Compiler Annotation Solutions 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

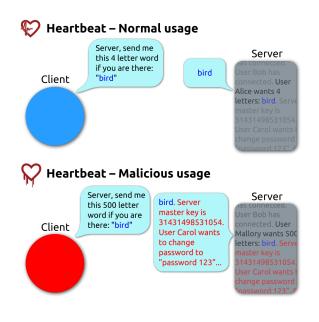


Figure 1: The Heartbleed bug. [13]

of "hello" could be sent with the length of the message, 5. However, due to a improper input validation (buffer over-read), one could send a length longer that the string they actually sent. This would cause the server to respond with the original message and anything that was in the allocated memory at the time, including any potentially sensitive information. An example of this is shown in figure 1 [13].

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

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 authorised users. Together, these principles create the CIA triad [28]. To enforce a secure system, all three principles must be upheld.

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

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

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

Figure 2: Implicit flow of data to a public variable

plicit data flow, where the state of secure data can be inferred from the state of public data or a public variable [9]. Take for example figure 2. In this example, a public, readable variable is initially set to the value of 1. There is also a secret variable which may contain a key, password or some other secret that must be kept secure from any attackers. Depending on the value of the secret variable an attacker can infer information about this variable depending on whether the value of the public variable is updated to a value of 0. Assuming that the inner workings of the system is known by the attacker, information about the secret variable can be leaked implicitly and inferred by the state of public variables.

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

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

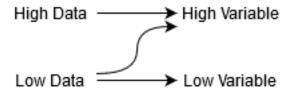


Figure 3: Permitted flow of data

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

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 concur-1 crypt() { rent threads can interleave their steps and behaviour, 2 there is a lot of complexity and possibilities for the ³ overall behaviour. This concept of assumptions (or $\frac{4}{5}$ 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

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

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

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

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

This leads to the question, what security guarantees in source code are being violated by compiler optimisations? Although one could analyse each individual compiler optimisation to check for potential security violations in source code, defensively programming against the compiler can be counter-initiative. Additionally, compilers are getting better at optimising away tricks programmers write to work against the

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

2.5 Annotations

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

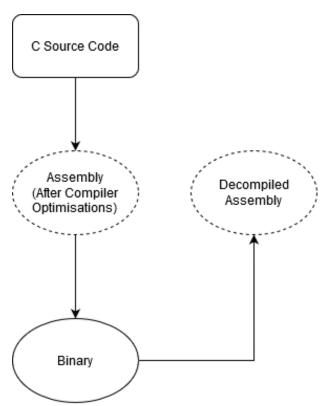


Figure 5: The static analysis options after compilation

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

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

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

Vu et al. [30] explore capturing and propagating properties from the source code level 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 amd memory locations. If an optimisation pass had cased the verification to fail the programmer would then be notified, to which they could annotate differently or disable the optimisation.

3 Approach and Execution

The approach was set out by first analysing existing methods of preserving annotations through intermediate representations. This primarily consisted of the CompCert compiler and assembly annotation tools provided with the compiler. It was found that the CompCert compiler could not handle all cases necessary for the wpif analysis, namely volatile variables and loop invariants. Following this, inline assembly, the GNU C extension, was explored as a possibility to preserve annotations in C. This technique prevailed and was found to be excellent in handling assembly annotations by injecting comments in to the compiled assembly output, and was. 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.

A suite of test C programs (See Appendix A) were created to assist in guiding the process of evaluating the CompCert compiler 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 elements 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.

TODO: Include:

- analysis of the results & quality
- runtime efficiency comparison

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

Table 1: CompCert install specifications

3.1 CompCert 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 with the *comment.c* test file, where the comment was replaced with the call to generate an annotation in the compiled assembly. 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"
);
```

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. The full compiled output can be seen in Appendix C.16.

3.1.1 CompCert Results

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() {
    // L(i) = High
    int i = 0;
    return i + 1;
}
```

A.3 volatile.c

```
volatile int x;

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

A.4 loop.c

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

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

A.6 password.c

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

#define BUFF_LEN 256

typedef struct user_t user_t;

struct user_t {
    user_t* next;
    char name[BUFF_LEN];
    // L(password) = High
```

```
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
      user_admin->balance = 1000000;
21
22
      user_t* user_alice = malloc(sizeof(user_t));
23
      strcpy(user_alice->name, "alice");
24
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
26
      user_alice->balance = 783;
27
      user_t* user_abdul = malloc(sizeof(user_t));
28
29
      strcpy(user_abdul->name, "abdul");
      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
  void print_users(user_t* users) {
40
      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
52
           if (strcmp(user_list->name, name) == 0) {
               return user_list;
54
55
           user_list = user_list->next;
      }
56
      return NULL;
57
58 }
59
60 int main() {
      user_t* users = setup_users();
61
62
      printf("Welcome to BigBank Australia!\n");
63
64
65
      char username[BUFF_LEN];
      printf("Username: ");
66
67
      scanf("%255s", username);
68
   user_t* user = getUser(users, username);
69
```

```
if (user == NULL) {
70
71
           printf("User < %s > does not exist.\n", username);
           return 0;
72
73
74
      char password[BUFF_LEN];
75
      printf("Password: ");
76
      scanf("%255s", password);
77
78
      if (strcmp(user->password, password) != 0) {
           printf("ERROR: incorrect password\n");
79
           return 0;
80
81
82
      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
3
      // do some work
      int result = 0;
      while (i > n) {
          result += key;
8
9
          i--;
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);
18
19 }
```

A.8 pread.c

```
volatile int z;
volatile 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;
   // {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
23
              // {_Gamma: r1 -> LOW}
24
              r1 = z;
          } while (r1 %2 != 0);
25
              r2 = x;
26
          } 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

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);
9
       int r1 = 0;
       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);
__builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1, r2);
14
```

```
16
17
      while(1) {
18
      do {
           __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", r1
20
      , r2, z);
21
          do {
               __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
22
23
               __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
24
               r1 = z;
           } while (r1 %2 != 0);
25
              r2 = x;
26
27
           } while (z != r1);
      }
28
      return r2;
29
```

B.5 rooster.c

```
int rooster;
2 int drake;
3 int goose;
5 int fun(int a, int b, int c) {
       __builtin_ais_annot("%here CRITICAL COMMENT");
       static int count = 0;
      int sum = a + b + c;
      if (sum < 0) {</pre>
9
10
           return sum;
11
      if (a < b && b < c) {
12
13
           while (a != b) {
               a++;
14
15
               count++;
               while (b != c) {
16
                   c--;
17
18
                   count++;
               }
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
28
       rooster = 1;
       drake = 5;
29
       goose = 10;
30
      int result;
31
       result = fun(rooster, drake, goose);
32
33
       return 0;
34 }
```

B.6 password.c

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

```
#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
      size_t balance;
14 };
15
user_t* setup_users() {
      user_t* user_admin = malloc(sizeof(user_t));
17
18
      strcpy(user_admin->name, "admin");
      strcpy(user_admin->password, "4dm1n__4eva");
19
      __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
20
      user_admin->balance = 1000000;
21
22
      user_t* user_alice = malloc(sizeof(user_t));
23
      strcpy(user_alice->name, "alice");
24
      strcpy(user_alice->password, "!alice12!_veuje@@hak");
25
       __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
26
27
      user_alice->balance = 783;
28
      user_t* user_abdul = malloc(sizeof(user_t));
29
      strcpy(user_abdul->name, "abdul");
30
      strcpy(user_abdul->password, "passw0rd123");
31
       __builtin_ais_annot("%here L(%e1) = high", user_abdul->password);
32
      user_abdul->balance = 2;
33
34
35
      user_admin->next = user_alice;
36
      user_alice->next = user_abdul;
37
      user_abdul ->next = NULL;
38
39
      return user_admin;
40 }
41
42
  void print_users(user_t* users) {
      printf("--- USERS ---\n");
43
44
      size_t count = 0;
      while (users != NULL) {
45
          printf(" %021d. %s\n", ++count, users->name);
46
47
          users = users->next;
48
49
      printf("\n");
50 }
51
s2 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
```

```
return NULL;
59
60 }
61
62 int main() {
      user_t* users = setup_users();
63
64
      printf("Welcome to BigBank Australia!\n");
65
66
67
      char username[BUFF_LEN];
      printf("Username: ");
68
      scanf("%255s", username);
69
70
      user_t* user = getUser(users, username);
71
      if (user == NULL) {
72
           printf("User < %s > does not exist.\n", username);
73
74
           return 0;
75
76
77
      char password[BUFF_LEN];
      printf("Password: ");
78
79
      scanf("%255s", password);
      if (strcmp(user->password, password) != 0) {
80
           printf("ERROR: incorrect password\n");
81
82
           return 0;
83
84
      printf("Logged in as < %s >!\n", user->name);
85
      printf("\n");
86
      printf("Welcome, %s!\n", user->name);
87
      printf("Your balance: $%ld\n", user->balance);
88
89 }
```

B.7 deadStoreElimination.c

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

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);
      int r1 = 0;
9
      int r2 = 0;
                                           {L(r2)=False}
10
      __builtin_ais_annot("%here L(%e1)= false", r2);
11
12
      // Predicates on initial state
13
      __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
19
20
               __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1 <= %e2", r1, z);
               __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1 == %e3), %e3 -> LOW"
21
      , r1, r2, z);
              do {
22
                   __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
23
24
                   __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
                  r1 = z;
25
               } while (r1 %2 != 0);
26
                  r2 = x;
27
          } while (z != r1);
28
      }
29
      return r2;
30
31 }
```

C CompCert Annotated Assembly Output

C.1 comment-O0.s

```
1 # File generated by CompCert 3.7
2 # Command line: comment.c -00 -S
    .text
    .align 16
    .globl main
6 main:
    .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
9
    leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
10
11
12 .L100:
    xorl %eax, %eax
13
14
    addq $8, %rsp
15
    ret
16
    .cfi_endproc
    .type main, @function
17
    .size main, . - main
    .section "__compcert_ais_annotations","", @note
19
.ascii "# file:comment.c line:2 function:main\n"
```

```
.byte 7,8
.quad .L100
.ascii " Critical Comment\n"
```

C.2 comment-O3.s

```
_{\rm 1} # File generated by CompCert 3.7
2 # Command line: comment.c -03 -S
     .text
    .align 16
    .globl main
6 main:
    .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
10
11
12 .L100:
xorl %eax, %eax
    addq $8, %rsp
14
1.5
    ret
    .cfi_endproc
16
17
    .type main, @function
    .size main, . - main
.section "__compcert_ais_annotations","",@note
18
19
    .ascii "# file:comment.c line:2 function:main\n"
20
21
    .byte 7,8
    .quad .L100
22
.ascii " Critical Comment\n"
```

C.3 variable-O0.s

```
# File generated by CompCert 3.7
2 # Command line: variable.c -00 -S
```

C.4 variable-O3.s

```
# File generated by CompCert 3.7
2 # Command line: variable.c -03 -S
```

C.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.
```

C.6 volatile-O3.s

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

2 1 error detected.
```

C.7 loop-O0.s

```
# File generated by CompCert 3.7
2 # Command line: loop.c -00 -S
    .comm z, 4, 4
    .comm x, 4, 4
    .text
    .align 16
6
     .globl main
8 main:
   .cfi_startproc
subq $8, %rsp
    .cfi_adjust_cfa_offset 8
11
12 leaq 16(%rsp), %rax
13 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
29
    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
38
     testl %ecx, %ecx
     jne .L108
39
     movl x(%rip), %edi
movl z(%rip), %esi
40
41
     jmp .L105
42
43
     .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"
64
     .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"
71
     .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
76
     .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"
83
     .byte 7,8
84
85
     .quad .L107
     .ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\") == mem("
86
87
     .byte 7,8
     .quad z
     .ascii ", 4)), mem("
89
     .byte 7,8
90
91
     .quad z
     .ascii ", 4) \rightarrow 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"
100
     .byte 7,8
101
     .quad .L110
102
    .ascii " _Gamma: reg(\"rdx\") -> LOW\n"
103
```

C.8 loop-O3.s

```
# File generated by CompCert 3.7

# Command line: loop.c -03 -S

.comm z, 4, 4

.comm x, 4, 4

.text

.align 16

.globl main

main:
```

```
9
    .cfi_startproc
10
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
11
    leaq 16(%rsp), %rax
12
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
46
    .section "__compcert_ais_annotations","", Onote
    .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
53
     .ascii ", 4)) = true\n"
     .ascii "# file:loop.c line:8 function:main\n"
54
    .byte 7,8
55
56
    .quad .L101
     .ascii " L(mem("
57
     .byte 7,8
58
    .quad x
59
     .ascii ", 4))= mem("
60
61
     .byte 7,8
    .quad z
62
     .ascii ", 4) % 2 == 0\n"
63
    .ascii "# file:loop.c line:11 function:main\n"
64
65 .byte 7,8
```

```
.quad .L102
66
67
     .ascii " L(0) = false \n"
     .ascii "# file:loop.c line:14 function:main\n"
68
     .byte 7,8
69
     .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
74
     .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
84
     .byte 7,8
     .quad .L107
85
     .ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\") == mem("
86
     .byte 7,8
87
     .quad z
88
     .ascii ", 4)), mem("
89
     .byte 7,8
90
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
96
     .ascii " _invariant: reg(\"rdx\") <= mem("
     .byte 7,8
97
     .quad z
98
     .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"
```

C.9 rooster-O0.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -00 -S
   .comm rooster, 4, 4
    .comm drake, 4, 4
    .comm goose, 4, 4
    .data
6
    .align 4
8 count:
    .long 0
10
   .type count, @object
11
    .size count, . - count
12
    .text
   .align 16
13
.globl fun
15 fun:
.cfi_startproc
```

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

```
movl %eax, drake(%rip)
movl $10, %eax
movl %eax, goose(%rip)
74
75
76
     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
82
83
     ret
     .cfi_endproc
84
     .type main, @function
85
     .size main, . - main
86
     .section "__compcert_ais_annotations","",@note
87
     .ascii "# file:rooster.c line:6 function:fun\n"
88
     .byte 7,8
89
     .quad .L100
90
     .ascii " CRITICAL COMMENT\n"
91
92
     .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
99
     .byte 7,8
100
     .quad .L108
101
.ascii " EXCEPTIONAL\n"
```

C.10 rooster-O3.s

```
# File generated by CompCert 3.7
2 # Command line: rooster.c -03 -S
    .comm rooster, 4, 4
    .comm drake, 4, 4
    .comm goose, 4, 4
    .data
    .align 4
8 count:
   .long 0
9
   .type count, @object
10
11
    .size count, . - count
12
    .text
   .align 16
13
.globl fun
15 fun:
16
    .cfi_startproc
    subq $8, %rsp
17
    .cfi_adjust_cfa_offset 8
18
   leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
19
20
21 .L100:
leal 0(%edi,%esi,1), %r9d
leal 0(%r9d,%edx,1), %eax
testl %eax, %eax
25 jl .L101
```

```
cmpl %edx, %esi
setl %al
26
    movzbl %al, %eax
28
    xorl %r8d, %r8d
29
    cmpl %esi, %edi
30
    cmovge %r8, %rax
31
    cmpl $0, %eax je .L102
32
33
34 .L103:
    cmpl %esi, %edi
35
    je .L102
36
    leal 1(%edi), %edi
37
    movl count(%rip), %ecx
38
    leal 1(%ecx), %r8d
    movl %r8d, count(%rip)
40
41 .L104:
    cmpl %edx, %esi
42
    je .L103
43
    leal -1(\%edx), \%edx
44
    movl count(%rip), %r10d leal 1(%r10d), %r8d
45
    mov1 %r8d, count(%rip)
47
    jmp .L104
48
49 .L102:
   movl
          count(%rip), %eax
50
51 .L101:
    addq $8, %rsp
52
53
    ret
    .cfi_endproc
54
    .type fun, @function
55
56
    .size fun, . - fun
57
    .text
    .align 16
58
    .globl main
59
60 main:
61
    .cfi_startproc
    subq $8, %rsp
62
    .cfi_adjust_cfa_offset 8
    leaq 16(%rsp), %rax
movq %rax, 0(%rsp)
64
65
66 .L105:
67 .L106:
   movl $1, %eax
    movl %eax, rooster(%rip)
69
70
    movl
           $5, %eax
    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
80
    ret
    .cfi_endproc
81
.type main, @function
```

```
.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"
89
    .byte 7,8
90
    .quad .L105
91
    .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"
96
97
    .byte 7,8
    .quad .L106
   .ascii " EXCEPTIONAL\n"
```

C.11 password-O0.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -00 -S
   .section .rodata
    .align 1
5 __stringlit_7:
  .ascii "--- USERS ---\012\000"
    .type __stringlit_7, @object
    .size __stringlit_7, . - __stringlit_7
9
    .section .rodata
10
   .align 1
11 __stringlit_6:
.ascii "passw0rd123\000"
    .type __stringlit_6, @object
13
    .size __stringlit_6, . - __stringlit_6
14
   .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
    .section .rodata
21
    .align 1
22
23 __stringlit_14:
    .ascii "Password: \000"
24
   .type __stringlit_14, @object
25
   .size __stringlit_14, . - __stringlit_14
26
27
    .section .rodata
28
    .align 1
29 __stringlit_18:
   .ascii "Your balance: $%ld\012\000"
31
    .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
```

```
.size __stringlit_13, . - __stringlit_13
    .section .rodata
    .align 1
40
41 __stringlit_8:
  .ascii " %02ld. %s\012\000"
42
    .type __stringlit_8, @object
43
44
    .size __stringlit_8, . - __stringlit_8
    .section .rodata
45
    .align 1
47 __stringlit_1:
    .ascii "admin\000"
48
    .type __stringlit_1, @object
49
    .size __stringlit_1, . - __stringlit_1
50
51
    .section .rodata
    .align 1
52
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:
   .ascii "alice\000"
60
    .type __stringlit_3, @object
61
    .size __stringlit_3, . - __stringlit_3
62
63
    .section .rodata
    .align 1
64
65 __stringlit_11:
  .ascii "Username: \000"
66
    .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"
78
    .type __stringlit_17, @object
79
    .size __stringlit_17, . - __stringlit_17
    .section .rodata
81
    .align 1
82
83 __stringlit_12:
    .ascii "%255s\000"
84
85
    .type __stringlit_12, @object
    .size __stringlit_12, . - __stringlit_12
86
87
    .section .rodata
    .align 1
88
89 __stringlit_9:
  .ascii "\012\000"
90
    .type __stringlit_9, @object
91
    .size __stringlit_9, . - __stringlit_9
    .section .rodata
93
94 .align 1
```

```
95 __stringlit_15:
     .ascii "ERROR: incorrect password\012\000"
.type __stringlit_15, @object
97
     .size __stringlit_15, . - __stringlit_15
98
     .section .rodata
99
     .align 1
100
101 __stringlit_10:
     .ascii "Welcome to BigBank Australia!\012\000"
102
     .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
     .size __stringlit_16, . - __stringlit_16
110
111
     .text
     .align 16
112
113
   .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
movq %rax, %rbp
leaq 8(%rbp), %rdi
124
125
126
     leaq __stringlit_1(%rip), %rsi
127
128
     call strcpy
     leaq 264(%rbp), %rdi
129
     leaq __stringlit_2(%rip), %rsi
call strcpy
130
131
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
             strcpy
     leaq 264(%r12), %rdi
141
     leaq __stringlit_4(%rip), %rsi
call strcpy
142
143
144 .L101:
     movq $783, %r9
145
     movq %r9, 520(%r12)
146
147
     movq $528, %rdi
     call
movq
            malloc
148
149
            %rax, %rbx
     leaq 8(%rbx), %rdi
150
151 leaq __stringlit_5(%rip), %rsi
```

```
call strcpy
152
153
      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
165
      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
      .text
     .align 16
173
     .globl print_users
174
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)
movq %rbx 16(%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
call printf
movq O(%rbx), %rbx
195
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
208 .type print_users, @function
```

```
.size print_users, . - print_users
210
     .align 16
211
     .globl getUser
212
213 getUser:
     .cfi_startproc
214
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)
movq %rsi, %rbp
219
220
221
222
      movq %rdi, %rbx
223 .L105:
     cmpq $0, %rbx
224
      je .L106
225
     leaq 8(%rbx), %rdi
226
     movq %rbp, %rsi
call strcmp
testl %eax, %eax
227
228
229
     je .L107
230
     movq 0(%rbx), %rbx
231
    jmp .L105
232
233 .L106:
234
     xorq %rbx, %rbx
235 .L107:
    movq %rbx, %rax
236
     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
246
     .globl main
247 main:
248
     .cfi_startproc
      subq $536, %rsp
249
      .cfi_adjust_cfa_offset 536
250
251
     leaq 544(%rsp), %rax
     movq %rax, 0(%rsp)
movq %rbx, 8(%rsp)
call setup users
252
253
254
      call
             setup_users
      movq %rax, %rbx
255
     leaq __stringlit_10(%rip), %rdi
movl $0, %eax
256
257
      call printf
258
      leaq
            __stringlit_11(%rip), %rdi
259
      movl $0, %eax
260
261
      call printf
     leaq __stringlit_12(%rip), %rdi
leaq 16(%rsp), %rsi
262
263
     movl $0, %eax
264
call __isoc99_scanf
```

```
leaq 16(%rsp), %rsi
movq %rbx, %rdi
266
267
             getUser
268
      call
      movq %rax, %rbx
269
      cmpq $0, %rbx
270
      jne .L108
271
     leaq __stringlit_13(%rip), %rdi
leaq 16(%rsp), %rsi
272
273
      movl $0, %eax
274
      call printf
xorl %eax, %eax
275
276
277
      jmp .L109
278 .L108:
      leaq __stringlit_14(%rip), %rdi
movl $0, %eax
279
280
      call printf
281
             __stringlit_12(%rip), %rdi
282
      leaq
      leaq 272(%rsp), %rsi
283
284
      movl $0, %eax
      call __isoc99_scanf
leaq 264(%rbx), %rdi
leaq 272(%rsp), %rsi
285
286
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
293
294
      jmp .L109
295
296 .L110:
     leaq
             __stringlit_16(%rip), %rdi
297
      leaq 8(%rbx), %rsi
298
299
      mov1 $0, %eax
      call printf
300
301
      leaq
             __stringlit_9(%rip), %rdi
      movl $0, %eax
302
303
      call printf
     leaq __stringlit_17(%rip), %rdi
leaq 8(%rbx), %rsi
movl $0, %eax
304
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, @function
      .size main, . - main
319
      .section "__compcert_ais_annotations","", @note
320
      .ascii "# file:password.c line:20 function:setup_users\n"
321
322 .byte 7,8
```

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

C.12 password-O3.s

```
# File generated by CompCert 3.7
2 # Command line: password.c -03 -S
    .section .rodata
    .align 1
5 __stringlit_7:
    .ascii "--- USERS ---\012\000"
    .type __stringlit_7, @object
    .size __stringlit_7, . - __stringlit_7
    .section .rodata
9
10
    .align 1
11 __stringlit_6:
   .ascii "passw0rd123\000"
12
13
    .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
    .section .rodata
21
    .align 1
22
23 __stringlit_14:
    .ascii "Password: \000"
24
25
    .type __stringlit_14, @object
    .size __stringlit_14, . - __stringlit_14
26
27
    .section .rodata
    .align 1
28
29 __stringlit_18:
    .ascii "Your balance: $%ld\012\000"
.type __stringlit_18, @object
30
31
    .size __stringlit_18, . - __stringlit_18
32
    .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:
.ascii " %021d. %s\012\000"
.type __stringlit_8, @object
.size __stringlit_8, . - __stringlit_8
```

```
45 .section .rodata
    .align 1
47 __stringlit_1:
    .ascii "admin\000"
48
    .type __stringlit_1, @object
49
    .size __stringlit_1, . - __stringlit_1
50
51
    .section .rodata
    .align 1
52
53 __stringlit_2:
   .ascii "4dm1n_4eva\\000"
54
    .type __stringlit_2, @object
.size __stringlit_2, . - __stringlit_2
55
56
    .section .rodata
57
    .align 1
59 __stringlit_3:
    .ascii "alice\000"
60
    .type __stringlit_3, @object
61
    .size __stringlit_3, . - __stringlit_3
62
63
   .section .rodata
64 .align 1
65 __stringlit_11:
    .ascii "Username: \000"
66
    .type __stringlit_11, @object
67
68
    .size __stringlit_11, . - __stringlit_11
    .section .rodata
69
70
    .align 1
71 __stringlit_5:
    .ascii "abdul\000"
72
    .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
    .align 1
83 __stringlit_12:
84
    .ascii "%255s\000"
     .type __stringlit_12, @object
85
    .size __stringlit_12, . - __stringlit_12
86
87
    .section .rodata
    .align 1
88
89 __stringlit_9:
    .ascii "\012\000"
90
    .type __stringlit_9, @object
91
92
    .size __stringlit_9, . - __stringlit_9
    .section .rodata
93
94
    .align 1
95 __stringlit_15:
    .ascii "ERROR: incorrect password\012\000"
96
97
    .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
__stringlit_16:
     .ascii "Logged in as < %s >!\012\000"
108
     .type __stringlit_16, @object
109
     .size __stringlit_16, . - __stringlit_16
110
111
     .text
112
     .align 16
     .globl setup_users
113
114 setup_users:
115
     .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
121
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
131
132 .L100:
     movq $1000000, %r10
133
     movq %r10, 520(%rbp)
134
     movq $528, %rdi
135
     call malloc
movq %rax, %r12
leaq 8(%r12), %rdi
136
137
138
     leaq __stringlit_3(%rip), %rsi
call strcpy
139
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
            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
176
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
      leaq __string
movl $0, %eax
184
             __stringlit_7(%rip), %rdi
185
      call printf
186
     xorq %rbx, %rbx
187
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
     .globl getUser
212
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)
     call
           setup_users
254
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
265
      call
            __isoc99_scanf
     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
leaq 272(%rsp), %rsi
movl $0, %eax
282
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:
     movq 8(%rsp), %rbx
314
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(\"rbp\") + 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"
```

C.13 deadStoreElimination-O0.s

```
_{\rm 1} # File generated by CompCert 3.7
2 # Command line: deadStoreElimination.c -00 -S
    .align 16
    .globl deadStore
6 deadStore:
    .cfi_startproc
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
    leaq 16(%rsp), %rax
   movq %rax, 0(%rsp)
11
12
   movl $43981, %ecx
13 .L100:
14 nop
15 .L101:
  cmpl %esi, %edi
16
    jle .L102
17
   leal -1(%edi), %edi
18
19
    jmp .L101
20 .L102:
    leal 0(%edi,%esi,1), %eax
21
22
    addq $8, %rsp
    ret
23
    .cfi_endproc
24
    .type deadStore, @function
25
    .size deadStore, . - deadStore
26
27
    .text
    .align 16
28
    .globl main
29
30 main:
    .cfi_startproc
31
32
    subq $8, %rsp
    .cfi_adjust_cfa_offset 8
33
34
    leaq 16(%rsp), %rax
    movq %rax, 0(%rsp)
movl $2, %esi
call deadStore
35
36
37
    xorl %eax, %eax
38
39
    addq $8, %rsp
    ret
40
41
    .cfi_endproc
    .type main, Ofunction
42
    .size main, . - main
43
    .section "__compcert_ais_annotations","", @note
44
    .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
45
    .byte 7,8
46
    .quad .L100
47
ascii " L(reg(\"rcx\")) = high\n"
```

C.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7
_{2} # Command line: deadStoreElimination.c -03 -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:
13 nop
14 .L101:
   cmpl %esi, %edi
15
    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
26
    .text
    .align 16
27
    .globl main
28
29 main:
    .cfi_startproc
30
31
    subq $8, %rsp
    .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
38
    addq $8, %rsp
39
    ret
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"
```

C.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.
```

C.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:23: error: access to volatile variable 'z' for parameter '%e3' is not supported in ais annotations

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

7 errors detected.
```