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

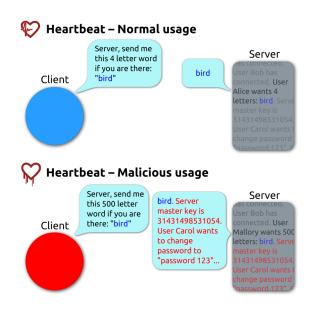


Figure 1: The Heartbleed bug. [13]

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,

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

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 on insecure variables. Additionally, to protect integrity, untrusted data from public sources must be prevented from flowing into secure or trusted destinations [2]. An information flow security policy can be introduced to classify or label data, or more formally, a set of security levels to which each object is bound by across a multi-level security lattice [8]. In this thesis, we will focus primarily on preserving confidentiality.

Many security levels can be identified to classify different classes of objects, however, for now we will consider two security levels: high and low. Data labelled as high signifies that the data is secret, and

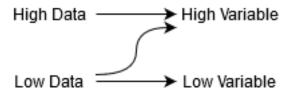


Figure 3: Permitted flow of data

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 possibilities for the overall behaviour. This concept of assumptions (or rely) and guarantee conditions can reduce the complexity of understanding interleaving behaviour in threads and assist in verifying the correctness of information flow security in concurrency. However, this approach is limited in the types of assumptions and guarantees it supports. Building on this, Murray et al. [12] [21] provide information flow logic on how to handle dynamic, value-dependent security levels in concurrent programs. In this case, the security level of a particular variable may depend on one or more other variables in the program. As such, the variable's security level can change as the state of the program changes. This logic is essential where the security level of data depends on its source. However, this approach is not sufficient when analysing non-blocking programs. The approach relies heavily on locks which block particular threads from executing. This in turn leads to slower processing due to blocked threads [23].

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

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-

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

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 counterinitiative. Additionally, compilers are getting better at optimising away tricks programmers write to work against the compiler, and thus is not a future-proof

solution [26]. One might also suggest turning compiler optimisations off, however, this leads to slower code. In a concurrent system where execution time is critical, turning compiler optimisations off is not a viable option. Instead an alternative solution is to perform a static analysis on binary or assembly for security violations. As compilation has already been executed, such analysis would reveal security guarantee violations that result due to compiler optimisations.

2.5 Annotations

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

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

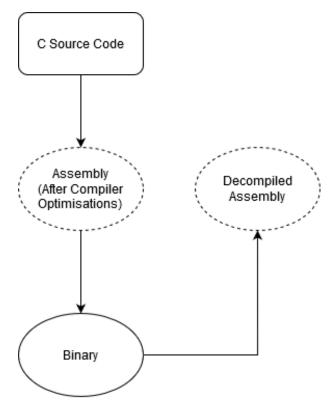


Figure 5: The static analysis options after compilation.

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

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

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

assembler files [11].

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

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

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

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

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

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

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

Table 1: CompCert install specifications

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.*

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

Listing 1: comment.c

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

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

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

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

Listing 2: comment-O0.s

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

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

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

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

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

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

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

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

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

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

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

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

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

90

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

Listing 3: rooster-O0.s

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

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

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

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

Listing 4: password.c

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

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

Listing 5: password-O0.s

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

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:

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

Listing 6: comment-O3.s diff

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

Next, variable-O3.s will be compared.

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

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

Listing 7: variable-O3.s diff

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

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

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

Listing 8: count.c

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

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

Listing 9: count-O0.s

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

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

Listing 10: count-O3.s

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

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.

```
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.

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

Listing 12: comment-O0.s

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

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

Following, the compiled output would be:

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

One may suggest using the APP and NO_APP comments above and below our annotation to identify it, however, these comments are system and compiler dependent. As such, a more robust solution has been used here.

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

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

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

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

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

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

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

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

Listing 13: variable-O0.s

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

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

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

Listing 14: variable-O3.s

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

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

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

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

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

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

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

Listing 15: variable-O0.s

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

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

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

Listing 16: variable-O3.s

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

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

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

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

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

Listing 17: variable-O3.s

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

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

```
volatile int x;

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

Listing 18: volatile.c

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

```
8 # %bb.0:
    movl x(%rip), %eax
    #APP
10
    # annotation: %eax = High
11
    #NO_APP
    movl
          x(%rip), %eax
13
    addl
14
          $1, %eax
    reta
15
16 .Lfunc_end0:
    .size main, .Lfunc_end0-main
17
18
    .cfi_endproc
      -- End function
    .type x,@object
                                     # @x
    .comm x,4,4
21
    .ident "clang version 10.0.0-4ubuntu1"
22
    .section ".note.GNU-stack","", @progbits
23
    .addrsig
24
    .addrsig_sym x
```

Listing 19: volatile-O3.s

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

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

4.2.1 Quality Analysis

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

One proposed method of handling this is to build a tool to assist in this parsing and analysis. This tool is covered in section 4.2.2.

TODO: Quality analysis

4.2.2 Tool Assisted Annotations

- 1. Install specifications
- 2. Discuss implications of additional move statements
- 3. Quality Analysis -
- 4. speed comparison

Tool building:

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

4.3 CompCert Builtin Annotations

TODO: Revisit compaert using builtin annoations 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 Interna*tional Workshop on Information Security Theory and Practice. Springer. 2012, pp. 1–16.
- [2] Musard Balliu. "Logics for information flow security: from specification to verification". PhD thesis. KTH Royal Institute of Technology, 2014.
- [3] Nick Benton. "Simple relational correctness proofs for static analyses and program transformations". In: *ACM SIGPLAN Notices* 39.1 (2004), pp. 14–25.
- [4] CompCert The CompCert C compiler. Accessed: 2020-09-01. 2020. URL: http://compcert.inria.fr/compcert-C.html.
- [5] Compiler Optimization Removal or Modification of Security-critical Code. Accessed: 2020-09-01. 2008. URL: https://cwe.mitre.org/ data/definitions/733.html.
- [6] Compiler Removal of Code to Clear Buffers. Accessed: 2020-09-01. 2006. URL: https:// cwe.mitre.org/data/definitions/14. html.
- [7] Vijay D'Silva, Mathias Payer, and Dawn Song. "The correctness-security gap in compiler optimization". In: 2015 IEEE Security and Privacy Workshops. IEEE. 2015, pp. 73–87.
- [8] Dorothy E Denning. "A lattice model of secure information flow". In: Communications of the ACM 19.5 (1976), pp. 236–243.
- [9] Dorothy E Denning and Peter J Denning. "Certification of programs for secure information flow". In: Communications of the ACM 20.7 (1977), pp. 504–513.
- [10] Quang Do, Ben Martini, and Kim-Kwang Raymond Choo. "The role of the adversary model in applied security research". In: Computers & Security 81 (2019), pp. 156–181.

- [11] K Eder, K Georgiou, and N Grech. Common Assertion Language. ENTRA Project: Whole-Systems Energy Transparency (FET project 318337). Deliverable 2.1. 2013.
- [12] Gidon Ernst and Toby Murray. "SecCSL: Security concurrent separation logic". In: International Conference on Computer Aided Verification. Springer. 2019, pp. 208–230.
- [13] File:Simplified Heartbleed explanation.svg Wikimedia Commons. Accessed: 2020-09-02. 2014. URL: https://commons.wikimedia.org/wiki/File:Simplified_Heartbleed_explanation.svg # mediaviewer / File:Simplified_Heartbleed_explanation.svg.
- [14] GCC Developer Options. Accessed: 2020-09-02. URL: https://gcc.gnu.org/onlinedocs/ gcc/Developer-Options.html.
- [15] Barbara Guttman and Edward A Roback. An introduction to computer security: the NIST handbook. Diane Publishing, 1995.
- [16] Heartbleed Bug. Accessed: 2020-09-02. 2020. URL: https://heartbleed.com/.
- [17] Xavier Leroy. "Formal certification of a compiler back-end or: programming a compiler with a proof assistant". In: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2006, pp. 42–54.
- [18] Xavier Leroy et al. "CompCert-a formally verified optimizing compiler". In: 2016.
- [19] Heiko Mantel, Matthias Perner, and Jens Sauer. "Noninterference under weak memory models". In: 2014 IEEE 27th Computer Security Foundations Symposium. IEEE. 2014, pp. 80–94.
- [20] Heiko Mantel, David Sands, and Henning Sudbrock. "Assumptions and guarantees for compositional noninterference". In: 2011 IEEE 24th Computer Security Foundations Symposium. IEEE. 2011, pp. 218–232.

- [21] Toby Murray, Robert Sison, and Kai Engelhardt. "COVERN: A logic for compositional verification of information flow control". In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE. 2018, pp. 16–30.
- [22] Options That Control Optimization. Accessed: 2020-09-01. URL: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html.
- [23] Sundeep Prakash, Yann-Hang Lee, and Theodore Johnson. "Non-blocking algorithms for concurrent data structures". MA thesis. Citeseer, 1991.
- [24] Andrei Sabelfeld and Andrew C Myers. "Language-based information-flow security". In: *IEEE Journal on selected areas in communications* 21.1 (2003), pp. 5–19.
- [25] Bernhard Schommer et al. "Embedded program annotations for WCET analysis". In: 18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.
- [26] Laurent Simon, David Chisnall, and Ross Anderson. "What you get is what you C: Controlling side effects in mainstream C compilers". In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE. 2018, pp. 1–15.
- [27] Graeme Smith, Nicholas Coughlin, and Toby Murray. "Value-Dependent Information-Flow Security on Weak Memory Models". In: *International Symposium on Formal Methods*. Springer. 2019, pp. 539–555.
- [28] William Stallings et al. Computer security: principles and practice. Pearson Education Upper Saddle River, NJ, USA, 2012.
- [29] Jeffrey A Vaughan and Todd Millstein. "Secure information flow for concurrent programs under total store order". In: 2012 IEEE 25th Computer Security Foundations Symposium. IEEE. 2012, pp. 19–29.

- [30] Son Tuan Vu et al. "Secure delivery of program properties through optimizing compilation". In: *Proceedings of the 29th International Conference on Compiler Construction*. 2020, pp. 14–26.
- [31] Kirsten Winter, Graeme Smith, and Nicholas Coughlin. "Information flow security in the presence of fine-grained concurrency". Unpublished. Aug. 2020.

Appendices

A Test C Programs

A.1 comment.c

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

A.2 variable.c

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

A.3 volatile.c

```
volatile int x;

int main() {
    // L(x) = High
    return x + 1;
6 }
```

A.4 loop.c

```
int z;
int x;

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

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

int main() {
```

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

A.5 rooster.c

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

A.6 password.c

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
    user_t* next;
10
      char name[BUFF_LEN];
11
      // L(password) = High
12
     char password[BUFF_LEN];
13
14
      size_t balance;
15 };
16
17 user_t* setup_users() {
     user_t* user_admin = malloc(sizeof(user_t));
18
19
      strcpy(user_admin->name, "admin");
      strcpy(user_admin->password, "4dm1n__4eva");
20
21
      user_admin->balance = 1000000;
22
     user_t* user_alice = malloc(sizeof(user_t));
23
      strcpy(user_alice->name, "alice");
24
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
26
      user_alice->balance = 783;
27
     user_t* user_abdul = malloc(sizeof(user_t));
28
      strcpy(user_abdul->name, "abdul");
29
      strcpy(user_abdul ->password, "passw0rd123");
30
31
      user_abdul ->balance = 2;
32
     user_admin->next = user_alice;
33
      user_alice->next = user_abdul;
34
      user_abdul ->next = NULL;
35
36
      return user_admin;
37
38 }
39
40 void print_users(user_t* users) {
      printf("--- USERS ---\n");
41
      size_t count = 0;
42
43
      while (users != NULL) {
          printf(" %021d. %s\n", ++count, users->name);
44
          users = users->next;
45
46
      printf("\n");
47
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
      while (user_list != NULL) {
51
          if (strcmp(user_list->name, name) == 0) {
52
53
              return user_list;
54
55
          user_list = user_list->next;
56
return NULL;
```

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

A.7 deadStoreElimination.c

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

A.8 pread.c

```
volatile int z;
volatile int x;
```

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

B CompCert Annotated C Programs

B.1 comment.c

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

B.2 variable.c

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

17 }

B.3 volatile.c

```
volatile int x;

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

6 }
```

B.4 loop.c

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

B.5 rooster.c

```
int rooster;
int drake;
int goose;

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

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

B.6 password.c

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
   user_t* next;
10
11
     char name[BUFF_LEN];
                                 //
                                             { L(password) = High }
     char password[BUFF_LEN];
12
13
     size_t balance;
14 };
15
16 user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
17
      strcpy(user_admin->name, "admin");
18
     strcpy(user_admin->password, "4dm1n_{-}4eva");
19
      __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
20
21
      user_admin->balance = 1000000;
22
     user_t* user_alice = malloc(sizeof(user_t));
24
      strcpy(user_alice->name, "alice");
     strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
      __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
26
     user_alice->balance = 783;
27
      user_t* user_abdul = malloc(sizeof(user_t));
29
   strcpy(user_abdul->name, "abdul");
```

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

```
printf("Your balance: $%1d\n", user->balance);
89 }
```

B.7 deadStoreElimination.c

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

B.8 pread.c

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

C CompCert Assembly Output

C.1 comment-O0.s

```
# File generated by CompCert 3.7
2 # Command line: comment.c -S -00 -o compCert/out/comment-00.s
   .align 16
   .globl main
6 main:
   .cfi_startproc
   subq $8, %rsp
    .cfi_adjust_cfa_offset 8
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
xorl %eax, %eax
10
11
12
    addq $8, %rsp
   ret
14
   .cfi_endproc
15
   .type main, @function
.size main, . - main
```

C.2 comment-O3.s

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

C.3 variable-O0.s

```
# File generated by CompCert 3.7

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

.text

align 16

glob1 main

main:

cfi_startproc

subq $8, %rsp

cfi_adjust_cfa_offset 8
```

```
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
movl $-10, %esi
10
11
12
    mov1 $98, %ecx
13
    negl %ecx
movl $1, %r8d
14
15
    movabsq $4294967296, %rax
16
    movsd .L100(%rip), %xmm1 # 3.14159265358979312
17
    cvtsd2ss %xmm1, %xmm3
18
    cvtss2sd %xmm3, %xmm0
19
    movsd .L101(%rip), %xmm2 # 2.3784000000000007
20
    divsd %xmm2, %xmm0
21
    cqto
22
23
    shrq $59, %rdx
    leaq 0(%rax, %rdx, 1), %rax
24
    sarq $5, %rax
leal 0(%eax, %esi, 1), %r9d
25
26
    leal 0(%r9d,%ecx,1), %r10d
27
    leal 0(%r10d,%r8d,1), %r8d
    cvttss2si %xmm3, %edx
29
    leal 0(%r8d, %edx, 1), %r8d
    cvttsd2si %xmm0, %eax
31
    leal 0(%r8d,%eax,1), %r11d
32
33
    leal 0(%r11d, %edi, 1), %eax
    addq $8, %rsp
34
35
    ret
    .cfi_endproc
36
    .type main, @function
37
38
    .size main, . - main
    .section .rodata.cst8, "aM", @progbits,8
39
    .align 8
41 .L100: .quad 0x400921fb54442d18
42 .L101: .quad 0x400306f694467382
```

C.4 variable-O3.s

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

C.5 volatile-O0.s

```
# File generated by CompCert 3.7
```

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

C.6 volatile-O3.s

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

C.7 loop-O0.s

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

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

C.8 loop-O3.s

```
# File generated by CompCert 3.7
2 # Command line: loop.c -S -03 -o compCert/out/loop-03.s
   .comm z, 4, 4
    .comm x, 4, 4
    .text
   .align 16
7 .globl main
8 main:
9 .cfi_startproc
   subq $8, %rsp
10
11
    .cfi_adjust_cfa_offset 8
12 leaq 16(%rsp), %rax
13 movq %rax, 0(%rsp)
14 .L100:
   movl z(%rip), %edx
movq %rdx, %rax
15
16
   testl %eax, %eax
17
   leal 1(%eax), %ecx
   cmovl %rcx, %rax
19
   sarl $1, %eax
leal 0(,%eax,2), %esi
20
21
   movq %rdx, %rcx
subl %esi, %ecx
22
   testl %ecx, %ecx
24
    jne .L100
    movq %rdx, %rsi
26
   jmp .L100
   .cfi_endproc
   .type main, @function
30 .size main, . - main
```

C.9 rooster-O0.s

```
# File generated by CompCert 3.7

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

.comm rooster, 4, 4

.comm drake, 4, 4

.comm goose, 4, 4
```

```
6 .data 7 .align 4
8 count:
9 .long 0
    .type count, @object
10
    .size count, . - count
11
12
    .text
    .align 16
13
14 .globl fun
15 fun:
    .cfi_startproc
16
    subq $8, %rsp
17
    .cfi_adjust_cfa_offset 8
18
    leaq 16(%rsp), %rax
19
   movq %rax, 0(%rsp)
leal 0(%edi,%esi,1), %r8d
leal 0(%r8d,%edx,1), %eax
20
21
22
    testl %eax, %eax
23
24
    jl .L100
    cmpl %esi, %edi
25
    jl .L101
xorl %r8d, %r8d
27
28
   jmp .L102
29 .L101:
30 cmpl %edx, %esi
31 setl %r8b
movzbl %r8b, %r8d
33 .L102:
34 cmpl $0, %r8d
   je .L103
35
36 .L104:
37 cmpl %esi, %edi
    je .L103
38
   leal 1(%edi), %edi
movl count(%rip), %eax
leal 1(%eax), %ecx
39
40
41
movl %ecx, count(%rip)
43 .L105:
   cmpl %edx, %esi
44
45
    je .L104
    leal -1(%edx), %edx
46
47
    movl count(%rip), %r9d
    leal 1(%r9d), %r8d
movl %r8d, count(%rip)
50
    jmp .L105
51 .L103:
movl count(%rip), %eax
53 .L100:
54 addq $8, %rsp
55
    ret
    .cfi_endproc
56
    .type fun, @function
57
58
    .size fun, . - fun
    .text
59
60
    .align 16
.globl main
62 main:
```

```
.cfi_startproc
63
     subq $8, %rsp
     .cfi_adjust_cfa_offset 8
65
     leaq 16(%rsp), %rax
    movq %rax, 0(%rsp)
movl $1, %eax
movl %eax, rooster(%rip)
movl $5, %eax
67
68
69
70
     movl %eax, drake(%rip)
    movl $10, %eax
movl %eax, goose(%rip)
movl rooster(%rip), %edi
movl drake(%rip), %esi
72
73
74
75
76
     movl goose(%rip), %edx
    call fun
xorl %eax, %eax
addq $8, %rsp
77
78
79
    ret
80
81 .cfi_endproc
82 .type main, @function
size main, . - main
```

C.10 rooster-O3.s

```
# File generated by CompCert 3.7
 2 # Command line: rooster.c -S -03 -o compCert/out/rooster-03.s
    .comm rooster, 4, 4
    .comm drake, 4, 4
    .comm goose, 4, 4
   .data
 7 .align 4
8 count:
    .long 0
    .type count, @object
10
11 .size count, . - count
12
   .text
.align 16.globl fun
15 fun:
.cfi_startproc
    subq $8, %rsp
17
    .cfi_adjust_cfa_offset 8
18
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
leal 0(%edi,%esi,1), %r9d
leal 0(%r9d,%edx,1), %eax
19
20
21
22
    testl %eax, %eax
23
    jl .L100
24
    cmpl %edx, %esi
25
    setl %al
27
    movzbl %al, %eax
    xorl %r8d, %r8d
cmpl %esi, %edi
28
29
cmovge %r8, %rax
31 cmpl $0, %eax
32 je .L101
33 .L102:
```

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

C.11 password-O0.s

```
# File generated by CompCert 3.7
# Command line: password.c -S -00 -o compCert/out/password-00.s
section .rodata
align 1
__stringlit_7:
ascii "--- USERS ---\012\000"
type __stringlit_7, @object
```

```
8 .size __stringlit_7, . - __stringlit_7
   .section .rodata
   .align 1
10
11 __stringlit_6:
.ascii "passw0rd123\000"
.type __stringlit_6, @object
14
   .size __stringlit_6, . - __stringlit_6
   .section .rodata
15
16 .align 1
17 __stringlit_4:
   .ascii "!alice12!_veuje@@hak\000"
18
   .type __stringlit_4, @object
19
   .size __stringlit_4, . - __stringlit_4
20
21 .section .rodata
22 .align 1
23 __stringlit_14:
   .ascii "Password: \000"
24
   .type __stringlit_14, @object
25
.size __stringlit_14, . - __stringlit_14
.section .rodata
   .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%1d\012\000"
   .type __stringlit_18, @object
   .size __stringlit_18, . - __stringlit_18
32
33
   .section .rodata
   .align 1
34
35 __stringlit_13:
.ascii "User < %s > does not exist.\012\000"
   .type __stringlit_13, @object
37
   .size __stringlit_13, . - __stringlit_13
39
   .section .rodata
   .align 1
41 __stringlit_8:
   .ascii " %021d. %s\012\000"
42
   .type __stringlit_8, @object
   .size __stringlit_8, . - __stringlit_8
44
45 .section .rodata
   .align 1
46
47 __stringlit_1:
   .ascii "admin\000"
   .type __stringlit_1, @object
49
   .size __stringlit_1, . - __stringlit_1
51
   .section .rodata
   .align 1
53 __stringlit_2:
.ascii "4dm1n_4eva\\000"
   .type __stringlit_2, @object
   .size __stringlit_2, . - __stringlit_2
56
57
   .section .rodata
   .align 1
58
59 __stringlit_3:
60 .ascii "alice\000"
   .type __stringlit_3, @object
61
   .size __stringlit_3, . - __stringlit_3
   .section .rodata
63
64 .align 1
```

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

```
movq %r12, 24(%rsp)
122
123
     movq $528, %rdi
     call malloc
124
     movq %rax, %rbp
125
     leaq 8(%rbp), %rdi
126
     leaq __stringlit_1(%rip), %rsi
call strcpy
127
128
     call
            strcpy
     leaq 264(%rbp), %rdi
129
     leaq __stringlit_2(%rip), %rsi
130
131
     call strcpy
     movq $1000000, %rax
movq %rax, 520(%rbp)
132
133
     movq $528, %rdi
134
     call malloc
     movq %rax, %r12
136
     leaq 8(%r12), %rdi
137
138
     leaq
           __stringlit_3(%rip), %rsi
     call strcpy
139
140
     leaq 264(%r12), %rdi
     leaq __stringlit_4(%rip), %rsi
call strcpy
141
     movq $783, %rsi
143
     movq %rsi, 520(%r12)
144
     movq $528, %rdi
145
     call malloc movq %rax,
146
            %rax, %rbx
147
     leaq 8(%rbx), %rdi
148
     leaq __stringlit_5(%rip), %rsi
call strcpy
149
150
     leaq 264(%rbx), %rdi
     leaq __stringlit_6(%rip), %rsi
call strcpy
152
153
     movq $2, %r10
154
     movq %r10, 520(%rbx)
155
     movq %r12, 0(%rbp)
movq %rbx, 0(%r12)
156
157
     xorq %r8, %r8
158
159
     movq %r8, 0(%rbx)
     movq %rbp, %rax
160
     movq 8(%rsp), %rbx
movq 16(%rsp), %rbp
161
162
     movq 24(%rsp), %r12
163
164
     addq $40, %rsp
165
     ret
     .cfi_endproc
166
167
     .type setup_users, @function
     .size setup_users, . - setup_users
168
     .text
     .align 16
170
     .globl print_users
172 print_users:
    .cfi_startproc
173
174
     subq $24, %rsp
     .cfi_adjust_cfa_offset 24
175
176 leaq 32(%rsp), %rax
177 movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
```

```
movq %rbp, 16(%rsp)
179
180
     movq %rdi, %rbx
            __stringlit_7(%rip), %rdi
    leaq
181
     movl $0, %eax
182
    call printf
183
    xorq %rbp, %rbp
184
185 .L100:
186 cmpq $0, %rbx
187
     je .L101
    leaq 1(%rbp), %rbp
188
     leaq __stringlit_8(%rip), %rdi
leaq 8(%rbx), %rdx
189
190
     movq %rbp, %rsi
191
     movl $0, %eax
192
    call printf
movq 0(%rbx), %rbx
jmp .L100
193
194
195
196 .L101:
call printf
movq 8(%rsp), %rbx
199
200
     movq 16(%rsp), %rbp
201
202
     addq $24, %rsp
     ret
203
204
     .cfi_endproc
    .type print_users, @function
205
    .size print_users, . - print_users
206
207
    .text
    .align 16
208
    .globl getUser
210 getUser:
    .cfi_startproc
211
     subq $24, %rsp
212
     .cfi_adjust_cfa_offset 24
213
    leaq 32(%rsp), %rax
movq %rax, 0(%rsp)
214
215
216
     movq %rbx, 8(%rsp)
    movq %rbp, 16(%rsp)
movq %rsi, %rbp
movq %rdi, %rbx
217
218
219
220 .L102:
221 cmpq $0, %rbx
    je .L103
222
    leaq 8(%rbx), %rdi
movq %rbp, %rsi
call strcmp
223
224
225
testl %eax, %eax
227
    je .L104
228
    movq 0(\%rbx), \%rbx
229 jmp .L102
230 .L103:
xorq %rbx, %rbx
232 .L104:
movq %rbx, %rax
234 movq 8(%rsp), %rbx
235 movq 16(%rsp), %rbp
```

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

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

C.12 password-O3.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -03 -o compCert/out/password-03.s
   .section .rodata
   .align 1
5 __stringlit_7:
   .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
   .section .rodata
10 .align 1
11 __stringlit_6:
   .ascii "passw0rd123\000"
12
   .type __stringlit_6, @object
13
   .size __stringlit_6, . - __stringlit_6
14
   .section .rodata
15
   .align 1
17 __stringlit_4:
.ascii "!alice12!_veuje@@hak\000"
   .type __stringlit_4, @object
19
   .size __stringlit_4, . - __stringlit_4
20
   .section .rodata
21
   .align 1
22
23 __stringlit_14:
24 .ascii "Password: \000"
   .type __stringlit_14, @object
25
   .size __stringlit_14, . - __stringlit_14
27 .section .rodata
28 .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%1d\012\000"
```

```
31 .type __stringlit_18, @object
   .size __stringlit_18, . - __stringlit_18
   .section .rodata
33
   .align 1
35 __stringlit_13:
   .ascii "User < %s > does not exist.\012\000"
36
   .type __stringlit_13, @object
   .size __stringlit_13, . - __stringlit_13
38
   .section .rodata
40 .align 1
41 __stringlit_8:
   .ascii " %021d. %s\012\000"
   .type __stringlit_8, @object
43
   .size __stringlit_8, . - __stringlit_8
45
   .section .rodata
   .align 1
47 __stringlit_1:
48 .ascii "admin\000"
   .type __stringlit_1, @object
   .size __stringlit_1, . - __stringlit_1
50
   .section .rodata
   .align 1
52
53 __stringlit_2:
.ascii "4dm1n_4eva\\000"
   .type __stringlit_2, @object
55
   .size __stringlit_2, . - __stringlit_2
   .section .rodata
57
58
   .align 1
59 __stringlit_3:
   .ascii "alice\000"
60
   .type __stringlit_3, @object
.size __stringlit_3, . - __stringlit_3
62
   .section .rodata
64
   .align 1
65 __stringlit_11:
   .ascii "Username: \000"
   .type __stringlit_11, @object
67
   .size __stringlit_11, . - __stringlit_11
   .section .rodata
69
   .align 1
70
71 __stringlit_5:
.ascii "abdul\000"
   .type __stringlit_5, @object
   .size __stringlit_5, . - __stringlit_5
74
   .section .rodata
75
76
    .align 1
77 __stringlit_17:
78 .ascii "Welcome, %s!\012\000"
   .type __stringlit_17, @object
79
   .size __stringlit_17, . - __stringlit_17
   .section .rodata
81
   .align 1
82
83 __stringlit_12:
84 .ascii "%255s\000"
.type __stringlit_12, @object
size __stringlit_12, . - __stringlit_12
87 .section .rodata
```

```
ss .align 1
89 __stringlit_9:
    .ascii "\012\000"
    .type __stringlit_9, @object
    .size __stringlit_9, . - __stringlit_9
92
    .section .rodata
93
     .align 1
95 __stringlit_15:
.ascii "ERROR: incorrect password\012\000"
    .type __stringlit_15, @object
97
98
     .size __stringlit_15, . - __stringlit_15
99
     .section .rodata
    .align 1
100
101 __stringlit_10:
.ascii "Welcome to BigBank Australia!\012\000"
    .type __stringlit_10, @object
103
104
    .size __stringlit_10, . - __stringlit_10
    .section .rodata
105
106 .align 1
107 __stringlit_16:
    .ascii "Logged in as < %s >!\012\000"
     .type __stringlit_16, @object
109
    .size __stringlit_16, . - __stringlit_16
110
111
    .text
    .align 16
.globl setup_users
112
113
114 setup_users:
.cfi_startproc
     subq $40, %rsp
116
     .cfi_adjust_cfa_offset 40
117
    leaq 48(%rsp), %rax
movq %rax, 0(%rsp)
118
119
     movq %rbx, 8(%rsp)
120
    movq %rbp, 16(%rsp)
movq %r12, 24(%rsp)
movq $528, %rdi
121
122
123
     call malloc
124
     movq %rax, %rbp
125
     leaq 8(%rbp), %rdi
126
     leaq __stringlit_1(%rip), %rsi
call strcpy
127
128
           strcpy
     leaq 264(%rbp), %rdi
129
130
     leaq __stringlit_2(%rip), %rsi
    call strcpy
movq $1000000, %rax
movq %rax, 520(%rbp)
131
132
133
     movq $528, %rdi
134
135
     call malloc
     movq %rax, %r12
136
     leaq 8(%r12), %rdi
137
138
     leaq __stringlit_3(%rip), %rsi
     call strcpy
139
    leaq 264(%r12), %rdi
140
    leaq __stringlit_4(%rip), %rsi
call strcpy
141
142
     call
           strcpy
143 movq $783, %rsi
movq %rsi, 520(%r12)
```

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

```
addq $24, %rsp
202
    .cfi_endproc
204
    .type print_users, @function
205
    .size print_users, . - print_users
206
207
    .text
    .align 16
208
209 .globl getUser
210 getUser:
211
    .cfi_startproc
    subq $24, %rsp
212
    .cfi_adjust_cfa_offset 24
213
    leaq 32(%rsp), %rax
214
215 movq %rax, 0(%rsp)
    movq %rbx, 8(%rsp)
movq %rbp, 16(%rsp)
movq %rsi, %rbp
216
217
218
movq %rdi, %rbx
220 .L102:
221 cmpq $0, %rbx
222
     je .L103
    leaq 8(%rbx), %rdi
223
224 movq %rbp, %rsi
225
    call strcmp
    testl %eax, %eax
226
227
    je .L104
228 movq 0(%rbx), %rbx
jmp .L102
230 .L103:
xorq %rbx, %rbx
232 .L104:
movq %rbx, %rax
    movq 8(%rsp), %rbx
234
    movq 16(%rsp), %rbp
addq $24, %rsp
235
236
237
    ret
    .cfi_endproc
238
239
    .type getUser, @function
    .size getUser, . - getUser
240
241
    .text
242
    .align 16
243 .globl main
244 main:
.cfi_startproc
     subq $536, %rsp
246
247
    .cfi_adjust_cfa_offset 536
    leaq 544(%rsp), %rax
248
    movq %rax, 0(%rsp)
249
    movq %rbx, 8(%rsp)
250
251
     call
           setup_users
    movq %rax, %rbx
252
    leaq
           __stringlit_10(%rip), %rdi
253
    movl $0, %eax
254
    call printf
255
           __stringlit_11(%rip), %rdi
256
    leaq
257 movl $0, %eax
258 call printf
```

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

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

C.13 deadStoreElimination-O0.s

```
# File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -00 -o compCert/out/deadStoreElimination-00.s
   .text
   .align 16
.globl deadStore
6 deadStore:
   .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
   leaq 16(%rsp), %rax movq %rax, 0(%rsp)
10
11
12 .L100:
cmpl %esi, %edi
   jle .L101
14
   leal -1(%edi), %edi
15
    jmp .L100
17 .L101:
18 leal 0(%edi,%esi,1), %eax
   addq $8, %rsp
19
20
   ret
21
    .cfi_endproc
   .type deadStore, @function
22
   .size deadStore, . - deadStore
   .text
24
   .align 16
25
    .globl main
26
27 main:
   .cfi_startproc
   subq $8, %rsp
29
    .cfi_adjust_cfa_offset 8
30
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
31
32
    movl $2, %esi
   call deadStore
xorl %eax, %eax
34
35
    addq $8, %rsp
36
37
   ret
38 .cfi_endproc
   .type main, @function
39
40 .size main, . - main
```

C.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7
# Command line: deadStoreElimination.c -S -03 -o compCert/out/deadStoreElimination-03.s
.text
align 16
sglobl deadStore
deadStore:
cfi_startproc
subq $8, %rsp
cfi_adjust_cfa_offset 8
leaq 16(%rsp), %rax
```

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

C.15 pread-O0.s

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

```
jne .L100
movl x(%rip), %esi
movl z(%rip), %esi
jmp .L100
cfi_endproc
type main, @function
size main, . - main
```

C.16 pread-O3.s

```
# File generated by CompCert 3.7
2 # Command line: pread.c -S -03 -o compCert/out/pread-03.s
   .comm z, 4, 4
   .comm x, 4, 4
    .text
   .align 16
7 .globl main
8 main:
9 .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
11
12 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
14 .L100:
   movl z(%rip), %edx
15
movq %rdx, %rsi
movq %rsi, %rax
testl %eax, %eax
   leal 1(%eax), %ecx
cmovl %rcx, %rax
19
21 sarl $1, %eax
22 leal 0(,%eax,2), %edi
23 subl %edi, %esi
   testl %esi, %esi
24
    jne .L100
movl x(%rip), %esi
27 movl z(%rip), %esi
28 jmp .L100
   .cfi_endproc
    .type main, Ofunction
31 .size main, . - main
```

D CompCert Annotated Assembly Output

D.1 comment-O0.s

```
# File generated by CompCert 3.7

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

.text

align 16

globl main

main:

cfi_startproc

subq $8, %rsp

cfi_adjust_cfa_offset 8

leaq 16(%rsp), %rax
```

```
movq %rax, 0(%rsp)
12 .L100:
xorl %eax, %eax
addq $8, %rsp
15
   ret
   .cfi_endproc
16
17
   .type main, @function
size main, . - main
   .section "__compcert_ais_annotations","", @note
   .ascii "# file:comment.c line:2 function:main\n"
20
   .byte 7,8
21
   .quad .L100
22
.ascii " Critical Comment\n"
```

D.2 comment-O3.s

```
# File generated by CompCert 3.7
2 # Command line: comment.c -S -03 -o annotated/comment-03.s
    .text
    .align 16
5 .globl main
6 main:
7 .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
12 .L100:
   xorl %eax, %eax
addq $8, %rsp
13
14
15
   ret
   .cfi_endproc
   .type main, @function
17
   .size main, . - main
.section "__compcert_ais_annotations","",@note
18
19
   .ascii "# file:comment.c line:2 function:main\n"
20
   .byte 7,8
   .quad .L100
22
.ascii " Critical Comment\n"
```

D.3 variable-O0.s

```
# File generated by CompCert 3.7

# Command line: variable.c -S -00

.text

align 16

globl main

main:

cfi_startproc

subq $8, %rsp

cfi_adjust_cfa_offset 8

leaq 16(%rsp), %rax

movq %rax, 0(%rsp)

movl $-10, %esi

13.L100:

movl $98, %ecx

L101:
```

```
16 negl %ecx
17 .L102:
movl $1, %r10d
19 .L103:
20 movabsq $4294967296, %rax
21 .L104:
22
   movsd .L105(%rip), %xmm1 # 3.14159265358979312
   cvtsd2ss %xmm1, %xmm3
23
    cvtss2sd %xmm3, %xmm0
    movsd .L106(%rip), %xmm2 # 2.3784000000000007
25
    divsd %xmm2, %xmm0
26
27
    cqto
    shrq $59, %rdx
28
    leaq 0(%rax, %rdx, 1), %rax
29
    sarq $5, %rax
30
    leal
         0(%eax,%esi,1), %r9d
31
    leal 0(%r9d,%ecx,1), %r8d
32
    leal 0(%r8d,%r10d,1), %r10d
33
    cvttss2si %xmm3, %edx
    leal 0(%r10d, %edx, 1), %r11d
35
    cvttsd2si %xmm0, %r8d
36
    leal 0(%r11d,%r8d,1), %ecx
37
    leal 0(%ecx, %edi, 1), %eax
38
39
    addq $8, %rsp
40
    ret
41
    .cfi_endproc
    .type main, Ofunction
42
    .size main, . - main
43
    .section .rodata.cst8, "aM", @progbits,8
44
    .align 8
45
46 .L105: .quad 0x400921fb54442d18
47 .L106: .quad 0x400306f694467382
   .section "__compcert_ais_annotations","", Onote
    .ascii "# file:variable.c line:3 function:main\n"
49
    .byte 7,8
50
51
    .quad .L100
    .ascii " reg(\"rsi\") = int\n"
52
    .ascii "# file:variable.c line:5 function:main\n"
53
    .byte 7,8
54
55
    .quad .L101
    .ascii " reg(\"rcx\") = char\n"
56
57
    .ascii "# file:variable.c line:7 function:main\n"
58
    .byte 7,8
    .quad .L102
59
    .ascii " reg(\"rcx\") = unsigned int\n"
60
    .ascii "# file:variable.c line:9 function:main\n"
61
    .byte 7,8
62
63
    .quad .L103
    .ascii " reg(\"r10\") = short\n"
64
    .ascii "# file:variable.c line:11 function:main\n"
    .byte 7,8
66
    .quad .L104
67
.ascii " reg(\rdot x) = long\rdot x")
```

D.4 variable-O3.s

```
# File generated by CompCert 3.7
```

```
2 # Command line: variable.c -S -03
   .align 16
5 .globl main
6 main:
   .cfi_startproc
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
12 .L100:
13 .L101:
14 .L102:
15 .L103:
16 .L104:
   leal
         134217625(%edi), %eax
17
   addq $8, %rsp
18
   ret
19
20
   .cfi_endproc
   .type main, @function
21
   .size main, . - main
   .section "__compcert_ais_annotations","", @note
23
   .ascii "# file:variable.c line:3 function:main\n"
24
25
   .byte 7,8
   .quad .L100
26
    .ascii " -10 = int n"
27
   .ascii "# file:variable.c line:5 function:main\n"
28
   .byte 7,8
29
   .quad .L101
30
   .ascii " 98 = char\n"
31
   .ascii "# file:variable.c line:7 function:main\n"
32
   .byte 7,8
33
   .quad .L102
34
   .ascii " -98 = unsigned int\n"
35
   .ascii "# file:variable.c line:9 function:main\n"
36
37
   .byte 7,8
   .quad .L103
38
   .ascii " 1 = short n"
   .ascii "# file:variable.c line:11 function:main\n"
40
41
   .byte 7,8
   .quad .L104
42
.ascii " 4294967296 = long \n"
```

D.5 volatile-O0.s

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

D.6 volatile-O3.s

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

D.7 loop-O0.s

```
# File generated by CompCert 3.7
 2 # Command line: loop.c -S -00 -o annotated/loop-00.s
    .comm z, 4, 4
    .comm x, 4, 4
    .text
 6 .align 16
    .globl main
 8 main:
 9 .cfi_startproc
10 subq $8, %rsp
    .cfi_adjust_cfa_offset 8
11
leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
14 .L100:
15 .L101:
16 xorl
          %edx, %edx
xorl %edi, %edi
18 .L102:
19 .L103:
20 .L104:
21 nop
22 .L105:
23 .L106:
24 .L107:
25 nop
26 .L108:
27 .L109:
28 .L110:
movl z(\%rip), \%edx
    movq %rdx, %rax
testl %eax, %eax
30
31
    leal 1(%eax), %ecx
32
    cmovl %rcx, %rax
33
    sarl $1, %eax
leal 0(,%eax,2), %edi
movq %rdx, %rcx
subl %edi, %ecx
34
35
36
37
    testl %ecx, %ecx
    jne .L108
39
    movl x(%rip), %edi
movl z(%rip), %esi
40
41
42
    jmp .L105
    .cfi_endproc
    .type main, @function
44
    .size main, . - main
.section "__compcert_ais_annotations","",@note
45
46
    .ascii "# file:loop.c line:7 function:main\n"
47
    .byte 7,8
48
    .quad .L100
49
     .ascii " L(mem("
50
    .byte 7,8
51
    .quad z
52
    .ascii ", 4)) = true\n"
53
    .ascii "# file:loop.c line:8 function:main\n"
54
55
    .byte 7,8
    .quad .L101
56
.ascii " L(mem("
```

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

D.8 loop-O3.s

```
# File generated by CompCert 3.7

# Command line: loop.c -S -O3 -o annotated/loop-O3.s

.comm z, 4, 4

.comm x, 4, 4

.text

.align 16

.globl main

main:
```

```
9 .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
11
12 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
14 .L100:
15 .L101:
16 xorl %edx, %edx
17 xorl %edi, %edi
18 .L102:
19 .L103:
20 .L104:
21 nop
22 .L105:
23 .L106:
24 .L107:
25 nop
26 .L108:
27 .L109:
28 .L110:
29
   movl z(%rip), %edx
   movq %rdx, %rax
30
    testl %eax, %eax
31
32
    leal 1(%eax), %ecx
    cmovl %rcx, %rax
sarl $1, %eax
33
34
    leal 0(,%eax,2), %edi
35
    movq %rdx, %rcx
subl %edi, %ecx
36
37
    testl %ecx, %ecx
38
39
    jne .L108
    movl x(%rip), %edi
40
    movq %rdx, %rsi
41
    jmp .L105
42
    .cfi_endproc
43
44
    .type main, Ofunction
    .size main, . - main
45
    .section "__compcert_ais_annotations","", @note
    .ascii "# file:loop.c line:7 function:main\n"
47
48
    .byte 7,8
    .quad .L100
49
50
    .ascii " L(mem("
    .byte 7,8
51
    .quad z
52
    .ascii ", 4)) = true\n"
53
    .ascii "# file:loop.c line:8 function:main\n"
54
    .byte 7,8
55
56
    .quad .L101
    .ascii " L(mem("
57
58
    .byte 7,8
    .quad x
59
    .ascii ", 4))= mem("
60
61
    .byte 7,8
    .quad z
62
    .ascii ", 4) % 2 == 0\n"
    .ascii "# file:loop.c line:11 function:main\n"
64
65 .byte 7,8
```

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

D.9 rooster-O0.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -S -00 -o annotated/rooster-00.s
   .comm rooster, 4, 4
   .comm drake, 4, 4
   .comm goose, 4, 4
   .data
   .align 4
8 count:
   .long 0
10
   .type count, @object
   .size count, . - count
11
12
   .text
   .align 16
13
14 .globl fun
15 fun:
.cfi_startproc
```

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

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

D.10 rooster-O3.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -S -03 -o annotated/rooster-03.s
   .comm rooster, 4, 4
   .comm drake, 4, 4
   .comm goose, 4, 4
    .data
   .align 4
8 count:
   .long 0
   .type count, @object
10
11
    .size count, . - count
   .text
12
13 .align 16
14 .globl fun
15 fun:
16
   .cfi_startproc
17 subq $8, %rsp
   .cfi_adjust_cfa_offset 8
19 leaq 16(%rsp), %rax
20 movq %rax, 0(%rsp)
21 .L100:
leal 0(%edi,%esi,1), %r9d
leal 0(%r9d,%edx,1), %eax
testl %eax, %eax
25 jl .L101
```

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

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

D.11 password-O0.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -00 -o annotated/password-00.s
   .section .rodata
    .align 1
5 __stringlit_7:
6 .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
   .section .rodata
10 .align 1
11 __stringlit_6:
.ascii "passw0rd123\000"
   .type __stringlit_6, @object
.size __stringlit_6, . - __stringlit_6
14
15 .section .rodata
16 .align 1
17 __stringlit_4:
    .ascii "!alice12!_veuje@@hak\000"
   .type __stringlit_4, @object
19
   .size __stringlit_4, . - __stringlit_4
20
21
   .section .rodata
   .align 1
22
23 __stringlit_14:
   .ascii "Password: \000"
24
   .type __stringlit_14, @object
   .size __stringlit_14, . - __stringlit_14
26
   .section .rodata
27
   .align 1
29 __stringlit_18:
30 .ascii "Your balance: $%1d\012\000"
   .type __stringlit_18, @object
32
   .size __stringlit_18, . - __stringlit_18
33
   .section .rodata
   .align 1
34
35 __stringlit_13:
.ascii "User < %s > does not exist.\012\000"
.type __stringlit_13, @object
```

```
.size __stringlit_13, . - __stringlit_13
   .section .rodata
   .align 1
40
41 __stringlit_8:
.ascii " %021d. %s\012\000"
   .type __stringlit_8, @object
43
   .size __stringlit_8, . - __stringlit_8
   .section .rodata
45
46 .align 1
47 __stringlit_1:
   .ascii "admin\000"
48
   .type __stringlit_1, @object
49
   .size __stringlit_1, . - __stringlit_1
50
   .section .rodata
52 .align 1
53 __stringlit_2:
   .ascii "4dm1n__4eva\000"
54
   .type __stringlit_2, @object
55
.size __stringlit_2, . - __stringlit_2
57
   .section .rodata
   .align 1
59 __stringlit_3:
60 .ascii "alice\000"
   .type __stringlit_3, @object
   .size __stringlit_3, . - __stringlit_3
62
   .section .rodata
   .align 1
64
65 __stringlit_11:
66 .ascii "Username: \000"
   .type __stringlit_11, @object
67
   .size __stringlit_11, . - __stringlit_11
69
   .section .rodata
   .align 1
71 __stringlit_5:
   .ascii "abdul\000"
72
   .type __stringlit_5, @object
73
   .size __stringlit_5, . - __stringlit_5
74
   .section .rodata
   .align 1
76
77 __stringlit_17:
   .ascii "Welcome, %s!\012\000"
   .type __stringlit_17, @object
79
   .size __stringlit_17, . - __stringlit_17
81
   .section .rodata
   .align 1
83 __stringlit_12:
84 .ascii "%255s\000"
   .type __stringlit_12, @object
   .size __stringlit_12, . - __stringlit_12
86
   .section .rodata
   .align 1
88
89 __stringlit_9:
90 .ascii "\012\000"
   .type __stringlit_9, @object
91
   .size __stringlit_9, . - __stringlit_9
   .section .rodata
94 .align 1
```

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

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

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

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

```
.quad .L100
323
    .ascii " L((reg(\"rbp\") + 264)) = high\n"
    .ascii "# file:password.c line:26 function:setup_users\n"
325
    .byte 7,8
326
    .quad .L101
327
    .ascii " L((reg(\"r12\") + 264)) = high\n"
328
    .ascii "# file:password.c line:32 function:setup_users\n"
329
    .byte 7,8
330
    .quad .L102
.ascii " L((reg(\rc v) + 264)) = high\n"
```

D.12 password-O3.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -S -03 -o annotated/password-03.s
   .section .rodata
   .align 1
5 __stringlit_7:
   .ascii "--- USERS ---\012\000"
   .type __stringlit_7, @object
   .size __stringlit_7, . - __stringlit_7
   .section .rodata
   .align 1
10
11 __stringlit_6:
.ascii "passw0rd123\000"
   .type __stringlit_6, @object
14
   .size __stringlit_6, . - __stringlit_6
   .section .rodata
15
16
   .align 1
17 __stringlit_4:
.ascii "!alice12!_veuje@@hak\000"
   .type __stringlit_4, @object
19
   .size __stringlit_4, . - __stringlit_4
   .section .rodata
21
   .align 1
22
23 __stringlit_14:
   .ascii "Password: \000"
24
   .type __stringlit_14, @object
   .size __stringlit_14, . - __stringlit_14
26
   .section .rodata
   .align 1
28
29 __stringlit_18:
   .ascii "Your balance: $%ld\012\000"
.type __stringlit_18, @object
31
.size __stringlit_18, . - __stringlit_18
   .section .rodata
33
   .align 1
34
35 __stringlit_13:
   .ascii "User < %s > does not exist.\012\000"
36
   .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"
.type __stringlit_8, @object
.size __stringlit_8, . - __stringlit_8
```

```
45 .section .rodata
   .align 1
47 __stringlit_1:
48 .ascii "admin\000"
.type __stringlit_1, @object
    .size __stringlit_1, . - __stringlit_1
50
51
    .section .rodata
   .align 1
52
53 __stringlit_2:
.ascii "4dm1n_4eva\\000"
    .type __stringlit_2, @object
.size __stringlit_2, . - __stringlit_2
55
56
    .section .rodata
57
   .align 1
59 __stringlit_3:
    .ascii "alice\000"
    .type __stringlit_3, @object
61
.size __stringlit_3, . - __stringlit_3
63 .section .rodata
64 .align 1
65 __stringlit_11:
    .ascii "Username: \000"
66
   .type __stringlit_11, @object
67
    .size __stringlit_11, . - __stringlit_11
    .section .rodata
69
70
    .align 1
71 __stringlit_5:
_{72} .ascii "abdul\000"
    .type __stringlit_5, @object
73
74
    .size __stringlit_5, . - __stringlit_5
75
    .section .rodata
    .align 1
76
77 __stringlit_17:
  .ascii "Welcome, %s!\012\000"
78
    .type __stringlit_17, @object
79
    .size __stringlit_17, . - __stringlit_17
80
    .section .rodata
81
82 .align 1
83 __stringlit_12:
84
    .ascii "%255s\000"
    .type __stringlit_12, @object
86
    .size __stringlit_12, . - __stringlit_12
    .section .rodata
88
   .align 1
89 __stringlit_9:
    .ascii "\012\000"
90
   .type __stringlit_9, @object
91
.size __stringlit_9, . - __stringlit_9
   .section .rodata
93
    .align 1
95 __stringlit_15:
96 .ascii "ERROR: incorrect password\012\000"
    .type __stringlit_15, @object
    .size __stringlit_15, . - __stringlit_15
98
    .section .rodata
   .align 1
100
101 __stringlit_10:
```

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

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

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

```
leaq 16(%rsp), %rsi
movl $0, %eax
273
274
     call printf
275
     xorl %eax, %eax
276
    jmp .L109
277
278 .L108:
279
    leaq
            __stringlit_14(%rip), %rdi
     movl $0, %eax
280
     call printf
281
     leaq __stringlit_12(%rip), %rdi
leag 272(%rep) %rei
282
     leaq 272(%rsp
movl $0, %eax
           272(%rsp), %rsi
283
284
     call __isoc99_scanf
285
     leaq 264(%rbx), %rdi
286
     leaq 272(%rsp), %rsi
287
     call
           strcmp
288
     testl %eax, %eax
289
     je .L110
290
     leaq __stringlit_15(%rip), %rdi
movl $0, %eax
291
292
     call printf
xorl %eax, %eax
294
     jmp .L109
295
296 .L110:
    leaq __stringlit_16(%rip), %rdi
leaq 8(%rbx), %rsi
297
298
     movl $0, %eax
299
     call printf
300
     leaq __stringlit_9(%rip), %rdi
movl $0, %eax
301
302
303
     call
            printf
            __stringlit_17(%rip), %rdi
     leaq
304
     leaq 8(%rbx), %rsi
305
306
     movl $0, %eax
     call printf
307
308
     leaq
            __stringlit_18(%rip), %rdi
     movq 520(%rbx), %rsi
309
310
     movl $0, %eax
    call printf
xorl %eax, %eax
311
312
313 .L109:
314
    movq 8(%rsp), %rbx
315
     addq $536, %rsp
     ret
316
     .cfi_endproc
317
318
     .type main, Ofunction
     .size main, . - main
319
     .section "__compcert_ais_annotations","", @note
320
     .ascii "# file:password.c line:20 function:setup_users\n"
321
322
     .byte 7,8
     .quad .L100
323
     .ascii " L((reg(\rphi") + 264)) = high\n"
324
     .ascii "# file:password.c line:26 function:setup_users\n"
325
     .byte 7,8
326
327
     .quad .L101
     .ascii " L((reg(\"r12\") + 264)) = high\n"
328
.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

```
# File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -00 -o annotated/deadStoreElimination-00.s
   .align 16
   .globl deadStore
6 deadStore:
   .cfi_startproc
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
12
   movl $43981, %ecx
13 .L100:
14 nop
15 .L101:
cmpl %esi, %edi
    jle .L102
17
18 leal -1(%edi), %edi
19 jmp .L101
20 .L102:
   leal 0(%edi,%esi,1), %eax
21
    addq $8, %rsp
   ret
23
   .cfi_endproc
24
   .type deadStore, @function
   .size deadStore, . - deadStore
26
    .text
   .align 16
28
   .globl main
29
30 main:
   .cfi_startproc
31
32
    subq $8, %rsp
   .cfi_adjust_cfa_offset 8
33
   leaq 16(%rsp), %rax
   movq %rax, 0(%rsp)
movl $2, %esi
call deadStore
35
36
37
   xorl %eax, %eax
38
    addq $8, %rsp
   ret
40
41
   .cfi_endproc
    .type main, Ofunction
42
   .size main, . - main
43
   .section "__compcert_ais_annotations","", @note
   .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
45
   .byte 7,8
   .quad .L100
47
.ascii " L(reg(\rcx\")) = high\n"
```

D.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -S -03 -o annotated/deadStoreElimination-03.s
    .text
   .align 16
   .globl deadStore
6 deadStore:
    .cfi_startproc
   subq $8, %rsp
   .cfi_adjust_cfa_offset 8
10 leaq 16(%rsp), %rax
11 movq %rax, 0(%rsp)
12 .L100:
13 nop
14 .L101:
cmpl %esi, %edi
    jle .L102
16
17 leal -1(%edi), %edi
   jmp .L101
18
19 .L102:
   leal 0(%edi,%esi,1), %eax
20
21
    addq $8, %rsp
22
    ret
   .cfi_endproc
23
24
   .type deadStore, @function
    .size deadStore, . - deadStore
25
    .text
26
    .align 16
27
   .globl main
28
29 main:
   .cfi_startproc
30
    subq $8, %rsp
31
    .cfi_adjust_cfa_offset 8
32
   leaq 16(%rsp), %rax
33
   movq %rax, 0(%rsp)
movl $2, %esi
call deadStore
34
35
36
    xorl %eax, %eax
37
    addq $8, %rsp
   ret
39
40
    .cfi_endproc
    .type main, @function
41
   .size main, . - main
42
   .section "__compcert_ais_annotations","", @note
    .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
44
    .byte 7,8
45
46
    .quad .L100
.ascii " L(43981) = high n"
```

D.15 pread-O0.s

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

```
ais annotations
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

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

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

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

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

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

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

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

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

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

E Inline Assembly Annotated C Programs

E.1 comment.c

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

E.2 variable.c

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

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

E.3 volatile.c

```
volatile int x;

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

E.4 loop.c

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

E.5 rooster.c

```
int rooster;
int drake;
// MEDIUM
int goose;
int fun(int a, int b, int c) {
```

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

E.6 password.c

```
#include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
5 #define BUFF_LEN 256
7 typedef struct user_t user_t;
9 struct user_t {
10
   user_t* next;
     char name[BUFF_LEN];
11
12
     // L(password) = High
     char password[BUFF_LEN];
13
     size_t balance;
14
15 };
16
17 user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
18
19
     strcpy(user_admin->name, "admin");
20
      strcpy(user_admin->password, "4dm1n__4eva");
     user_admin->balance = 1000000;
21
22
     user_t* user_alice = malloc(sizeof(user_t));
23
   strcpy(user_alice->name, "alice");
     strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
user_alice->balance = 783;
```

```
27
      user_t* user_abdul = malloc(sizeof(user_t));
      strcpy(user_abdul->name, "abdul");
29
      strcpy(user_abdul ->password, "passw0rd123");
30
      user_abdul->balance = 2;
31
32
33
      user_admin->next = user_alice;
      user_alice->next = user_abdul;
34
35
      user_abdul ->next = NULL;
36
      return user_admin;
37
38 }
39
40 void print_users(user_t* users) {
      printf("--- USERS ---\n");
41
      size_t count = 0;
42
43
      while (users != NULL) {
          printf(" %02ld. %s\n", ++count, users->name);
44
45
          users = users->next;
46
47
      printf("\n");
48 }
49
50 user_t* getUser(user_t* user_list, char* name) {
      while (user_list != NULL) {
51
          if (strcmp(user_list->name, name) == 0) {
52
               return user_list;
53
54
          user_list = user_list->next;
55
56
57
      return NULL;
58 }
59
60 int main() {
      user_t* users = setup_users();
61
62
      printf("Welcome to BigBank Australia!\n");
63
      char username[BUFF_LEN];
65
66
      printf("Username: ");
      scanf("%255s", username);
67
68
69
      user_t* user = getUser(users, username);
      if (user == NULL) {
70
          printf("User < %s > does not exist.\n", username);
71
72
          return 0;
73
74
      char password[BUFF_LEN];
75
76
      printf("Password: ");
      scanf("%255s", password);
77
      if (strcmp(user->password, password) != 0) {
78
          printf("ERROR: incorrect password \n");\\
79
          return 0;
80
81
82
      printf("Logged in as < %s >!\n", user->name);
```

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

E.7 deadStoreElimination.c

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

E.8 pread.c

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

F Inline Assembly Annotated Assembly Output

F.1 comment-O0.s

```
1 .text
   .file "comment.c"
   .globl main
                                    # -- Begin function main
.p2align 4, 0x90
.type main, @function
                                          # @main
6 main:
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register %rbp
movl $0, -4(%rbp)
   #APP
15
   # CRITICAL COMMENT
16
17
   #NO_APP
18 xorl %eax, %eax
popq %rbp
.cfi_def_cfa %rsp, 8
retq
_{22} .Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
                                          # -- End function
   .ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","",@progbits
28 .addrsig
```

F.2 comment-O3.s

```
1 .text
   .file "comment.c"
  .globl main
                                  # -- Begin function main
p2align 4, 0x90 type main,@function
                                        # @main
7 .cfi_startproc
8 # %bb.0:
  #APP
10 # CRITICAL COMMENT
#NO_APP
xorl %eax, %eax
13
   retq
.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc
                                        # -- End function
17
.ident "clang version 10.0.0-4ubuntu1"
```

```
.section ".note.GNU-stack","",@progbits
.addrsig
```

F.3 variable-O0.s

```
1 .text
2 .file "variable.c"
section .rodata.cst8, "aM", @progbits,8
                 # -- Begin function main
   .p2align 3
5 .LCPI0_0:
.quad 4612538099476886402
                                # double 2.3784000000000001
.section .rodata.cst4,"aM",@progbits,4
   .p2align 2
9 .LCPI0_1:
.long 1078530011
                                # float 3.14159274
11 .text
12 .globl main
.p2align 4, 0x90
.type main,@function
                                          # @main
15 main:
.cfi_startproc
17 # %bb.0:
pushq %rbp
19
    .cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
22
   .cfi_def_cfa_register %rbp
   movl $0, -4(%rbp)
movl %edi, -8(%rbp)
23
24
   movq %rsi, -16(%rbp)
movl $-10, -20(%rbp)
25
   movl -20(%rbp), %eax
27
    #APP
   # annotation: %eax = int
29
   #NO_APP
30
   movb $98, -21(%rbp)
31
   movb
         -21(%rbp), %cl
32
33
    #APP
    # annotation: %cl = char
34
35
    #NO_APP
   xorl %eax, %eax
36
    movsbl -21(%rbp), %edx
37
    subl %edx, %eax
    mov1 %eax, -28(%rbp)
39
    movl -28(\%rbp), \%eax
40
    #APP
41
    # annotation: %eax = unsigned int
42
43
    #NO_APP
    movw $1, -30(\%rbp)
44
    movw -30(%rbp), %r8w
46
    #APP
    # annotation: %r8w = short
47
48
   #NO_APP
movabsq $4294967296, %rsi
                                    # imm = 0x100000000
50 movq %rsi, -40(%rbp)
51 movq -40(%rbp), %rsi
52 #APP
```

```
# annotation: %rsi = long
    #NO_APP
    movss .LCPIO_1(%rip), %xmm0
                                  # xmm0 = mem[0],zero,zero,zero
55
    movss %xmm0, -44(%rbp)
   movss -44(%rbp), %xmm0
                                   # xmm0 = mem[0],zero,zero
57
   #APP
58
59
   # annotation: %xmm0 = float
60 #NO_APP
   movsd .LCPI0_0(%rip), %xmm0
                                  # xmm0 = mem[0],zero
   movss -44(%rbp), %xmm1
cvtss2sd %xmm1, %xmm1
                                  # xmm1 = mem[0],zero,zero,zero
62
63
   divsd %xmm0, %xmm1
64
   movsd %xmm1, -56(%rbp)
65
   movsd -56(%rbp), %xmm0
                                  # xmm0 = mem[0], zero
   #APP
67
    # annotation: %xmm0 = double
68
69
   #NO_APP
   movq -40(\%rbp), \%rax
70
71
    cqto
   movl $32, %esi
72
73
    idivq %rsi
                                           # kill: def $eax killed $eax killed $rax
74
   addl -20(%rbp), %eax
75
    addl -28(%rbp), %eax
76
   movswl -30(%rbp), %edi
77
    addl %edi, %eax
78
   cvttss2si -44(%rbp), %edi
79
   addl %edi, %eax
80
   cvttsd2si -56(%rbp), %edi
81
   addl %edi, %eax
addl -8(%rbp), %eax
82
   popq %rbp
84
   .cfi_def_cfa %rsp, 8
86 retq
87 .Lfunc_end0:
   .size main, .Lfunc_end0-main
   .cfi_endproc
89
                                           # -- End function
   .ident "clang version 10.0.0-4ubuntu1"
91
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:
1 .quad 4608627556095693531 # double 1.3208849398808329
1 .text
1 .globl main
1 .p2align 4, 0x90
```

```
.type main, @function
15 main:
                                         # @main
.cfi_startproc
17 # %bb.0:
                                         # kill: def $edi killed $edi def $rdi
18
   #APP
19
   # annotation: \$-10 = int
20
21 #NO_APP
   #APP
   # annotation: $98 = char
23
   #NO_APP
24
   #APP
25
   # annotation: $-98 = unsigned int
26
27
   #NO_APP
   #APP
28
   # annotation: $1 = short
29
30
   #NO_APP
   #APP
31
32
   # annotation: $4294967296 = long
   #NO_APP
33
   movss .LCPIO_0(%rip), %xmm0 # xmm0 = mem[0],zero,zero,zero
   #APP
35
   # annotation: %xmm0 = float
36
37
   #NO_APP
   movsd .LCPIO_1(%rip), %xmm0
                                # xmm0 = mem[0],zero
38
   #APP
39
   # annotation: %xmm0 = double
40
41 #NO_APP
42 leal 134217625(%rdi), %eax
   retq
43
44 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
                                         # -- End function
47
   .ident "clang version 10.0.0-4ubuntu1 "
48
   .section ".note.GNU-stack","",@progbits
50 .addrsig
```

F.5 volatile-O0.s

```
1 .text
2 .file "volatile.c"
3 .globl main
4 .p2align 4, 0x90
5 .type main, @function
                                        # -- Begin function main
                                              # @main
6 main:
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
13
    .cfi_def_cfa_register %rbp
movl $0, -4(\%rbp)
movl x, %eax
#APP # annotation: %eax = High
```

```
18 #NO_APP
movl x, %eax addl $1, %eax
21 popq %rbp
.cfi_def_cfa %rsp, 8
23
   retq
24 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
                                      # -- End function
27
   .type x,@object
                                # @x
28
   .comm x,4,4
29
.ident "clang version 10.0.0-4ubuntu1"
.section ".note.GNU-stack","", @progbits
32 .addrsig
33 .addrsig_sym x
```

F.6 volatile-O3.s

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

F.7 loop-O0.s

```
.cfi_offset %rbp, -16
12
    movq %rsp, %rbp
    .cfi_def_cfa_register %rbp
13
14 movl $0, -4(%rbp)
15 movl $0, -8(%rbp)
16 movl $0, -12(%rbp)
17 .LBB0_1:
                                             # =>This Loop Header: Depth=1
                                                Child Loop BBO_2 Depth 2
18
                                                     Child Loop BBO_3 Depth 3
20 jmp .LBB0_2
21 .LBB0_2:
                                               Parent Loop BBO_1 Depth=1
                                             # => This Loop Header: Depth=2
22
                                                     Child Loop BBO_3 Depth 3
23
24 jmp .LBB0_3
25 .LBB0_3:
                                                Parent Loop BBO_1 Depth=1
                                             #
                                                  Parent Loop BBO_2 Depth=2
27
                                                    This Inner Loop Header: Depth=3
movl z, %eax
movl \%eax, -8(\%rbp)
30 # %bb.4:
                                             # in Loop: Header=BB0_3 Depth=3
movl -8(%rbp), %eax
32
    cltd
33 movl $2, %ecx
34 idivl %ecx
^{35} cmpl $0, \%edx
    jne .LBB0_3
36
37 # %bb.5:
                                             # in Loop: Header=BBO_2 Depth=2
38 movl x, %eax
39 movl %eax, -12(%rbp)
40 # %bb.6:
                                                in Loop: Header=BBO_2 Depth=2
movl z, %eax
cmpl -8(%rbp), %eax
jne .LBB0_2
44 # %bb.7:
                                                in Loop: Header=BBO_1 Depth=1
45 jmp .LBB0_1
46 .Lfunc_end0:
.size main, .Lfunc_end0-main
    .cfi_endproc
                                            # -- End function
49
    .type z,@object
50
                                    # @z
51
    .comm z, 4, 4
    .type x,@object
52
    .comm x,4,4
    .ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","", @progbits
54
56
    .addrsig
.addrsig_sym z
.addrsig_sym x
```

F.8 loop-O3.s

```
7 .cfi_startproc
8 # %bb.0:
9 testb $1, z(%rip)
jne .LBB0_2
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
.Tfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
19
                                          # -- End function
                                 # @z
   .type z,@object
21
   .comm z,4,4
23
   .type x,@object
                                  # @x
.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
    .cfi_def_cfa_offset 16
10
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register %rbp
movl %edi, -8(%rbp)
movl %esi, -12(%rbp)
movl %edx, -16(%rbp)
movl -8(%rbp), %eax
addl -12(%rbp), %eax
addl -16(%rbp), %eax
20 movl %eax, -20(%rbp)
21 cmpl $0, -20(%rbp)
jge .LBB0_2
23 # %bb.1:
movl -20(%rbp), %eax
25 movl %eax, -4(%rbp)
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
                                                 Child Loop BBO_7 Depth 2
38
39 movl -8(%rbp), %eax
40 cmpl -12(%rbp), %eax
je .LBB0_10
42 # %bb.6:
                                                in Loop: Header=BBO_5 Depth=1
movl -8(%rbp), %eax
44 addl $1, %eax
45 movl %eax, -8(%rbp)
46 movl fun.count, %eax
47 addl $1, %eax
movl %eax, fun.count
49 .LBB0_7:
                                               Parent Loop BBO_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)
movl fun.count, %eax addl $1, %eax
60 movl %eax, fun.count
61 jmp .LBB0_7
                                                in Loop: Header=BBO_5 Depth=1
62 .LBB0_9:
63 jmp .LBB0_5
64 .LBB0_10:
65 jmp .LBB0_11
66 .LBB0_11:
movl fun.count, %eax movl %eax, -4(%rbp)
69 .LBB0_12:
movl -4(%rbp), %eax
71 popq %rbp
.cfi_def_cfa %rsp, 8
73
    retq
74 .Lfunc_end0:
.size fun, .Lfunc_end0-fun
.cfi_endproc
                                            # -- End function
    .globl main
                                     # -- Begin function main
   .p2align 4, 0x90
79
    .type main, @function
80
                                            # @main
81 main:
82 .cfi_startproc
83 # %bb.0:
84 pushq %rbp
    .cfi_def_cfa_offset 16
85
    .cfi_offset %rbp, -16
movq %rsp, %rbp
    .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
    movl rooster, %edi
movl drake, %esi
movl goose, %edx
callq fun
95
96
97
98 xorl %ecx, %ecx
99 movl %eax, -8(%rbp)
    movl %ecx, %eax addq $16, %rsp popq %rbp
100
101
102
    .cfi_def_cfa %rsp, 8
103
104 retq
105 .Lfunc_end1:
    .size main, .Lfunc_end1-main
106
107
     .cfi_endproc
                                               # -- End function
108
.type fun.count,@object # @fun.count
     .local fun.count
110
111
     .comm fun.count,4,4
    .type rooster,@object
                                     # @rooster
112
    .comm rooster,4,4
113
    .type drake,@object
                                      # @drake
114
    .comm drake,4,4
115
116
     .type goose,@object
                                      # @goose
    .comm goose,4,4
117
    .ident "clang version 10.0.0-4ubuntu1"
118
     .section ".note.GNU-stack","", @progbits
119
120
     .addrsig
121
     .addrsig_sym fun
    .addrsig_sym fun.count
122
    .addrsig_sym rooster
123
.addrsig_sym drake
.addrsig_sym goose
```

F.10 rooster-O3.s

```
1 .text
 2 .file "rooster.c"
 3 .globl fun
                                     # -- Begin function fun
 4 .p2align 4, 0x90
    .type fun, @function
                                           # @fun
6 fun:
 7 .cfi_startproc
 8 # %bb.0:
                                           # kill: def $edx killed $edx def $rdx
                                           # kill: def $esi killed $esi def $rsi
10
                                           # kill: def $edi killed $edi def $rdi
11
12 leal (%rsi,%rdi), %eax
addl %edx, %eax
    js .LBB0_9
14
15 # %bb.1:
movl fun.count(%rip), %eax
cmpl %esi, %edi
18 jge .LBBO_9
19 # %bb.2:
```

```
cmpl %edx, %esi
jge .LBB0_9
22 # %bb.3:
leal 1(%rdi), %ecx
cmpl %esi, %ecx
25
    jne .LBB0_5
26 # %bb.4:
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
jne .LBB0_6
38 # %bb.7:
39 addl %edx, %eax
40 subl %esi, %eax
41 .LBB0_8:
movl %eax, fun.count(%rip)
43 .LBB0_9:
                                            # kill: def $eax killed $eax killed $rax
44
45
    retq
46 .Lfunc_end0:
.size fun, .Lfunc_end0-fun
    .cfi_endproc
48
                                            # -- End function
49
    .globl main
                                     # -- Begin function main
50
   .p2align 4, 0x90
51
.type main, @function
                                           # @main
53 main:
.cfi_startproc
55 # %bb.0:
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:
.size main, .Lfunc_end1-main
    .cfi_endproc
64
                                           # -- End function
65
   .type fun.count,@object
                                 # @fun.count
66
    .local fun.count
67
    .comm fun.count,4,4
68
69
    .type rooster,@object
                                   # @rooster
    .comm rooster,4,4
70
    .type drake, @object
71
                                   # @drake
72
    .comm drake,4,4
    .type goose,@object
                                   # @goose
73
74
    .comm goose,4,4
    .ident "clang version 10.0.0-4ubuntu1 "
75
.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
.p2align 4, 0x90.type setup_users,@function
                                                 # @setup_users
 6 setup_users:
7 .cfi_startproc
 8 # %bb.0:
9 pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
11
movq %rsp, %rbp
13
    .cfi_def_cfa_register %rbp
    subq $80, %rsp
mov1 $528, %edi
14
15
                                       # imm = 0x210
    callq malloc
16
    movq %rax, -8(%rbp)
17
    movq -8(%rbp), %rax
18
    addq $8, %rax
movl $.L.str, %esi
19
20
    movq %rax, %rdi
21
    callq strcpy
    movq -8(%rbp), %rcx
addq $264, %rcx
movl $.L.str.1, %esi
23
                                       # imm = 0x108
24
25
    movq %rcx, %rdi
26
    movq %rax, -32(%rbp)
                                     # 8-byte Spill
28
    callq strcpy
    movq -8(%rbp), %rcx
movq $1000000, 520(%rcx)
mov1 $528, %edi
                                     # imm = 0xF4240
30
                                       # imm = 0x210
31
32
    movq %rax, -40(%rbp)
                                      # 8-byte Spill
    callq malloc
33
    movq %rax, -16(%rbp)
    movq -16(%rbp), %rax
35
    addq $8, %rax
36
    movl $.L.str.2, %esi
movq %rax, %rdi
37
38
    callq strcpy
39
40
    movq -16(%rbp), %rcx
    addq $264, %rcx
movl $.L.str.3, %esi
                                       # imm = 0x108
41
42
    movq %rcx, %rdi
movq %rax, -48(%rbp)
43
                                      # 8-byte Spill
44
    callq strcpy
45
    movq -16(\%rbp), \%rcx
    movq $783, 520(%rcx)
movl $528, %edi
movq %rax, -56(%rbp)
47
                                       # imm = 0x30F
                                       # imm = 0x210
48
                                       # 8-byte Spill
49
50 callq malloc
movq %rax, -24(%rbp)
52 movq -24(%rbp), %rax
53 addq $8, %rax
```

```
movl $.L.str.4, %esi
movq %rax, %rdi
callq strcpy
     movq -24(\%rbp), \%rcx
57
                                     # imm = 0x108
     addq $264, %rcx
58
    movl $.L.str.5, %esi
movq %rcx, %rdi
movq %rax, -64(%rbp)
59
60
                                   # 8-byte Spill
61
    callq strcpy
    movq -24(%rbp), %rcx
movq $2,520(%rcx)
63
64
     movq -16(%rbp), %rcx
65
    movq -8(%rbp), %rdx
66
    movq %rcx, (%rdx)
    movq -24(%rbp), %rcx
68
           -16(%rbp), %rdx
69
    movq
    movq %rcx, (%rdx)
70
    movq -24(\%rbp), \%rcx
71
72
    movq $0, (%rcx)
    movq -8(%rbp), %rcx
movq %rax, -72(%rbp)
movq %rcx, %rax
73
                                   # 8-byte Spill
74
75
76
    addq $80, %rsp
77
     popq %rbp
    .cfi_def_cfa %rsp, 8
78
79
    retq
80 .Lfunc_end0:
size setup_users, .Lfunc_end0-setup_users
    .cfi_endproc
82
                                              # -- End function
83
    .globl print_users
                                       # -- Begin function print_users
    .p2align 4, 0x90
85
.type print_users,@function
87 print_users:
                                              # @print_users
88 .cfi_startproc
89 # %bb.0:
90 pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
92
    movq %rsp, %rbp
93
    .cfi_def_cfa_register %rbp
94
95
    subq $16, %rsp
    movq %rdi, -8(%rbp)
    movabsq $.L.str.6, %rdi
97
    movb $0, %al
98
99
    callq printf
movq $0, -16(%rbp)
                                              # =>This Inner Loop Header: Depth=1
101 .LBB1_1:
102 cmpq $0, -8(%rbp)
103
    je .LBB1_3
104 # %bb.2:
                                              # in Loop: Header=BB1_1 Depth=1
movq -16(%rbp), %rax
106
     addq $1, %rax
    movq %rax, -16(%rbp)
107
movq -8(%rbp)
addq $8, %rcx
           -8(%rbp), %rcx
movabsq $.L.str.7, %rdi
```

```
movq %rax, %rsi
movq %rcx, %rdx
movb $0, %al
     callq printf
114
    movq -8(%rbp), %rcx
movq (%rcx), %rcx
movq %rcx, -8(%rbp)
jmp .LBB1_1
115
116
117
118
119 .LBB1_3:
movabsq $.L.str.8, %rdi
     movb $0, %al
121
122
     callq printf
     addq $16, %rsp
123
     popq %rbp
124
     .cfi_def_cfa %rsp, 8
125
126
     retq
127 .Lfunc_end1:
.size print_users, .Lfunc_end1-print_users
129
     .cfi_endproc
                                                # -- End function
130
131
     .globl getUser
                                         # -- Begin function getUser
     .p2align 4, 0x90
132
.type getUser,@function
134 getUser:
                                                # @getUser
.cfi_startproc
136 # %bb.0:
pushq %rbp
     .cfi_def_cfa_offset 16
138
     .cfi_offset %rbp, -16
139
    movq %rsp, %rbp
140
141
     .cfi_def_cfa_register %rbp
142 subq $32, %rsp
movq %rdi, -16(%rbp)
movq %rsi, -24(%rbp)
145 .LBB2_1:
                                                # =>This Inner Loop Header: Depth=1
146    cmpq    $0, -16(%rbp)
147    je    .LBB2_5
148 # %bb.2:
                                                # in Loop: Header=BB2_1 Depth=1
149 movq -16(%rbp), %rax

150 addq $8, %rax

151 movq -24(%rbp), %rsi

152 movq %rax, %rdi
callq strcmp
cmpl $0, %eax
     jne .LBB2_4
155
156 # %bb.3:
movq -16(%rbp), %rax
movq %rax, -8(%rbp)
jmp .LBB2_6
160 .LBB2_4:
                                                # in Loop: Header=BB2_1 Depth=1
movq -16(%rbp), %rax
162 movq (%rax), %rax
movq %rax, -16(%rbp)
164 jmp .LBB2_1
165 .LBB2_5:
166 movq $0, -8(%rbp)
167 .LBB2_6:
```

```
movq -8(%rbp), %rax
addq $32, %rsp
popq %rbp
168
170
     .cfi_def_cfa %rsp, 8
171
172
    retq
173 .Lfunc_end2:
    .size getUser, .Lfunc_end2-getUser
174
     .cfi_endproc
175
                                               # -- End function
176
                                        # -- Begin function main
177
    .globl main
     .p2align 4, 0x90
178
179
    .type main,@function
180 main:
                                               # @main
.cfi_startproc
182 # %bb.0:
    pushq %rbp
183
184
     .cfi_def_cfa_offset 16
     .cfi_offset %rbp, -16
185
     movq %rsp, %rbp
     .cfi_def_cfa_register %rbp
187
     subq $576, %rsp
movl $0, -4(%rbp)
                                      # imm = 0x240
188
189
     callq setup_users
190
191
     movq %rax, -16(%rbp)
     movabsq $.L.str.9, %rdi
192
193
     movb $0, %al
     callq printf
194
     movabsq $.L.str.10, %rdi
195
     movl %eax, -548(%rbp)
movb $0, %al
                                      # 4-byte Spill
196
197
     callq printf
198
     leaq -272(%rbp), %rsi
199
     movabsq $.L.str.11, %rdi
200
     mov1 \%eax, -552(\%rbp)
                                      # 4-byte Spill
201
     movb $0, %al
202
203
     callq __isoc99_scanf
     leaq -272(%rbp), %rsi
204
     movq -16(%rbp), %rdi
movl %eax, -556(%rbp)
205
                                     # 4-byte Spill
206
     callq getUser
movq %rax, -280(%rbp)
cmpq $0, -280(%rbp)
207
208
209
    jne .LBB3_2
211 # %bb.1:
    leaq -272(%rbp), %rsi
212
213
     movabsq $.L.str.12, %rdi
214 movb $0, %al
215 callq printf
    mov1 $0, -4(\%rbp)
216
217
    jmp .LBB3_5
218 .LBB3_2:
movabsq $.L.str.13, %rdi
220 movb $0, %al
221
222
     callq printf
     leaq -544(%rbp), %rsi
movabsq $.L.str.11, %rdi
224 movl %eax, -560(%rbp)
                                 # 4-byte Spill
```

```
225 movb $0, %al
     callq __isoc99_scanf
leaq -544(%rbp), %rsi
226
227
     movq -280(%rbp), %rcx
228
     addq $264, %rcx
                                     # imm = 0x108
229
    movq %rcx, %rdi
movl %eax, -564(%rbp)
230
231
                                    # 4-byte Spill
232
    callq strcmp
233 cmpl $0, %eax
234  je .LBB3_4
235 # %bb.3:
movabsq $.L.str.14, %rdi
237 movb $0, %al
callq printf
movl $0, -4(\%rbp)
     jmp .LBB3_5
240
241 .LBB3_4:
_{242} movq -280(%rbp), %rax
     addq $8, %rax
243
     movabsq $.L.str.15, %rdi
244
     movq %rax, %rsi
movb $0, %al
245
246
     callq printf
247
     movabsq $.L.str.8, %rdi
248
    movl %eax, -568(%rbp)
movb $0, %al
                                   # 4-byte Spill
249
250
     callq printf
251
    movq -280(%rbp), %rcx addq $8, %rcx
252
253
     movabsq $.L.str.16, %rdi
254
    movq %rcx, %rsi
movl %eax, -572(%rbp)
255
                                # 4-byte Spill
256
     movb $0, %al
257
258
     callq printf
    movq -280(%rbp), %rcx
movq 520(%rcx), %rsi
259
260
261
     movabsq $.L.str.17, %rdi
262 movl %eax, -576(%rbp)
263 movb $0, %al
                                     # 4-byte Spill
264
    callq printf
265 .LBB3_5:
266 movl -4(%rbp), %eax
                                    # imm = 0x240
     addq $576, %<mark>rsp</mark>
268
    popq %rbp
    .cfi_def_cfa %rsp, 8
269
270
    retq
271 .Lfunc_end3:
.size main, .Lfunc_end3-main
    .cfi_endproc
273
274
                                              # -- End function
                                    # @.str
    .type .L.str,@object
275
.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"
    .size .L.str.1, 12
284
                              # @.str.2
    .type .L.str.2,@object
286
287 .L.str.2:
    .asciz "alice"
288
    .size .L.str.2, 6
289
    .type .L.str.3,@object
                           # @.str.3
291
292 .L.str.3:
.asciz "!alice12!_veuje@@hak"
   .size .L.str.3, 21
294
295
    .type .L.str.4,@object # @.str.4
296
297 .L.str.4:
298 .asciz "abdul"
    .size .L.str.4, 6
299
300
                                # @.str.5
    .type .L.str.5,@object
301
302 .L.str.5:
303 .asciz "passw0rd123"
304 .size .L.str.5, 12
305
    .type .L.str.6,@object
                                # @.str.6
306
307 .L.str.6:
308 .asciz "--- USERS ---\n"
    .size .L.str.6, 15
309
310
    .type .L.str.7,@object
                                 # @.str.7
311
312 .L.str.7:
313 .asciz " %021d. %s\n"
314 .size .L.str.7, 12
315
    .type .L.str.8,@object
                                # @.str.8
316
317 .L.str.8:
318 .asciz "\n"
319
    .size .L.str.8, 2
320
321
    .type .L.str.9,@object
322 .L.str.9:
.asciz "Welcome to BigBank Australia!\n"
324 .size .L.str.9, 31
325
    .type .L.str.10,@object # @.str.10
326
327 .L.str.10:
328 .asciz "Username: "
    .size .L.str.10, 11
329
330
    .type .L.str.11,@object
                             # @.str.11
332 .L.str.11:
333 .asciz "%255s"
334
    .size .L.str.11, 6
335
    .type .L.str.12,@object
                                # @.str.12
337 .L.str.12:
.asciz "User < %s > does not exist.\n"
```

```
339 .size .L.str.12, 29
340
    .type .L.str.13,@object
                               # @.str.13
341
342 .L.str.13:
343 .asciz "Password: "
344
    .size .L.str.13, 11
345
    .type .L.str.14,@object
                                 # @.str.14
346
347 .L.str.14:
348 .asciz "ERROR: incorrect password \n"
    .size .L.str.14, 27
349
350
    .type .L.str.15,@object
                                # @.str.15
351
352 .L.str.15:
.asciz "Logged in as < %s > ! \n"
    .size .L.str.15, 22
354
355
.type .L.str.16,@object
                                  # @.str.16
357 .L.str.16:
358 .asciz "Welcome, %s!\n"
359
    .size .L.str.16, 14
360
    .type .L.str.17,@object # @.str.17
361
362 .L.str.17:
363 .asciz "Your balance: \frac{n}{n}
    .size .L.str.17, 20
364
365
.ident "clang version 10.0.0-4ubuntu1"
    .section ".note.GNU-stack","", @progbits
367
    .addrsig
368
369
    .addrsig_sym setup_users
    .addrsig_sym malloc
370
    .addrsig_sym strcpy
371
372
    .addrsig_sym printf
373
    .addrsig_sym getUser
374
    .addrsig_sym strcmp
.addrsig_sym __isoc99_scanf
```

F.12 password-O3.s

```
1 .text
   .file "password.c"
.globl setup_users
                                       # -- Begin function setup_users
4 .p2align 4, 0x90
5 .type setup_users, @function
                                             # @setup_users
6 setup_users:
7 .cfi_startproc
8 # %bb.0:
9 pushq %r14
   .cfi_def_cfa_offset 16
pushq %rbx
    .cfi_def_cfa_offset 24
12
13
   pushq %rax
   .cfi_def_cfa_offset 32
14
.cfi_offset %rbx, -24
16 .cfi_offset %r14, -16
17 movl $528, %edi
                                    # imm = 0x210
```

```
18 callq malloc
    movq %rax, %r14
movl $1768776801, 8(%rax)
19
                                     # imm = 0x696D6461
20
    movw $110, 12(\%rax)
21
    movabsq 3773839939640058932, %rax # imm = 0x345F5F6E316D6434
22
    movq %rax, 264(%r14)
movl $6387301, 272(%r14)
23
                                     # imm = 0x617665
24
    movq $1000000, 520(%r14)
movl $528, %edi
                                    # imm = 0xF4240
25
                                     # imm = 0x210
27
    callq malloc
    movq %rax, %rbx
movl $1667853409, 8(%rax)
28
                                  # imm = 0x63696C61
29
    movw $101, 12(%rax)
30
    movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
    movq %rax, 277(%rbx)
32
    movups .L.str.3(%rip), %xmm0
movups %xmm0, 264(%rbx)
33
34
    movq $783, 520(%rbx)
                                     # imm = 0x30F
35
36
    movl $528, %edi
                                    # imm = 0x210
    {\tt callq\ malloc}
37
    movl $1969513057, 8(%rax)
                                    # imm = 0x75646261
    movw $108, 12(%rax)
39
    movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
40
41
    movq %rcx, 264(%rax)
    movl $3355185, 272(%rax)
movq $2, 520(%rax)
movq %rbx, (%r14)
                                   # imm = 0x333231
42
43
44
    movq %rax, (%rbx)
45
    movq $0, (%rax)
46
    movq %r14, %rax addq $8, %rsp
47
48
    .cfi_def_cfa_offset 24
49
    popq %rbx
50
    .cfi_def_cfa_offset 16
51
52
    popq %r14
53
    .cfi_def_cfa_offset 8
54
    retq
55 .Lfunc_end0:
    .size setup_users, .Lfunc_end0-setup_users
56
57
    .cfi_endproc
                                              # -- End function
59
    .globl print_users
                                        # -- Begin function print_users
o.p2align 4, 0x90
.type print_users,@function
62 print_users:
                                              # @print_users
63
   .cfi_startproc
64 # %bb.0:
pushq %r14
    .cfi_def_cfa_offset 16
66
    pushq %rbx
    .cfi_def_cfa_offset 24
68
69
    pushq %rax
    .cfi_def_cfa_offset 32
70
    .cfi_offset %rbx, -24
71
    .cfi_offset %r14, -16
72
73 movq %rdi, %r14
74 movl $.Lstr, %edi
```

```
75 callq puts
    testq %r14, %r14
77 je .LBB1_3
78 # %bb.1:
^{79} movl $1, \%ebx
80
    .p2align 4, 0x90
                                           # =>This Inner Loop Header: Depth=1
81 .LBB1_2:
82 leaq 8(%r14), %rdx
movl $.L.str.7, %edi
movq %rbx, %rsi
so xorl %eax, %eax
    callq printf
86
    movq (%r14), %r14
addq $1, %rbx
87
89
    testq %r14, %r14
    jne .LBB1_2
90
91 .LBB1_3:
92 movl $10, %edi
    addq $8, %rsp
    .cfi_def_cfa_offset 24
94
    popq %rbx
    .cfi_def_cfa_offset 16
96
97 popq %r14
.cfi_def_cfa_offset 8
    jmp putchar
                                 # TAILCALL
99
.Lfunc_end1:
.size print_users, .Lfunc_end1-print_users
102
    .cfi_endproc
                                           # -- End function
103
    .globl getUser
                                     # -- Begin function getUser
104
.p2align 4, 0x90
.type getUser,@function
107 getUser:
                                           # @getUser
.cfi_startproc
109 # %bb.0:
pushq %r14
    .cfi_def_cfa_offset 16
111
pushq %rbx
    .cfi_def_cfa_offset 24
113
114
    pushq %rax
    .cfi_def_cfa_offset 32
115
.cfi_offset %rbx, -24
    .cfi_offset %r14, -16
117
testq %rdi, %rdi
    je .LBB2_4
119
120 # %bb.1:
movq %rsi, %r14
movq %rdi, %rbx
    .p2align 4, 0x90
123
124 .LBB2_2:
                                           # =>This Inner Loop Header: Depth=1
125 leaq 8(%rbx), %rdi
movq %r14, %rsi
127 callq strcmp
testl %eax, %eax
je .LBB2_5
130 # %bb.3:
                                           # in Loop: Header=BB2_2 Depth=1
131 movq (%rbx), %rbx
```

```
testq %rbx, %rbx
133
    jne .LBB2_2
134 .LBB2 4:
xorl %ebx, %ebx
136 .LBB2_5:
movq %rbx, %rax addq $8, %rsp
    .cfi_def_cfa_offset 24
139
    popq %rbx
140
    .cfi_def_cfa_offset 16
141
142
    popq %r14
143
    .cfi_def_cfa_offset 8
144 retq
145 .Lfunc_end2:
.size getUser, .Lfunc_end2-getUser
    .cfi_endproc
147
                                            # -- End function
148
149 .globl main
                                      # -- Begin function main
.p2align 4, 0x90
.type main, @function
                                            # @main
152 main:
.cfi_startproc
154 # %bb.0:
pushq %r15
    .cfi_def_cfa_offset 16
156
157
    pushq %r14
    .cfi_def_cfa_offset 24
158
    pushq %rbx
159
    .cfi_def_cfa_offset 32
160
                                    # imm = 0x200
    subq $512, %rsp
161
162
    .cfi_def_cfa_offset 544
    .cfi_offset %rbx, -32
163
    .cfi_offset %r14, -24
164
    .cfi_offset %r15, -16
165
    movl $528, %edi
                                    \# imm = 0x210
166
167
    callq malloc
    movq %rax, %rbx
168
    movl $1768776801, 8(%rax)
movw $110, 12(%rax)
169
                                    # imm = 0x696D6461
170
171
    movabsq \$3773839939640058932, \%rax # imm = 0x345F5F6E316D6434
    movq %rax, 264(%rbx)
172
    movl $6387301, 272(%rbx)
173
                                   # imm = 0x617665
174
    movq $1000000, 520(%rbx)
                                   # imm = 0xF4240
    movl $528, %edi
                                    # imm = 0x210
175
    callq malloc
176
    movq %rax, %r14
movl $1667853409, 8(%rax)
177
                                 # imm = 0x63696C61
178
179
    movw $101, 12(%rax)
    movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
180
    movq %rax, 277(%r14)
181
    movups .L.str.3(%rip), %xmm0
movups %xmm0, 264(%r14)
182
183
    movq $783, 520(%r14)
movl $528, %edi
184
                                   # imm = 0x30F
                                    # imm = 0x210
185
    callq malloc
movl $1969513057, 8(%rax)
                                 # imm = 0x75646261
movw $108, 12(%rax)
```

```
movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
189
     movq %rcx, 264(%rax)
movl $3355185, 272(%rax)
190
                                       # imm = 0x333231
191
     movq $2, 520(%rax)
192
     movq %r14, (%rbx)
193
     movq %rax, (%r14)
movq $0, (%rax)
movl $.Lstr.18, %edi
194
195
196
     callq puts
197
     movl $.L.str.10, %edi
xorl %eax, %eax
198
199
200
     callq printf
     movq %rsp, %rsi
201
202 movl $.L.str.11, %edi
     xorl %eax, %eax
203
callq __isoc99_scanf
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
testl %eax, %eax
216 je .LBB3_5
217 # %bb.3:
                                                      in Loop: Header=BB3_2 Depth=1
movq (%rbx), %rbx
testq %rbx, %rbx
220     jne .LBB3_2
221 .LBB3_4:
222 movq %rsp, %rsi
223 movl $.L.str.12, %edi
224 .LBB3_8:
225 xorl %eax, %eax
226 callq printf
227 jmp .LBB3_9
228 .LBB3_5:
229 movl $.L.str.13, %edi
230 xorl %eax, %eax
callq printf
    leaq 256(%rsp), %r15
movl $.L.str.11, %edi
movq %r15, %rsi
xorl %eax, %eax
232
233
234
235
236
     callq __isoc99_scanf
     leaq 264(%rbx), %rdi
movq %r15, %rsi
237
238
     callq strcmp
239
testl %eax, %eax
241 je .LBB3_7
242 # %bb.6:
movl $.Lstr.19, %edi
244 callq puts
245 .LBB3_9:
```

```
246 xorl %eax, %eax
247 addq $512, %rsp
248 .cfi_def_cfa_offset 32
                                  # imm = 0x200
    popq %rbx
249
    .cfi_def_cfa_offset 24
250
251
    popq %r14
252
     .cfi_def_cfa_offset 16
253 popq %r15
254
    .cfi_def_cfa_offset 8
255 retq
256 .LBB3_7:
\tt .cfi\_def\_cfa\_offset 544
    movl $.L.str.15, %edi
258
259 movq %r14, %rsi
260 xorl %eax, %eax
    callq printf movl $10, %edi
261
262
263 callq putchar
264 movl $.L.str.16, %edi
    movq %r14, %rsi
xorl %eax, %eax
265
266
267 callq printf
268 movq 520(%rbx), %rsi
269 movl $.L.str.17, %edi
    jmp .LBB3_8
270
271 .Lfunc_end3:
.size main, .Lfunc_end3-main
    .cfi_endproc
273
                                            # -- End function
274
    .type .L.str,@object # @.str
275
276
    .section .rodata.str1.1, "aMS", @progbits,1
277 .L.str:
278 .asciz "admin"
    .size .L.str, 6
279
280
281
    .type .L.str.1,@object
                                  # @.str.1
282 .L.str.1:
283 .asciz "4dm1n__4eva"
    .size .L.str.1, 12
284
285
.type .L.str.2,@object
                               # @.str.2
287 .L.str.2:
288 .asciz "alice"
    .size .L.str.2, 6
289
290
    .type .L.str.3,@object
291
                                  # @.str.3
292 .L.str.3:
.asciz "!alice12!_veuje@@hak"
294
    .size .L.str.3, 21
295
.type .L.str.4,@object
                             # @.str.4
297 .L.str.4:
298 .asciz "abdul"
    .size .L.str.4, 6
299
.type .L.str.5,@object
                                  # @.str.5
302 .L.str.5:
```

```
303 .asciz "passw0rd123"
304
    .size .L.str.5, 12
305
    .type .L.str.7,@object
                                 # @.str.7
306
307 .L.str.7:
308 .asciz " %021d. %s\n"
    .size .L.str.7, 12
309
310
311 .type .L.str.10,@object
                                 # @.str.10
312 .L.str.10:
313 .asciz "Username: "
    .size .L.str.10, 11
314
315
316
   .type .L.str.11,@object
                               # @.str.11
317 .L.str.11:
    .asciz "%255s"
318
    .size .L.str.11, 6
319
320
.type .L.str.12,@object # @.str.12
322 .L.str.12:
323 .asciz "User < %s > does not exist.\n"
    .size .L.str.12, 29
324
325
    .type .L.str.13,@object
                               # @.str.13
326
327 .L.str.13:
    .asciz "Password: "
328
    .size .L.str.13, 11
329
330
.type .L.str.15,@object
                                 # @.str.15
332 .L.str.15:
.asciz "Logged in as < %s >!\n"
    .size .L.str.15, 22
334
335
                              # @.str.16
    .type .L.str.16,@object
336
337 .L.str.16:
338 .asciz "Welcome, %s!\n"
    .size .L.str.16, 14
339
340
   .type .L.str.17,@object # @.str.17
341
342 .L.str.17:
343 .asciz "Your balance: $%ld\n"
344
    .size .L.str.17, 20
345
                                # @str
    .type .Lstr,@object
346
347 .Lstr:
348 .asciz "--- USERS ---"
   .size .Lstr, 14
349
350
    .type .Lstr.18,@object
                                # @str.18
351
352 .Lstr.18:
.asciz "Welcome to BigBank Australia!"
    .size .Lstr.18, 30
354
355
    .type .Lstr.19,@object # @str.19
356
357 .Lstr.19:
.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
    .file "deadStoreElimination.c"
    .globl deadStore
                                      # -- Begin function deadStore
   .p2align 4, 0x90
 5 .type deadStore, @function
                                            # @deadStore
 6 deadStore:
    .cfi_startproc
 8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
10
    .cfi_offset %rbp, -16
11
12
   movq %rsp, %rbp
    .cfi_def_cfa_register %rbp
13
14 movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
16 movl $43981, -12(%rbp)
17 movl $0, -16(%rbp)
                                   # imm = OxABCD
18 .LBB0_1:
                                            # =>This Inner Loop Header: Depth=1
19 movl -4(%rbp), %eax
20 cmpl -8(%rbp), %eax
    jle .LBB0_3
21
22 # %bb.2:
                                            # in Loop: Header=BB0_1 Depth=1
movl -12(%rbp), %eax
24 addl -16(%rbp), %eax
25 movl %eax, -16(%rbp)
26 movl -4(%rbp), %eax
27 addl $-1, %eax
movl \%eax, -4(\%rbp)
29 jmp .LBB0_1
30 .LBB0_3:
31
   movl $0, -12(%rbp)
   movl -4(%rbp), %eax
32
33
    addl -8(\%rbp), \%eax
34 popq %rbp
35
36
    .cfi_def_cfa %rsp, 8
    retq
37 .Lfunc_end0:
.size deadStore, .Lfunc_endO-deadStore
    .cfi_endproc
39
                                           # -- End function
40
    .globl main
                                      # -- Begin function main
41
.p2align 4, 0x90
.type main, @function
44 main:
                                            # @main
45 .cfi_startproc
46 # %bb.0:
47 pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
```

```
.cfi_def_cfa_register %rbp
    subq $32, %rsp
movl %edi, -4(%rbp)
52
53
   movq %rsi, -16(%rbp)
54
   movl -4(%rbp), %edi
movl $2, %esi
55
56
    callq deadStore
57
xorl %ecx, %ecx
59 movl %eax, -20(%rbp)
                                  # 4-byte Spill
60 movl %ecx, %eax
61 addq $32, %rsp
62 popq %rbp
    .cfi_def_cfa %rsp, 8
63
64 retq
65 .Lfunc_end1:
    .size main, .Lfunc_end1-main
67
    .cfi_endproc
                                             # -- End function
68
.ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","",@progbits
    .addrsig
.addrsig_sym deadStore
```

F.14 deadStoreElimination-O3.s

```
2 .file "deadStoreElimination.c"
   .globl deadStore
                                   # -- Begin function deadStore
   .p2align 4, 0x90
5 .type deadStore,@function
                                         # @deadStore
 6 deadStore:
7 .cfi_startproc
 8 # %bb.0:
                                         # kill: def $esi killed $esi def $rsi
                                         # kill: def $edi killed $edi def $rdi
10
cmpl %edi, %esi
cmovlel %esi, %edi
leal (%rdi %rei)
   leal (%rdi,%rsi), %eax
14 retq
.Lfunc_end0:
.size deadStore, .Lfunc_endO-deadStore
   .cfi_endproc
17
                                         # -- End function
18
                                   # -- Begin function main
19 .globl main
.p2align 4, 0x90
.type main, @function
22 main:
                                         # @main
.cfi_startproc
24 # %bb.0:
25 xorl %eax, %eax
26 retq
27 .Lfunc_end1:
   .size main, .Lfunc_end1-main
28
   .cfi_endproc
29
                                         # -- End function
.ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","", @progbits
```

33 .addrsig

F.15 pread-O0.s

```
1 .text
 2 .file "pread.c"
 3 .globl main
                                    # -- Begin function main
p2align 4, 0x90
type main, @function
                                           # @main
6 main:
7 .cfi_startproc
8 # %bb.0:
9 pushq %rbp
   .cfi_def_cfa_offset 16
   .cfi_offset %rbp, -16
11
movq %rsp, %rbp
.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
                                              Child Loop BBO_2 Depth 2
                                                  Child Loop BBO_3 Depth 3
19
   jmp .LBB0_2
20
21 .LBB0_2:
                                           # Parent Loop BB0_1 Depth=1
                                           # => This Loop Header: Depth=2
                                                 Child Loop BBO_3 Depth 3
25 .LBB0_3:
                                           # Parent Loop BB0_1 Depth=1
                                               Parent Loop BBO_2 Depth=2
                                           # => This Inner Loop Header: Depth=3
28 movl z, %eax
29 movl %eax, -8(%rbp)
30 # %bb.4:
                                           # in Loop: Header=BB0_3 Depth=3
movl -8(\%rbp), \%eax
32
   cltd
   movl $2, %ecx
33
   idivl %ecx
35 cmpl $0, %edx
jne .LBB0_3
37 # %bb.5:
                                              in Loop: Header=BBO_2 Depth=2
38 movl x, %eax
39 movl %eax, -12(%rbp)
40 # %bb.6:
                                           # in Loop: Header=BB0_2 Depth=2
movl z, %eax
42 cmpl -8(%rbp), %eax
ine .LBB0_2
44 # %bb.7:
                                             in Loop: Header=BBO_1 Depth=1
jmp .LBB0_1
46 .Lfunc_end0:
.size main, .Lfunc_end0-main
   .cfi_endproc
48
                                           # -- End function
49
50 .type z,@object
                                 # @z
.comm z,4,4
.type x,@object
                                  # @x
53 .comm x,4,4
```

```
.ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","", Oprogbits
.addrsig
.addrsig_sym z
.addrsig_sym x
```

F.16 pread-O3.s

```
1 .text
   .file "pread.c"
.globl main
                                        # -- Begin function main
p2align 4, 0x90
type main, @function
                                               # @main
 6 main:
 7 .cfi_startproc
 8 # %bb.0:
9 .p2align 4, 0x90
10 .LBB0_1:
                                               # =>This Inner Loop Header: Depth=1
movl z(%rip), %eax
testb $1, %al
jne .LBB0_1
14 # %bb.2:
                                                  in Loop: Header=BBO_1 Depth=1
15  movl x(%rip), %eax
16  movl z(%rip), %eax
jmp .LBB0_1
18 .Lfunc_end0:
.size main, .Lfunc_end0-main
cfi_endproc
                                               # -- End function
21
                                    # @z
   .type z,@object
22
23 .comm z,4,4
                                      # @x
    .type x,@object
    .comm x,4,4
25
    .ident "clang version 10.0.0-4ubuntu1 "
.section ".note.GNU-stack","", @progbits
26
27
28 .addrsig
29 .addrsig_sym z
30 .addrsig_sym x
```