlit_2:
54 .ascii "4dmin__4eva\000"
55 .type __stringlit_2, @object
56 .size __stringlit_2, . - __stringlit_2
57 .section .rodata
58 .align 1
59 __stringlit_3:
60 .ascii "alice\000"
61 .type __stringlit_3, @object
62 .size __stringlit_3, . - __stringlit_3
63 .section .rodata
64 .align 1
65 __stringlit_11:
66 .ascii "Username: \000"
67 .type __stringlit_11, @object
68 .size __stringlit_11, . - __stringlit_11
69 .section .rodata
70 .align 1
71 __stringlit_5:
72 .ascii "abdul\000"
73 .type __stringlit_5, @object
74 .size __stringlit_5, . - __stringlit_5
75 .section .rodata
76 .align 1
77 __stringlit_17:
78 .ascii "Welcome, %s!\012\000"
79 .type __stringlit_17, @object
80 .size __stringlit_17, . - __stringlit_17
81 .section .rodata
82 .align 1
83 __stringlit_12:
84 .ascii "%255s\000"
85 .type __stringlit_12, @object
86 .size __stringlit_12, . - __stringlit_12
87 .section .rodata
88 .align 1
89 __stringlit_9:
90 .ascii "\012\000"
91 .type __stringlit_9, @object
92 .size __stringlit_9, . - __stringlit_9
93 .section .rodata
94 .align 1

```

```

95 __stringlit_15:
96     .ascii  "ERROR: incorrect password\012\000"
97     .type  __stringlit_15, @object
98     .size  __stringlit_15, . - __stringlit_15
99     .section .rodata
100    .align  1
101 __stringlit_10:
102    .ascii  "Welcome to BigBank Australia!\012\000"
103    .type  __stringlit_10, @object
104    .size  __stringlit_10, . - __stringlit_10
105    .section .rodata
106    .align  1
107 __stringlit_16:
108    .ascii  "Logged in as < %s >!\012\000"
109    .type  __stringlit_16, @object
110    .size  __stringlit_16, . - __stringlit_16
111    .text
112    .align  16
113    .globl setup_users
114 setup_users:
115    .cfi_startproc
116    subq  $40, %rsp
117    .cfi_adjust_cfa_offset 40
118    leaq  48(%rsp), %rax
119    movq  %rax, 0(%rsp)
120    movq  %rbx, 8(%rsp)
121    movq  %rbp, 16(%rsp)
122    movq  %r12, 24(%rsp)
123    movq  $528, %rdi
124    call  malloc
125    movq  %rax, %rbp
126    leaq  8(%rbp), %rdi
127    leaq  __stringlit_1(%rip), %rsi
128    call  strcpy
129    leaq  264(%rbp), %rdi
130    leaq  __stringlit_2(%rip), %rsi
131    call  strcpy
132 .L100:
133    movq  $1000000, %r10
134    movq  %r10, 520(%rbp)
135    movq  $528, %rdi
136    call  malloc
137    movq  %rax, %r12
138    leaq  8(%r12), %rdi
139    leaq  __stringlit_3(%rip), %rsi
140    call  strcpy
141    leaq  264(%r12), %rdi
142    leaq  __stringlit_4(%rip), %rsi
143    call  strcpy
144 .L101:
145    movq  $783, %r9
146    movq  %r9, 520(%r12)
147    movq  $528, %rdi
148    call  malloc
149    movq  %rax, %rbx
150    leaq  8(%rbx), %rdi
151    leaq  __stringlit_5(%rip), %rsi

```

```

152  call    strcpy
153  leaq    264(%rbx), %rdi
154  leaq    __stringlit_6(%rip), %rsi
155  call    strcpy
156 .L102:
157  movq    $2, %r11
158  movq    %r11, 520(%rbx)
159  movq    %r12, 0(%rbp)
160  movq    %rbx, 0(%r12)
161  xorq    %r8, %r8
162  movq    %r8, 0(%rbx)
163  movq    %rbp, %rax
164  movq    8(%rsp), %rbx
165  movq    16(%rsp), %rbp
166  movq    24(%rsp), %r12
167  addq    $40, %rsp
168  ret
169  .cfi_endproc
170  .type   setup_users, @function
171  .size   setup_users, . - setup_users
172  .text
173  .align  16
174  .globl  print_users
175 print_users:
176  .cfi_startproc
177  subq    $24, %rsp
178  .cfi_adjust_cfa_offset 24
179  leaq    32(%rsp), %rax
180  movq    %rax, 0(%rsp)
181  movq    %rbx, 8(%rsp)
182  movq    %rbp, 16(%rsp)
183  movq    %rdi, %rbx
184  leaq    __stringlit_7(%rip), %rdi
185  movl    $0, %eax
186  call    printf
187  xorq    %rbp, %rbp
188 .L103:
189  cmpq    $0, %rbx
190  je      .L104
191  leaq    1(%rbp), %rbp
192  leaq    __stringlit_8(%rip), %rdi
193  leaq    8(%rbx), %rdx
194  movq    %rbp, %rsi
195  movl    $0, %eax
196  call    printf
197  movq    0(%rbx), %rbx
198  jmp     .L103
199 .L104:
200  leaq    __stringlit_9(%rip), %rdi
201  movl    $0, %eax
202  call    printf
203  movq    8(%rsp), %rbx
204  movq    16(%rsp), %rbp
205  addq    $24, %rsp
206  ret
207  .cfi_endproc
208  .type   print_users, @function

```

```

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

```



```

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

```

```

323 .quad .L100
324 .ascii " L((reg(\"rbp\") + 264)) = high\n"
325 .ascii "# file:password.c line:26 function:setup_users\n"
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"

```

## D.12 password-O3.s

```

1 # File generated by CompCert 3.7
2 # Command line: password.c -S -O3 -o annotated/password-O3.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:
12 .ascii "passwOrd123\000"
13 .type __stringlit_6, @object
14 .size __stringlit_6, . - __stringlit_6
15 .section .rodata
16 .align 1
17 __stringlit_4:
18 .ascii "!alice12!_veuje@@hak\000"
19 .type __stringlit_4, @object
20 .size __stringlit_4, . - __stringlit_4
21 .section .rodata
22 .align 1
23 __stringlit_14:
24 .ascii "Password: \000"
25 .type __stringlit_14, @object
26 .size __stringlit_14, . - __stringlit_14
27 .section .rodata
28 .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%ld\012\000"
31 .type __stringlit_18, @object
32 .size __stringlit_18, . - __stringlit_18
33 .section .rodata
34 .align 1
35 __stringlit_13:
36 .ascii "User < %s > does not exist.\012\000"
37 .type __stringlit_13, @object
38 .size __stringlit_13, . - __stringlit_13
39 .section .rodata
40 .align 1
41 __stringlit_8:
42 .ascii " %02ld. %s\012\000"
43 .type __stringlit_8, @object
44 .size __stringlit_8, . - __stringlit_8

```

```

45 .section .rodata
46 .align 1
47 __stringlit_1:
48 .ascii "admin\000"
49 .type __stringlit_1, @object
50 .size __stringlit_1, . - __stringlit_1
51 .section .rodata
52 .align 1
53 __stringlit_2:
54 .ascii "4dmin__4eva\000"
55 .type __stringlit_2, @object
56 .size __stringlit_2, . - __stringlit_2
57 .section .rodata
58 .align 1
59 __stringlit_3:
60 .ascii "alice\000"
61 .type __stringlit_3, @object
62 .size __stringlit_3, . - __stringlit_3
63 .section .rodata
64 .align 1
65 __stringlit_11:
66 .ascii "Username: \000"
67 .type __stringlit_11, @object
68 .size __stringlit_11, . - __stringlit_11
69 .section .rodata
70 .align 1
71 __stringlit_5:
72 .ascii "abdul\000"
73 .type __stringlit_5, @object
74 .size __stringlit_5, . - __stringlit_5
75 .section .rodata
76 .align 1
77 __stringlit_17:
78 .ascii "Welcome, %s!\012\000"
79 .type __stringlit_17, @object
80 .size __stringlit_17, . - __stringlit_17
81 .section .rodata
82 .align 1
83 __stringlit_12:
84 .ascii "%255s\000"
85 .type __stringlit_12, @object
86 .size __stringlit_12, . - __stringlit_12
87 .section .rodata
88 .align 1
89 __stringlit_9:
90 .ascii "\012\000"
91 .type __stringlit_9, @object
92 .size __stringlit_9, . - __stringlit_9
93 .section .rodata
94 .align 1
95 __stringlit_15:
96 .ascii "ERROR: incorrect password\012\000"
97 .type __stringlit_15, @object
98 .size __stringlit_15, . - __stringlit_15
99 .section .rodata
100 .align 1
101 __stringlit_10:

```

```

102 .ascii "Welcome to BigBank Australia!\012\000"
103 .type __stringlit_10, @object
104 .size __stringlit_10, . - __stringlit_10
105 .section .rodata
106 .align 1
107 __stringlit_16:
108 .ascii "Logged in as < %s >!\012\000"
109 .type __stringlit_16, @object
110 .size __stringlit_16, . - __stringlit_16
111 .text
112 .align 16
113 .globl setup_users
114 setup_users:
115 .cfi_startproc
116 subq $40, %rsp
117 .cfi_adjust_cfa_offset 40
118 leaq 48(%rsp), %rax
119 movq %rax, 0(%rsp)
120 movq %rbx, 8(%rsp)
121 movq %rbp, 16(%rsp)
122 movq %r12, 24(%rsp)
123 movq $528, %rdi
124 call malloc
125 movq %rax, %rbp
126 leaq 8(%rbp), %rdi
127 leaq __stringlit_1(%rip), %rsi
128 call strcpy
129 leaq 264(%rbp), %rdi
130 leaq __stringlit_2(%rip), %rsi
131 call strcpy
132 .L100:
133 movq $1000000, %r10
134 movq %r10, 520(%rbp)
135 movq $528, %rdi
136 call malloc
137 movq %rax, %r12
138 leaq 8(%r12), %rdi
139 leaq __stringlit_3(%rip), %rsi
140 call strcpy
141 leaq 264(%r12), %rdi
142 leaq __stringlit_4(%rip), %rsi
143 call strcpy
144 .L101:
145 movq $783, %r9
146 movq %r9, 520(%r12)
147 movq $528, %rdi
148 call malloc
149 movq %rax, %rbx
150 leaq 8(%rbx), %rdi
151 leaq __stringlit_5(%rip), %rsi
152 call strcpy
153 leaq 264(%rbx), %rdi
154 leaq __stringlit_6(%rip), %rsi
155 call strcpy
156 .L102:
157 movq $2, %r11
158 movq %r11, 520(%rbx)

```

```

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

```

```

216 .cfi_adjust_cfa_offset 24
217 leaq 32(%rsp), %rax
218 movq %rax, 0(%rsp)
219 movq %rbx, 8(%rsp)
220 movq %rbp, 16(%rsp)
221 movq %rsi, %rbp
222 movq %rdi, %rbx
223 .L105:
224 cmpq $0, %rbx
225 je .L106
226 leaq 8(%rbx), %rdi
227 movq %rbp, %rsi
228 call strcmp
229 testl %eax, %eax
230 je .L107
231 movq 0(%rbx), %rbx
232 jmp .L105
233 .L106:
234 xorq %rbx, %rbx
235 .L107:
236 movq %rbx, %rax
237 movq 8(%rsp), %rbx
238 movq 16(%rsp), %rbp
239 addq $24, %rsp
240 ret
241 .cfi_endproc
242 .type getUser, @function
243 .size getUser, . - getUser
244 .text
245 .align 16
246 .globl main
247 main:
248 .cfi_startproc
249 subq $536, %rsp
250 .cfi_adjust_cfa_offset 536
251 leaq 544(%rsp), %rax
252 movq %rax, 0(%rsp)
253 movq %rbx, 8(%rsp)
254 call setup_users
255 movq %rax, %rbx
256 leaq __stringlit_10(%rip), %rdi
257 movl $0, %eax
258 call printf
259 leaq __stringlit_11(%rip), %rdi
260 movl $0, %eax
261 call printf
262 leaq __stringlit_12(%rip), %rdi
263 leaq 16(%rsp), %rsi
264 movl $0, %eax
265 call __isoc99_scanf
266 leaq 16(%rsp), %rsi
267 movq %rbx, %rdi
268 call getUser
269 movq %rax, %rbx
270 cmpq $0, %rbx
271 jne .L108
272 leaq __stringlit_13(%rip), %rdi

```

```

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

```

## D.13 deadStoreElimination-O0.s

```

1 # File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -O0 -o annotated/deadStoreElimination-O0.s
3 .text
4 .align 16
5 .globl deadStore
6 deadStore:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 movl $43981, %ecx
13 .L100:
14 nop
15 .L101:
16 cmpl %esi, %edi
17 jle .L102
18 leal -1(%edi), %edi
19 jmp .L101
20 .L102:
21 leal 0(%edi,%esi,1), %eax
22 addq $8, %rsp
23 ret
24 .cfi_endproc
25 .type deadStore, @function
26 .size deadStore, . - deadStore
27 .text
28 .align 16
29 .globl main
30 main:
31 .cfi_startproc
32 subq $8, %rsp
33 .cfi_adjust_cfa_offset 8
34 leaq 16(%rsp), %rax
35 movq %rax, 0(%rsp)
36 movl $2, %esi
37 call deadStore
38 xorl %eax, %eax
39 addq $8, %rsp
40 ret
41 .cfi_endproc
42 .type main, @function
43 .size main, . - main
44 .section "__compcert_ais_annotations","",@note
45 .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
46 .byte 7,8
47 .quad .L100
48 .ascii " L(reg(\"rcx\")) = high\n"

```

## D.14 deadStoreElimination-O3.s



```

1 # File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -O3 -o annotated/deadStoreElimination-O3.s
3 .text
4 .align 16
5 .globl deadStore
6 deadStore:
7 .cfi_startproc
8 subq $8, %rsp
9 .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
13 nop
14 .L101:
15 cmpl %esi, %edi
16 jle .L102
17 leal -1(%edi), %edi
18 jmp .L101
19 .L102:
20 leal 0(%edi,%esi,1), %eax
21 addq $8, %rsp
22 ret
23 .cfi_endproc
24 .type deadStore, @function
25 .size deadStore, . - deadStore
26 .text
27 .align 16
28 .globl main
29 main:
30 .cfi_startproc
31 subq $8, %rsp
32 .cfi_adjust_cfa_offset 8
33 leaq 16(%rsp), %rax
34 movq %rax, 0(%rsp)
35 movl $2, %esi
36 call deadStore
37 xorl %eax, %eax
38 addq $8, %rsp
39 ret
40 .cfi_endproc
41 .type main, @function
42 .size main, . - main
43 .section "__compcert_ais_annotations","",@note
44 .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
45 .byte 7,8
46 .quad .L100
47 .ascii " L(43981) = high\n"

```

## D.15 pread-O0.s

```

1 pread.c:6: error: access to volatile variable 'z' for parameter '%e1' is not supported in
  ais annotations
2 pread.c:7: error: access to volatile variable 'x' for parameter '%e1' is not supported in
  ais annotations
3 pread.c:7: error: access to volatile variable 'z' for parameter '%e2' is not supported in
  ais annotations
4 pread.c:20: error: access to volatile variable 'z' for parameter '%e2' is not supported in

```

```

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

```

## D.16 pread-O3.s

```

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

```

## E Inline Assembly Annotated C Programs

### E.1 comment.c

```

1 int main() {
2     asm("# CRITICAL COMMENT");
3     return 0;
4 }

```

### E.2 variable.c

```

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

```

```

19 float x = 3.141592653589793;
20 asm("# annotation: %0 = float" : : "X"(x));
21 double y = x / 2.3784;
22 asm("# annotation: %0 = double" : : "X"(y));
23 return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int) y + argc;
24 }

```

### E.3 volatile.c

```

1 volatile int x;
2
3 int main() {
4     // L(x) = High
5     return x + 1;
6 }

```

### E.4 loop.c

```

1 int z;
2 int x;
3
4 // security policies
5 // {L(z)=true}
6 // {L(x)=z % 2 == 0}
7
8 // predicates on initial state
9 // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
13     int r1 = 0;
14     // {L(r2)=False}
15     int r2 = 0;
16
17     while(1) {
18         do {
19             // {_invariant: r1 % 2 == 0 /\ r1 <= z}
20             // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
21             do {
22                 // {_invariant: r1 <= z}
23                 // {_Gamma: r1 -> LOW}
24                 r1 = z;
25             } while (r1 % 2 != 0);
26             r2 = x;
27             } while (z != r1);
28         }
29         return r2;
30 }

```

### E.5 rooster.c

```

1 int rooster;
2 int drake;
3 // MEDIUM
4 int goose;
5
6 int fun(int a, int b, int c) {

```

```

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

```

## E.6 password.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define BUFF_LEN 256
6
7 typedef struct user_t user_t;
8
9 struct user_t {
10     user_t* next;
11     char name[BUFF_LEN];
12     // L(password) = High
13     char password[BUFF_LEN];
14     size_t balance;
15 };
16
17 user_t* setup_users() {
18     user_t* user_admin = malloc(sizeof(user_t));
19     strcpy(user_admin->name, "admin");
20     strcpy(user_admin->password, "4dmin__4eva");
21     user_admin->balance = 1000000;
22
23     user_t* user_alice = malloc(sizeof(user_t));
24     strcpy(user_alice->name, "alice");
25     strcpy(user_alice->password, "!alice12!_veuje@h@hak");
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     user_abdul->balance = 2;
32
33     user_admin->next = user_alice;
34     user_alice->next = user_abdul;
35     user_abdul->next = NULL;
36
37     return user_admin;
38 }
39
40 void print_users(user_t* users) {
41     printf("--- USERS ---\n");
42     size_t count = 0;
43     while (users != NULL) {
44         printf(" %02ld. %s\n", ++count, users->name);
45         users = users->next;
46     }
47     printf("\n");
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
51     while (user_list != NULL) {
52         if (strcmp(user_list->name, name) == 0) {
53             return user_list;
54         }
55         user_list = user_list->next;
56     }
57     return NULL;
58 }
59
60 int main() {
61     user_t* users = setup_users();
62
63     printf("Welcome to BigBank Australia!\n");
64
65     char username[BUFF_LEN];
66     printf("Username: ");
67     scanf("%255s", username);
68
69     user_t* user = getUser(users, username);
70     if (user == NULL) {
71         printf("User < %s > does not exist.\n", username);
72         return 0;
73     }
74
75     char password[BUFF_LEN];
76     printf("Password: ");
77     scanf("%255s", password);
78     if (strcmp(user->password, password) != 0) {
79         printf("ERROR: incorrect password\n");
80         return 0;
81     }
82
83     printf("Logged in as < %s >!\n", user->name);

```

```

84     printf("\n");
85     printf("Welcome, %s!\n", user->name);
86     printf("Your balance: $%ld\n", user->balance);
87 }

```

## E.7 deadStoreElimination.c

```

1  int deadStore(int i, int n) {
2      int key = 0xabcd;
3      // L(key) = high
4
5      // do some work
6      int result = 0;
7      while (i > n) {
8          result += key;
9          i--;
10     }
11
12     // clear out our secret key
13     key = 0;
14     return i + n;
15 }
16
17 int main(int argc, char *argv[]) {
18     deadStore(argc, 2);
19 }

```

## E.8 pread.c

```

1  volatile int z;
2  volatile int x;
3
4  // security policies
5  // {L(z)=true}
6  // {L(x)=z % 2 == 0}
7
8  // predicates on initial state
9  // {_P_0: r1 % 2 == 0}
10 // {_Gamma_0: r1 -> LOW, r2 -> LOW}
11
12 int main() {
13     int r1 = 0;
14     // {L(r2)=False}
15     int r2 = 0;
16
17     while(1) {
18         do {
19             // {_invariant: r1 % 2 == 0 /\ r1 <= z}
20             // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
21             do {
22                 // {_invariant: r1 <= z}
23                 // {_Gamma: r1 -> LOW}
24                 r1 = z;
25             } while (r1 % 2 != 0);
26             r2 = x;
27         } while (z != r1);
28     }

```

```

29     return r2;
30 }

```

## F Inline Assembly Annotated Assembly Output

### F.1 comment-O0.s

```

1  .text
2  .file "comment.c"
3  .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
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl $0, -4(%rbp)
15 #APP
16 # CRITICAL COMMENT
17 #NO_APP
18 xorl %eax, %eax
19 popq %rbp
20 .cfi_def_cfa %rsp, 8
21 retq
22 .Lfunc_end0:
23 .size main, .Lfunc_end0-main
24 .cfi_endproc
25                                # -- End function
26 .ident "clang version 10.0.0-4ubuntu1 "
27 .section ".note.GNU-stack","",@progbits
28 .addrsig

```

### F.2 comment-O3.s

```

1  .text
2  .file "comment.c"
3  .globl main                # -- Begin function main
4  .p2align 4, 0x90
5  .type main,@function
6 main:                      # @main
7  .cfi_startproc
8  # %bb.0:
9  #APP
10 # CRITICAL COMMENT
11 #NO_APP
12 xorl %eax, %eax
13 retq
14 .Lfunc_end0:
15 .size main, .Lfunc_end0-main
16 .cfi_endproc
17                                # -- End function
18 .ident "clang version 10.0.0-4ubuntu1 "

```

```

19 .section ".note.GNU-stack","",@progbits
20 .addrsig

```

### F.3 variable-00.s

```

1  .text
2  .file "variable.c"
3  .section .rodata.cst8,"aM",@progbits,8
4  .p2align 3                      # -- Begin function main
5  .LCPI0_0:
6  .quad 4612538099476886402      # double 2.3784000000000001
7  .section .rodata.cst4,"aM",@progbits,4
8  .p2align 2
9  .LCPI0_1:
10 .long 1078530011                # float 3.14159274
11 .text
12 .globl main
13 .p2align 4, 0x90
14 .type main,@function
15 main:                           # @main
16 .cfi_startproc
17 # %bb.0:
18 pushq %rbp
19 .cfi_def_cfa_offset 16
20 .cfi_offset %rbp, -16
21 movq %rsp, %rbp
22 .cfi_def_cfa_register %rbp
23 movl $0, -4(%rbp)
24 movl %edi, -8(%rbp)
25 movq %rsi, -16(%rbp)
26 movl $-10, -20(%rbp)
27 movl -20(%rbp), %eax
28 #APP
29 # annotation: %eax = int
30 #NO_APP
31 movb $98, -21(%rbp)
32 movb -21(%rbp), %cl
33 #APP
34 # annotation: %cl = char
35 #NO_APP
36 xorl %eax, %eax
37 movsbl -21(%rbp), %edx
38 subl %edx, %eax
39 movl %eax, -28(%rbp)
40 movl -28(%rbp), %eax
41 #APP
42 # annotation: %eax = unsigned int
43 #NO_APP
44 movw $1, -30(%rbp)
45 movw -30(%rbp), %r8w
46 #APP
47 # annotation: %r8w = short
48 #NO_APP
49 movabsq $4294967296, %rsi      # imm = 0x100000000
50 movq %rsi, -40(%rbp)
51 movq -40(%rbp), %rsi
52 #APP

```



```

53 # annotation: %rsi = long
54 #NO_APP
55 movss .LCPIO_1(%rip), %xmm0 # xmm0 = mem[0],zero,zero,zero
56 movss %xmm0, -44(%rbp)
57 movss -44(%rbp), %xmm0 # xmm0 = mem[0],zero,zero,zero
58 #APP
59 # annotation: %xmm0 = float
60 #NO_APP
61 movsd .LCPIO_0(%rip), %xmm0 # xmm0 = mem[0],zero
62 movss -44(%rbp), %xmm1 # xmm1 = mem[0],zero,zero,zero
63 cvtss2sd %xmm1, %xmm1
64 divsd %xmm0, %xmm1
65 movsd %xmm1, -56(%rbp)
66 movsd -56(%rbp), %xmm0 # xmm0 = mem[0],zero
67 #APP
68 # annotation: %xmm0 = double
69 #NO_APP
70 movq -40(%rbp), %rax
71 cqto
72 movl $32, %esi
73 idivq %rsi
74 # kill: def $eax killed $eax killed $rax
75 addl -20(%rbp), %eax
76 addl -28(%rbp), %eax
77 movswl -30(%rbp), %edi
78 addl %edi, %eax
79 cvttss2si -44(%rbp), %edi
80 addl %edi, %eax
81 cvttss2si -56(%rbp), %edi
82 addl %edi, %eax
83 addl -8(%rbp), %eax
84 popq %rbp
85 .cfi_def_cfa %rsp, 8
86 retq
87 .Lfunc_end0:
88 .size main, .Lfunc_end0-main
89 .cfi_endproc
90 # -- End function
91 .ident "clang version 10.0.0-4ubuntu1 "
92 .section ".note.GNU-stack","",@progbits
93 .addrsig

```

## F.4 variable-O3.s

```

1 .text
2 .file "variable.c"
3 .section .rodata.cst4,"aM",@progbits,4
4 .p2align 2 # -- Begin function main
5 .LCPIO_0:
6 .long 1078530011 # float 3.14159274
7 .section .rodata.cst8,"aM",@progbits,8
8 .p2align 3
9 .LCPIO_1:
10 .quad 4608627556095693531 # double 1.3208849398808329
11 .text
12 .globl main
13 .p2align 4, 0x90

```

```

14 .type main,@function
15 main:                                     # @main
16 .cfi_startproc
17 # %bb.0:
18                                     # kill: def $edi killed $edi def $rdi
19 #APP
20 # annotation: $-10 = int
21 #NO_APP
22 #APP
23 # annotation: $98 = char
24 #NO_APP
25 #APP
26 # annotation: $-98 = unsigned int
27 #NO_APP
28 #APP
29 # annotation: $1 = short
30 #NO_APP
31 #APP
32 # annotation: $4294967296 = long
33 #NO_APP
34 movss .LCPI0_0(%rip), %xmm0 # xmm0 = mem[0],zero,zero,zero
35 #APP
36 # annotation: %xmm0 = float
37 #NO_APP
38 movsd .LCPI0_1(%rip), %xmm0 # xmm0 = mem[0],zero
39 #APP
40 # annotation: %xmm0 = double
41 #NO_APP
42 leal 134217625(%rdi), %eax
43 retq
44 .Lfunc_end0:
45 .size main, .Lfunc_end0-main
46 .cfi_endproc
47                                     # -- End function
48 .ident "clang version 10.0.0-4ubuntu1 "
49 .section ".note.GNU-stack","",@progbits
50 .addrsig

```

## F.5 volatile-O0.s

```

1 .text
2 .file "volatile.c"
3 .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
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl $0, -4(%rbp)
15 movl x, %eax
16 addl $1, %eax
17 popq %rbp

```

```

18 .cfi_def_cfa %rsp, 8
19 retq
20 .Lfunc_end0:
21 .size main, .Lfunc_end0-main
22 .cfi_endproc
23                                     # -- End function
24 .type x,@object                    # @x
25 .comm x,4,4
26 .ident "clang version 10.0.0-4ubuntu1 "
27 .section ".note.GNU-stack","",@progbits
28 .addrsig
29 .addrsig_sym x

```

## F.6 volatile-O3.s

```

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

```

## F.7 loop-O0.s

```

1 .text
2 .file "loop.c"
3 .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
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl $0, -4(%rbp)
15 movl $0, -8(%rbp)
16 movl $0, -12(%rbp)
17 .LBB0_1:                          # =>This Loop Header: Depth=1
18                                     # Child Loop BB0_2 Depth 2

```

```

19                                     #      Child Loop BB0_3 Depth 3
20     jmp .LBB0_2
21 .LBB0_2:                             #      Parent Loop BB0_1 Depth=1
22                                     # => This Loop Header: Depth=2
23                                     #      Child Loop BB0_3 Depth 3
24     jmp .LBB0_3
25 .LBB0_3:                             #      Parent Loop BB0_1 Depth=1
26                                     #      Parent Loop BB0_2 Depth=2
27                                     # => 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
31     movl -8(%rbp), %eax
32     cltd
33     movl $2, %ecx
34     idivl %ecx
35     cmpl $0, %edx
36     jne .LBB0_3
37 # %bb.5:                             #      in Loop: Header=BB0_2 Depth=2
38     movl x, %eax
39     movl %eax, -12(%rbp)
40 # %bb.6:                             #      in Loop: Header=BB0_2 Depth=2
41     movl z, %eax
42     cmpl -8(%rbp), %eax
43     jne .LBB0_2
44 # %bb.7:                             #      in Loop: Header=BB0_1 Depth=1
45     jmp .LBB0_1
46 .Lfunc_end0:
47     .size main, .Lfunc_end0-main
48     .cfi_endproc
49                                     # -- End function
50     .type z,@object                    # @z
51     .comm z,4,4
52     .type x,@object                    # @x
53     .comm x,4,4
54     .ident "clang version 10.0.0-4ubuntu1 "
55     .section ".note.GNU-stack","",@progbits
56     .addrsig
57     .addrsig_sym z
58     .addrsig_sym x

```

## F.8 loop-O3.s

```

1     .text
2     .file "loop.c"
3     .globl main                        # -- Begin function main
4     .p2align 4, 0x90
5     .type main,@function
6 main:                                # @main
7     .cfi_startproc
8     # %bb.0:
9     testb $1, z(%rip)
10    jne .LBB0_2
11    .p2align 4, 0x90
12 .LBB0_1:                            # =>This Inner Loop Header: Depth=1
13    jmp .LBB0_1
14    .p2align 4, 0x90

```

```

15 .LBB0_2:                                     # =>This Inner Loop Header: Depth=1
16     jmp .LBB0_2
17 .Lfunc_end0:
18     .size main, .Lfunc_end0-main
19     .cfi_endproc
20
21     .type z,@object                          # @z
22     .comm z,4,4
23     .type x,@object                          # @x
24     .comm x,4,4
25     .ident "clang version 10.0.0-4ubuntu1 "
26     .section ".note.GNU-stack","",@progbits
27     .addrsig

```

## F.9 rooster-O0.s

```

1  .text
2  .file "rooster.c"
3  .globl fun                                     # -- Begin function fun
4  .p2align 4, 0x90
5  .type fun,@function
6 fun:                                           # @fun
7  .cfi_startproc
8  # %bb.0:
9  pushq %rbp
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl %edi, -8(%rbp)
15 movl %esi, -12(%rbp)
16 movl %edx, -16(%rbp)
17 movl -8(%rbp), %eax
18 addl -12(%rbp), %eax
19 addl -16(%rbp), %eax
20 movl %eax, -20(%rbp)
21 cmpl $0, -20(%rbp)
22 jge .LBB0_2
23 # %bb.1:
24 movl -20(%rbp), %eax
25 movl %eax, -4(%rbp)
26 jmp .LBB0_12
27 .LBB0_2:
28 movl -8(%rbp), %eax
29 cmpl -12(%rbp), %eax
30 jge .LBB0_11
31 # %bb.3:
32 movl -12(%rbp), %eax
33 cmpl -16(%rbp), %eax
34 jge .LBB0_11
35 # %bb.4:
36 jmp .LBB0_5
37 .LBB0_5:                                     # =>This Loop Header: Depth=1
38                                     # Child Loop BB0_7 Depth 2
39 movl -8(%rbp), %eax
40 cmpl -12(%rbp), %eax
41 je .LBB0_10

```

```

42 # %bb.6:                                     #   in Loop: Header=BB0_5 Depth=1
43     movl  -8(%rbp), %eax
44     addl  $1, %eax
45     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    .LBB0_9
54 # %bb.8:                                     #   in Loop: Header=BB0_7 Depth=2
55     movl  -16(%rbp), %eax
56     addl  $-1, %eax
57     movl  %eax, -16(%rbp)
58     movl  fun.count, %eax
59     addl  $1, %eax
60     movl  %eax, fun.count
61     jmp   .LBB0_7
62 .LBB0_9:                                     #   in Loop: Header=BB0_5 Depth=1
63     jmp   .LBB0_5
64 .LBB0_10:
65     jmp   .LBB0_11
66 .LBB0_11:
67     movl  fun.count, %eax
68     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:
75     .size fun, .Lfunc_end0-fun
76     .cfi_endproc
77                                     # -- 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
85     .cfi_def_cfa_offset 16
86     .cfi_offset %rbp, -16
87     movq  %rsp, %rbp
88     .cfi_def_cfa_register %rbp
89     subq  $16, %rsp
90     movl  $0, -4(%rbp)
91     movl  $1, rooster
92     movl  $5, drake
93     movl  $10, goose
94     movl  rooster, %edi
95     movl  drake, %esi
96     movl  goose, %edx
97     callq fun
98     xorl  %ecx, %ecx

```

```

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

```

## F.10 rooster-O3.s

```

1  .text
2  .file "rooster.c"
3  .globl fun                                # -- Begin function fun
4  .p2align 4, 0x90
5  .type fun,@function
6 fun:                                       # @fun
7  .cfi_startproc
8  # %bb.0:
9
10                                     # kill: def $edx killed $edx def $rdx
11                                     # kill: def $esi killed $esi def $rsi
12                                     # kill: def $edi killed $edi def $rdi
13  leal  (%rsi,%rdi), %eax
14  addl  %edx, %eax
15  js    .LBB0_9
16  # %bb.1:
17  movl  fun.count(%rip), %eax
18  cmpl  %esi, %edi
19  jge   .LBB0_9
20  # %bb.2:
21  cmpl  %edx, %esi
22  jge   .LBB0_9
23  # %bb.3:
24  leal  1(%rdi), %ecx
25  cmpl  %esi, %ecx
26  jne   .LBB0_5
27  # %bb.4:
28  subl  %esi, %eax

```

```

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

```

## F.11 password-O0.s

```

1  .text
2  .file "password.c"
3  .globl setup_users                    # -- Begin function setup_users
4  .p2align 4, 0x90

```



```

5  .type setup_users,@function
6  setup_users:                                # @setup_users
7  .cfi_startproc
8  # %bb.0:
9  pushq %rbp
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 subq $80, %rsp
15 movl $528, %edi                                # imm = 0x210
16 callq malloc
17 movq %rax, -8(%rbp)
18 movq -8(%rbp), %rax
19 addq $8, %rax
20 movl $.L.str, %esi
21 movq %rax, %rdi
22 callq strcpy
23 movq -8(%rbp), %rcx
24 addq $264, %rcx                                # imm = 0x108
25 movl $.L.str.1, %esi
26 movq %rcx, %rdi
27 movq %rax, -32(%rbp)                            # 8-byte Spill
28 callq strcpy
29 movq -8(%rbp), %rcx
30 movq $1000000, 520(%rcx)                        # imm = 0xF4240
31 movl $528, %edi                                # imm = 0x210
32 movq %rax, -40(%rbp)                            # 8-byte Spill
33 callq malloc
34 movq %rax, -16(%rbp)
35 movq -16(%rbp), %rax
36 addq $8, %rax
37 movl $.L.str.2, %esi
38 movq %rax, %rdi
39 callq strcpy
40 movq -16(%rbp), %rcx
41 addq $264, %rcx                                # imm = 0x108
42 movl $.L.str.3, %esi
43 movq %rcx, %rdi
44 movq %rax, -48(%rbp)                            # 8-byte Spill
45 callq strcpy
46 movq -16(%rbp), %rcx
47 movq $783, 520(%rcx)                            # imm = 0x30F
48 movl $528, %edi                                # imm = 0x210
49 movq %rax, -56(%rbp)                            # 8-byte Spill
50 callq malloc
51 movq %rax, -24(%rbp)
52 movq -24(%rbp), %rax
53 addq $8, %rax
54 movl $.L.str.4, %esi
55 movq %rax, %rdi
56 callq strcpy
57 movq -24(%rbp), %rcx
58 addq $264, %rcx                                # imm = 0x108
59 movl $.L.str.5, %esi
60 movq %rcx, %rdi
61 movq %rax, -64(%rbp)                            # 8-byte Spill

```

```

62  callq strcpy
63  movq  -24(%rbp), %rcx
64  movq  $2, 520(%rcx)
65  movq  -16(%rbp), %rcx
66  movq  -8(%rbp), %rdx
67  movq  %rcx, (%rdx)
68  movq  -24(%rbp), %rcx
69  movq  -16(%rbp), %rdx
70  movq  %rcx, (%rdx)
71  movq  -24(%rbp), %rcx
72  movq  $0, (%rcx)
73  movq  -8(%rbp), %rcx
74  movq  %rax, -72(%rbp)      # 8-byte Spill
75  movq  %rcx, %rax
76  addq  $80, %rsp
77  popq  %rbp
78  .cfi_def_cfa %rsp, 8
79  retq
80 .Lfunc_end0:
81  .size setup_users, .Lfunc_end0-setup_users
82  .cfi_endproc
83
84          # -- End function
85  .globl  print_users      # -- Begin function print_users
86  .p2align 4, 0x90
87  .type  print_users,@function
88  print_users:              # @print_users
89  .cfi_startproc
90  # %bb.0:
91  pushq %rbp
92  .cfi_def_cfa_offset 16
93  .cfi_offset %rbp, -16
94  movq  %rsp, %rbp
95  .cfi_def_cfa_register %rbp
96  subq  $16, %rsp
97  movq  %rdi, -8(%rbp)
98  movabsq $.L.str.6, %rdi
99  movb  $0, %al
100 callq printf
101 movq  $0, -16(%rbp)
102 .LBB1_1:                  # =>This Inner Loop Header: Depth=1
103 cmpq  $0, -8(%rbp)
104 je    .LBB1_3
105 # %bb.2:                  # in Loop: Header=BB1_1 Depth=1
106 movq  -16(%rbp), %rax
107 addq  $1, %rax
108 movq  %rax, -16(%rbp)
109 movq  -8(%rbp), %rcx
110 addq  $8, %rcx
111 movabsq $.L.str.7, %rdi
112 movq  %rax, %rsi
113 movq  %rcx, %rdx
114 movb  $0, %al
115 callq printf
116 movq  -8(%rbp), %rcx
117 movq  (%rcx), %rcx
118 movq  %rcx, -8(%rbp)
119 jmp   .LBB1_1

```

```

119 .LBB1_3:
120     movabsq $.L.str.8, %rdi
121     movb $0, %al
122     callq printf
123     addq $16, %rsp
124     popq %rbp
125     .cfi_def_cfa %rsp, 8
126     retq
127 .Lfunc_end1:
128     .size print_users, .Lfunc_end1-print_users
129     .cfi_endproc
130
131     # -- End function
132     # -- Begin function getUser
133     .globl getUser
134     .p2align 4, 0x90
135     .type getUser,@function
136     # @getUser
137     .cfi_startproc
138     # %bb.0:
139     pushq %rbp
140     .cfi_def_cfa_offset 16
141     .cfi_offset %rbp, -16
142     movq %rsp, %rbp
143     .cfi_def_cfa_register %rbp
144     subq $32, %rsp
145     movq %rdi, -16(%rbp)
146     movq %rsi, -24(%rbp)
147
148     .LBB2_1:
149     cmpq $0, -16(%rbp)
150     je .LBB2_5
151     # %bb.2:
152     movq -16(%rbp), %rax
153     addq $8, %rax
154     movq -24(%rbp), %rsi
155     movq %rax, %rdi
156     callq strcmp
157     cmpl $0, %eax
158     jne .LBB2_4
159     # %bb.3:
160     movq -16(%rbp), %rax
161     movq %rax, -8(%rbp)
162     jmp .LBB2_6
163
164     .LBB2_4:
165     movq -16(%rbp), %rax
166     movq (%rax), %rax
167     movq %rax, -16(%rbp)
168     jmp .LBB2_1
169
170     .LBB2_5:
171     movq $0, -8(%rbp)
172
173     .LBB2_6:
174     movq -8(%rbp), %rax
175     addq $32, %rsp
176     popq %rbp
177     .cfi_def_cfa %rsp, 8
178     retq
179
180 .Lfunc_end2:
181     .size getUser, .Lfunc_end2-getUser
182     .cfi_endproc

```

```

176                                     # -- End function
177                                     # -- Begin function main
178 .globl main
179 .p2align 4, 0x90
180 .type main,@function
181 main:                                     # @main
182 .cfi_startproc
183 # %bb.0:
184 pushq %rbp
185 .cfi_def_cfa_offset 16
186 .cfi_offset %rbp, -16
187 movq %rsp, %rbp
188 .cfi_def_cfa_register %rbp
189 subq $576, %rsp                        # imm = 0x240
190 movl $0, -4(%rbp)
191 callq setup_users
192 movq %rax, -16(%rbp)
193 movabsq $.L.str.9, %rdi
194 movb $0, %al
195 callq printf
196 movabsq $.L.str.10, %rdi
197 movl %eax, -548(%rbp)                  # 4-byte Spill
198 movb $0, %al
199 callq printf
200 leaq -272(%rbp), %rsi
201 movabsq $.L.str.11, %rdi
202 movl %eax, -552(%rbp)                  # 4-byte Spill
203 movb $0, %al
204 callq __isoc99_scanf
205 leaq -272(%rbp), %rsi
206 movq -16(%rbp), %rdi
207 movl %eax, -556(%rbp)                  # 4-byte Spill
208 callq getUser
209 movq %rax, -280(%rbp)
210 cmpq $0, -280(%rbp)
211 jne .LBB3_2
212 # %bb.1:
213 leaq -272(%rbp), %rsi
214 movabsq $.L.str.12, %rdi
215 movb $0, %al
216 callq printf
217 movl $0, -4(%rbp)
218 jmp .LBB3_5
219 .LBB3_2:
220 movabsq $.L.str.13, %rdi
221 movb $0, %al
222 callq printf
223 leaq -544(%rbp), %rsi
224 movabsq $.L.str.11, %rdi
225 movl %eax, -560(%rbp)                  # 4-byte Spill
226 movb $0, %al
227 callq __isoc99_scanf
228 leaq -544(%rbp), %rsi
229 movq -280(%rbp), %rcx
230 addq $264, %rcx                        # imm = 0x108
231 movq %rcx, %rdi
232 movl %eax, -564(%rbp)                  # 4-byte Spill
233 callq strcmp

```

```

233  cmpl  $0, %eax
234  je    .LBB3_4
235  # %bb.3:
236  movabsq $.L.str.14, %rdi
237  movb  $0, %al
238  callq printf
239  movl  $0, -4(%rbp)
240  jmp   .LBB3_5
241 .LBB3_4:
242  movq  -280(%rbp), %rax
243  addq  $8, %rax
244  movabsq $.L.str.15, %rdi
245  movq  %rax, %rsi
246  movb  $0, %al
247  callq printf
248  movabsq $.L.str.8, %rdi
249  movl  %eax, -568(%rbp)      # 4-byte Spill
250  movb  $0, %al
251  callq printf
252  movq  -280(%rbp), %rcx
253  addq  $8, %rcx
254  movabsq $.L.str.16, %rdi
255  movq  %rcx, %rsi
256  movl  %eax, -572(%rbp)      # 4-byte Spill
257  movb  $0, %al
258  callq printf
259  movq  -280(%rbp), %rcx
260  movq  520(%rcx), %rsi
261  movabsq $.L.str.17, %rdi
262  movl  %eax, -576(%rbp)      # 4-byte Spill
263  movb  $0, %al
264  callq printf
265 .LBB3_5:
266  movl  -4(%rbp), %eax
267  addq  $576, %rsp           # imm = 0x240
268  popq  %rbp
269  .cfi_def_cfa %rsp, 8
270  retq
271 .Lfunc_end3:
272  .size main, .Lfunc_end3-main
273  .cfi_endproc
274                                     # -- End function
275  .type .L.str,@object           # @.str
276  .section .rodata.str1.1,"aMS",@progbits,1
277 .L.str:
278  .asciz "admin"
279  .size .L.str, 6
280
281  .type .L.str.1,@object         # @.str.1
282 .L.str.1:
283  .asciz "4dm1n__4eva"
284  .size .L.str.1, 12
285
286  .type .L.str.2,@object         # @.str.2
287 .L.str.2:
288  .asciz "alice"
289  .size .L.str.2, 6

```

```

290
291     .type .L.str.3,@object          # @.str.3
292 .L.str.3:
293     .asciz "!alice12!_veuje@@hak"
294     .size .L.str.3, 21
295
296     .type .L.str.4,@object          # @.str.4
297 .L.str.4:
298     .asciz "abdul"
299     .size .L.str.4, 6
300
301     .type .L.str.5,@object          # @.str.5
302 .L.str.5:
303     .asciz "passwOrd123"
304     .size .L.str.5, 12
305
306     .type .L.str.6,@object          # @.str.6
307 .L.str.6:
308     .asciz "--- USERS ---\n"
309     .size .L.str.6, 15
310
311     .type .L.str.7,@object          # @.str.7
312 .L.str.7:
313     .asciz "%02ld. %s\n"
314     .size .L.str.7, 12
315
316     .type .L.str.8,@object          # @.str.8
317 .L.str.8:
318     .asciz "\n"
319     .size .L.str.8, 2
320
321     .type .L.str.9,@object          # @.str.9
322 .L.str.9:
323     .asciz "Welcome to BigBank Australia!\n"
324     .size .L.str.9, 31
325
326     .type .L.str.10,@object         # @.str.10
327 .L.str.10:
328     .asciz "Username: "
329     .size .L.str.10, 11
330
331     .type .L.str.11,@object         # @.str.11
332 .L.str.11:
333     .asciz "%255s"
334     .size .L.str.11, 6
335
336     .type .L.str.12,@object         # @.str.12
337 .L.str.12:
338     .asciz "User < %s > does not exist.\n"
339     .size .L.str.12, 29
340
341     .type .L.str.13,@object         # @.str.13
342 .L.str.13:
343     .asciz "Password: "
344     .size .L.str.13, 11
345
346     .type .L.str.14,@object         # @.str.14

```

```

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

```

## F.12 password-O3.s

```

1 .text
2 .file "password.c"
3 .globl setup_users          # -- Begin function setup_users
4 .p2align 4, 0x90
5 .type setup_users,@function
6 setup_users:                # @setup_users
7 .cfi_startproc
8 # %bb.0:
9 pushq %r14
10 .cfi_def_cfa_offset 16
11 pushq %rbx
12 .cfi_def_cfa_offset 24
13 pushq %rax
14 .cfi_def_cfa_offset 32
15 .cfi_offset %rbx, -24
16 .cfi_offset %r14, -16
17 movl $528, %edi             # imm = 0x210
18 callq malloc
19 movq %rax, %r14
20 movl $1768776801, 8(%rax)    # imm = 0x696D6461
21 movw $110, 12(%rax)
22 movabsq $3773839939640058932, %rax # imm = 0x345F5F6E316D6434
23 movq %rax, 264(%r14)
24 movl $6387301, 272(%r14)    # imm = 0x617665
25 movq $1000000, 520(%r14)    # imm = 0xF4240

```

```

26  movl  $528, %edi          # imm = 0x210
27  callq malloc
28  movq  %rax, %rbx
29  movl  $1667853409, 8(%rax) # imm = 0x63696C61
30  movw  $101, 12(%rax)
31  movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
32  movq  %rax, 277(%rbx)
33  movups .L.str.3(%rip), %xmm0
34  movups %xmm0, 264(%rbx)
35  movq  $783, 520(%rbx)      # imm = 0x30F
36  movl  $528, %edi          # imm = 0x210
37  callq malloc
38  movl  $1969513057, 8(%rax) # imm = 0x75646261
39  movw  $108, 12(%rax)
40  movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
41  movq  %rcx, 264(%rax)
42  movl  $3355185, 272(%rax)  # imm = 0x333231
43  movq  $2, 520(%rax)
44  movq  %rbx, (%r14)
45  movq  %rax, (%rbx)
46  movq  $0, (%rax)
47  movq  %r14, %rax
48  addq  $8, %rsp
49  .cfi_def_cfa_offset 24
50  popq  %rbx
51  .cfi_def_cfa_offset 16
52  popq  %r14
53  .cfi_def_cfa_offset 8
54  retq
55 .Lfunc_end0:
56  .size setup_users, .Lfunc_end0-setup_users
57  .cfi_endproc
58                                     # -- End function
59  .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:
65  pushq %r14
66  .cfi_def_cfa_offset 16
67  pushq %rbx
68  .cfi_def_cfa_offset 24
69  pushq %rax
70  .cfi_def_cfa_offset 32
71  .cfi_offset %rbx, -24
72  .cfi_offset %r14, -16
73  movq  %rdi, %r14
74  movl  $.Lstr, %edi
75  callq puts
76  testq %r14, %r14
77  je    .LBB1_3
78 # %bb.1:
79  movl  $1, %ebx
80  .p2align 4, 0x90
81 .LBB1_2:                            # =>This Inner Loop Header: Depth=1
82  leaq  8(%r14), %rdx

```



```

83  movl  $.L.str.7, %edi
84  movq  %rbx, %rsi
85  xorl  %eax, %eax
86  callq printf
87  movq  (%r14), %r14
88  addq  $1, %rbx
89  testq %r14, %r14
90  jne   .LBB1_2
91 .LBB1_3:
92  movl  $10, %edi
93  addq  $8, %rsp
94  .cfi_def_cfa_offset 24
95  popq  %rbx
96  .cfi_def_cfa_offset 16
97  popq  %r14
98  .cfi_def_cfa_offset 8
99  jmp   putchar                # TAILCALL
100 .Lfunc_end1:
101  .size print_users, .Lfunc_end1-print_users
102  .cfi_endproc
103
104  .globl getUser                # -- End function
105  .p2align 4, 0x90             # -- Begin function getUser
106  .type  getUser,@function
107  getUser:                      # @getUser
108  .cfi_startproc
109  # %bb.0:
110  pushq %r14
111  .cfi_def_cfa_offset 16
112  pushq %rbx
113  .cfi_def_cfa_offset 24
114  pushq %rax
115  .cfi_def_cfa_offset 32
116  .cfi_offset %rbx, -24
117  .cfi_offset %r14, -16
118  testq %rdi, %rdi
119  je    .LBB2_4
120  # %bb.1:
121  movq  %rsi, %r14
122  movq  %rdi, %rbx
123  .p2align 4, 0x90
124  .LBB2_2:                      # =>This Inner Loop Header: Depth=1
125  leaq  8(%rbx), %rdi
126  movq  %r14, %rsi
127  callq strcmp
128  testl %eax, %eax
129  je    .LBB2_5
130  # %bb.3:                      # in Loop: Header=BB2_2 Depth=1
131  movq  (%rbx), %rbx
132  testq %rbx, %rbx
133  jne   .LBB2_2
134  .LBB2_4:
135  xorl  %ebx, %ebx
136  .LBB2_5:
137  movq  %rbx, %rax
138  addq  $8, %rsp
139  .cfi_def_cfa_offset 24

```

```

140 popq %rbx
141 .cfi_def_cfa_offset 16
142 popq %r14
143 .cfi_def_cfa_offset 8
144 retq
145 .Lfunc_end2:
146 .size getUser, .Lfunc_end2-getUser
147 .cfi_endproc
148                                     # -- End function
149 .globl main                         # -- Begin function main
150 .p2align 4, 0x90
151 .type main,@function
152 main:                               # @main
153 .cfi_startproc
154 # %bb.0:
155 pushq %r15
156 .cfi_def_cfa_offset 16
157 pushq %r14
158 .cfi_def_cfa_offset 24
159 pushq %rbx
160 .cfi_def_cfa_offset 32
161 subq $512, %rsp                    # imm = 0x200
162 .cfi_def_cfa_offset 544
163 .cfi_offset %rbx, -32
164 .cfi_offset %r14, -24
165 .cfi_offset %r15, -16
166 movl $528, %edi                  # imm = 0x210
167 callq malloc
168 movq %rax, %rbx
169 movl $1768776801, 8(%rax)        # imm = 0x696D6461
170 movw $110, 12(%rax)
171 movabsq $3773839939640058932, %rax # imm = 0x345F5F6E316D6434
172 movq %rax, 264(%rbx)
173 movl $6387301, 272(%rbx)        # imm = 0x617665
174 movq $1000000, 520(%rbx)       # imm = 0xF4240
175 movl $528, %edi                # imm = 0x210
176 callq malloc
177 movq %rax, %r14
178 movl $1667853409, 8(%rax)      # imm = 0x63696C61
179 movw $101, 12(%rax)
180 movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
181 movq %rax, 277(%r14)
182 movups .L.str.3(%rip), %xmm0
183 movups %xmm0, 264(%r14)
184 movq $783, 520(%r14)          # imm = 0x30F
185 movl $528, %edi              # imm = 0x210
186 callq malloc
187 movl $1969513057, 8(%rax)      # imm = 0x75646261
188 movw $108, 12(%rax)
189 movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
190 movq %rcx, 264(%rax)
191 movl $3355185, 272(%rax)      # imm = 0x333231
192 movq $2, 520(%rax)
193 movq %r14, (%rbx)
194 movq %rax, (%r14)
195 movq $0, (%rax)
196 movl $.Lstr.18, %edi

```

```

197  callq puts
198  movl $.L.str.10, %edi
199  xorl %eax, %eax
200  callq printf
201  movq %rsp, %rsi
202  movl $.L.str.11, %edi
203  xorl %eax, %eax
204  callq __isoc99_scanf
205  testq %rbx, %rbx
206  je .LBB3_4
207 # %bb.1:
208  movq %rsp, %r15
209  .p2align 4, 0x90
210 .LBB3_2:                                     # =>This Inner Loop Header: Depth=1
211  leaq 8(%rbx), %r14
212  movq %r14, %rdi
213  movq %r15, %rsi
214  callq strcmp
215  testl %eax, %eax
216  je .LBB3_5
217 # %bb.3:                                     # in Loop: Header=BB3_2 Depth=1
218  movq (%rbx), %rbx
219  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
227  jmp .LBB3_9
228 .LBB3_5:
229  movl $.L.str.13, %edi
230  xorl %eax, %eax
231  callq printf
232  leaq 256(%rsp), %r15
233  movl $.L.str.11, %edi
234  movq %r15, %rsi
235  xorl %eax, %eax
236  callq __isoc99_scanf
237  leaq 264(%rbx), %rdi
238  movq %r15, %rsi
239  callq strcmp
240  testl %eax, %eax
241  je .LBB3_7
242 # %bb.6:
243  movl $.L.str.19, %edi
244  callq puts
245 .LBB3_9:
246  xorl %eax, %eax
247  addq $512, %rsp                               # imm = 0x200
248  .cfi_def_cfa_offset 32
249  popq %rbx
250  .cfi_def_cfa_offset 24
251  popq %r14
252  .cfi_def_cfa_offset 16
253  popq %r15

```

```

254 .cfi_def_cfa_offset 8
255 retq
256 .LBB3_7:
257 .cfi_def_cfa_offset 544
258 movl $.L.str.15, %edi
259 movq %r14, %rsi
260 xorl %eax, %eax
261 callq printf
262 movl $10, %edi
263 callq putchar
264 movl $.L.str.16, %edi
265 movq %r14, %rsi
266 xorl %eax, %eax
267 callq printf
268 movq 520(%rbx), %rsi
269 movl $.L.str.17, %edi
270 jmp .LBB3_8
271 .Lfunc_end3:
272 .size main, .Lfunc_end3-main
273 .cfi_endproc
274                                     # -- End function
275 .type .L.str,@object                # @.str
276 .section .rodata.str1.1,"aMS",@progbits,1
277 .L.str:
278 .asciz "admin"
279 .size .L.str, 6
280
281 .type .L.str.1,@object              # @.str.1
282 .L.str.1:
283 .asciz "4dm1n_4eva"
284 .size .L.str.1, 12
285
286 .type .L.str.2,@object              # @.str.2
287 .L.str.2:
288 .asciz "alice"
289 .size .L.str.2, 6
290
291 .type .L.str.3,@object              # @.str.3
292 .L.str.3:
293 .asciz "!alice12!_veuje@@hak"
294 .size .L.str.3, 21
295
296 .type .L.str.4,@object              # @.str.4
297 .L.str.4:
298 .asciz "abdul"
299 .size .L.str.4, 6
300
301 .type .L.str.5,@object              # @.str.5
302 .L.str.5:
303 .asciz "passwOrd123"
304 .size .L.str.5, 12
305
306 .type .L.str.7,@object              # @.str.7
307 .L.str.7:
308 .asciz " %02ld. %s\n"
309 .size .L.str.7, 12
310

```

```

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

```

## F.13 deadStoreElimination-O0.s

```

1 .text

```

```

2  .file "deadStoreElimination.c"
3  .globl deadStore          # -- Begin function deadStore
4  .p2align 4, 0x90
5  .type deadStore,@function
6  deadStore:                # @deadStore
7  .cfi_startproc
8  # %bb.0:
9  pushq %rbp
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl %edi, -4(%rbp)
15 movl %esi, -8(%rbp)
16 movl $43981, -12(%rbp)    # imm = 0xABCD
17 movl $0, -16(%rbp)
18 .LBB0_1:                  # =>This Inner Loop Header: Depth=1
19 movl -4(%rbp), %eax
20 cmpl -8(%rbp), %eax
21 jle .LBB0_3
22 # %bb.2:                  # in Loop: Header=BB0_1 Depth=1
23 movl -12(%rbp), %eax
24 addl -16(%rbp), %eax
25 movl %eax, -16(%rbp)
26 movl -4(%rbp), %eax
27 addl $-1, %eax
28 movl %eax, -4(%rbp)
29 jmp .LBB0_1
30 .LBB0_3:
31 movl $0, -12(%rbp)
32 movl -4(%rbp), %eax
33 addl -8(%rbp), %eax
34 popq %rbp
35 .cfi_def_cfa %rsp, 8
36 retq
37 .Lfunc_end0:
38 .size deadStore, .Lfunc_end0-deadStore
39 .cfi_endproc
40                                     # -- End function
41 .globl main                  # -- Begin function main
42 .p2align 4, 0x90
43 .type main,@function
44 main:                        # @main
45 .cfi_startproc
46 # %bb.0:
47 pushq %rbp
48 .cfi_def_cfa_offset 16
49 .cfi_offset %rbp, -16
50 movq %rsp, %rbp
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
57 callq deadStore
58 xorl %ecx, %ecx

```

```

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

```

## F.14 deadStoreElimination-O3.s

```

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

```

## F.15 pread-O0.s

```

1  .text
2  .file "pread.c"
3  .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
10 .cfi_def_cfa_offset 16
11 .cfi_offset %rbp, -16
12 movq %rsp, %rbp
13 .cfi_def_cfa_register %rbp
14 movl $0, -4(%rbp)
15 movl $0, -8(%rbp)
16 movl $0, -12(%rbp)
17 .LBB0_1:                             # =>This Loop Header: Depth=1
18                                     #   Child Loop BB0_2 Depth 2
19                                     #   Child Loop BB0_3 Depth 3
20     jmp .LBB0_2
21 .LBB0_2:                             #   Parent Loop BB0_1 Depth=1
22                                     # => This Loop Header: Depth=2
23                                     #   Child Loop BB0_3 Depth 3
24     jmp .LBB0_3
25 .LBB0_3:                             #   Parent Loop BB0_1 Depth=1
26                                     #   Parent Loop BB0_2 Depth=2
27                                     # => 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
31     movl -8(%rbp), %eax
32     cltd
33     movl $2, %ecx
34     idivl %ecx
35     cmpl $0, %edx
36     jne .LBB0_3
37 # %bb.5:                             #   in Loop: Header=BB0_2 Depth=2
38     movl x, %eax
39     movl %eax, -12(%rbp)
40 # %bb.6:                             #   in Loop: Header=BB0_2 Depth=2
41     movl z, %eax
42     cmpl -8(%rbp), %eax
43     jne .LBB0_2
44 # %bb.7:                             #   in Loop: Header=BB0_1 Depth=1
45     jmp .LBB0_1
46 .Lfunc_end0:
47     .size main, .Lfunc_end0-main
48     .cfi_endproc
49                                     # -- End function
50 .type z,@object                       # @z
51 .comm z,4,4
52 .type x,@object                       # @x
53 .comm x,4,4
54 .ident "clang version 10.0.0-4ubuntu1 "
55 .section ".note.GNU-stack","",@progbits
56 .addrsig
57 .addrsig_sym z
58 .addrsig_sym x

```

## F.16 pread-O3.s



```

1  .text
2  .file "pread.c"
3  .globl main                                # -- Begin function main
4  .p2align 4, 0x90
5  .type main,@function
6 main:                                       # @main
7  .cfi_startproc
8  # %bb.0:
9  .p2align 4, 0x90
10 .LBB0_1:                                  # =>This Inner Loop Header: Depth=1
11  movl z(%rip), %eax
12  testb $1, %al
13  jne .LBB0_1
14 # %bb.2:                                  # in Loop: Header=BB0_1 Depth=1
15  movl x(%rip), %eax
16  movl z(%rip), %eax
17  jmp .LBB0_1
18 .Lfunc_end0:
19  .size main, .Lfunc_end0-main
20  .cfi_endproc
21                                           # -- End function
22  .type z,@object                          # @z
23  .comm z,4,4
24  .type x,@object                          # @x
25  .comm x,4,4
26  .ident "clang version 10.0.0-4ubuntu1 "
27  .section ".note.GNU-stack","",@progbits
28  .addrsig
29  .addrsig_sym z
30  .addrsig_sym x

```