# OpenTC Newsletter
**January 2008**

From the Open Trusted Computing (OpenTC) research project, sponsored by the European Union.

---

**In this issue:**

---

## Editorial: Verified compartments to protect users against malware

By: Arnd Weber, ITAS, Forschungszentrum Karlsruhe, Germany

Dear reader,

At the beginning of the OpenTC project, many opinions on Trusted Computing (TC) gathered on the Internet in discussion fora and articles were critical of the technology. Some people believed that TC would be used for tracking citizens' activities, locking computers into proprietary configurations, enforcing unfair business models for content or software provision, and even automatically deleting emails and documents without the user's approval. In this context, the project's work package on requirement analysis (WP02) found that a prototype showing TC's potential to protect users could be used to make TC more attractive, as opposed to an application protecting "only" organisations like employers or content providers. Such a prototype would make it possible to demonstrate to the public the broad range of purposes for which TC can be used. A particularly suitable use-case in our opinion is one that demonstrates how TC can be used to protect individuals against financial risks, e.g., security threats in homebanking or online auctions. Given that computer criminals frequently produce malware software to fool users into giving them their security credentials, so-called phishing, strong protection against such attacks would be the objective of a suitable use-case implementation. On the basis of the OpenTC architecture, the design of the prototype would comprise a TC-secured hypervisor with a verified compartment running an isolated browser, to be used for the risky transactions underlined in the scenario (i.e., not for general web surfing) in order to keep it secure. This compartment would run next to a compartment with a legacy operating system and untrusted applications used for casual tasks. We nicknamed this prototype the "PET", an abbreviation of "Private Electronic Transactions", and the consortium implemented a proof-of-concept prototype based on this scenario.

The PET prototype is described in the first article by Stephane Lo Presti (Royal Holloway, University of London), Gianluca Ramunno (Politecnico di Torino), and Dirk Kuhlmann (Hewlett Packard Laboratories, Bristol). The design and decision process which led to creating the PET is described in more detail in Kuhlmann/Weber [1], where the reader can find a review of the various arguments found in the media, a description of the consortium's internal decision process, and the conclusions drawn from this process, as well as information on the OpenTC architecture and PET design. The source code has been made publicly available [2] in order to enable developers, in particular FOSS (Free and Open Source Software) developers, to use it.

In future real-world systems that use the code produced by the OpenTC project, the various mechanisms from the OpenTC architecture should be implemented at a very high level of quality. In the second article in this OpenTC newsletter, we continue our description of the consortium’s approach to improving the source code of the project. In the last issue, Pascal Cuoq from CEA-LIST, Paris, provided an introduction to "Static analysis using Abstract Interpretation", while in this second part of his article he goes on explaining precisely how static analysis works. The consortium’s approach to the trustworthiness of code will continue to be explored in upcoming articles in the next issues of the OpenTC newsletter.

We conclude this newsletter with an announcement of a new research paper published by members of the OpenTC consortium.

References:
- [1] Kuhlmann, Dirk; Weber, Arnd (eds.): Requirements Definition and Specification. OpenTC Project Deliverable D02.2.
http://www.opentc.net/deliverables2006/OTC_D02.2_Requirements_Definition_and_Specification_update.pdf
- [2] Free and Open-Source Software (FOSS) components of the Open Trusted Computing architecture:
http://www.opentc.net/index.php?option=com_content&task=view&id=47&Itemid=65

Contact: arnd.weber at itas.fzk.de

Our thanks go to Richard Brown, Alison Hepper, Dirk Kuhlmann, Stephane Lo Presti and Dirk Weber for their help in preparing this issue.

---

## Private Electronic Transactions - The OpenTC proof-of-concept prototype

By: Stephane Lo Presti, Royal Holloway, University of London, UK; Gianluca Ramunno, Politecnico di Torino, Italy; and Dirk Kuhlmann, Hewlett Packard Laboratories, Bristol, UK

### Introduction

The OpenTC project recently published the first proof-of-concept prototype of its Open Trusted Computing architecture. This work involved several developers from HP (United Kingdom), IBM (Switzerland), Politecnico di Torino (Italy), Ruhr-Universitaet Bochum (Germany), Technische Universitaet Dresden (Germany), IAIK Graz University of Technology (Austria) and SuSE (Germany). The prototype aims to demonstrate some core ideas of the OpenTC approach, combining Trusted Computing (TC) and virtualisation technologies with Free and Open Source Software (FOSS) development. The release implements a concrete application of TC, is available to the general public, and could serve as a working basis for interested developers.

As a proof-of-concept prototype, the implementation is far from comprehensive, and has not been thoroughly tested. It does not include all the necessary components for the OpenTC architecture, nor are the existing ones in any finalised form. Nevertheless, we hope that the proof-of-concept work will give some impression of how TC could be applied in practice, foster discussions about the use of TC and the OpenTC architecture, and encourage contributions from the FOSS communities. With some minor exceptions, all source code is released under the GPLv2 license. It is provided as both a Live CD image (binaries) and as source code (see link at the end of the article). The prototype was developed and tested on HP nx6325 and IBM T60 laptops equipped with Trusted Platform Modules (TPMs) v1.2 and 1 GB RAM.

OpenTC makes use of virtualisation layers - also called Virtual Machine Monitors (VMMs) or hypervisors - and supports two different implementations (i.e., Xen and L4/Fiasco). This layer hosts compartments, also called virtual machines (VMs), domains or tasks, depending on the VMM being used. Some domains host trust services that are available to authorised user compartments. Various system components make use of TPM capabilities, e.g., in order to measure other components they depend on or to prove the system integrity to remote challengers.

The current prototype implements a scenario called Private Electronic Transactions (PET), which aims to improve the trustworthiness of interactions with remote servers. Transactions are simply performed by accessing a web server through a standard web browser running in a dedicated trusted compartment named "domT". In the PET scenario, the server is assumed to host web pages belonging to a bank; however, the setup also applies to other e-commerce services.

The communication setup between browser compartment and server was extended by a protocol for mutual remote attestation tunnelled through an SSL/TLS channel. During the attestation phase, each side assesses the trustworthiness of the other. If this assessment is negative on either side, the SSL/TLS tunnel is closed, preventing further end-to-end communication. If the assessment is positive, end-to-end communication between browser and server is enabled via standard HTTPS tunnelled over SSL/TLS.

## General approach

--------------------

The approach of this scenario relies on four elements:

1) Trusted platform setup
2) Authenticated boot process
3) Registration of the known-good measurement values
4) Trusted communication setup

The following sections give a cursory overview of these steps.

### 1. Trusted Platform Setup

As a first step, the user has to initialise the client platform and to prepare it for subsequent operations.

The setup procedure is performed as soon as the OpenTC system has started up for the first time. First, the user, acting as the platform owner, must "take ownership" of the TPM. This assumes that the TPM is enabled in the BIOS and, if it was already used before, is cleared and re-enabled. More information about this feature of TC can be found at [1].

Next, an Attestation Identity Key (AIK) must be created and certified, and the corresponding identity activated. In a real-world situation, Privacy Certification Authorities (PCAs) that operate a valid TC-enabled Public Key Infrastructure (PKI) would be used during this process. However, for the sake of simplicity, both the requests for and the release of the AIK certificate operations are currently handled locally on the client, using hard-coded passwords. This is done by a software library that comes with hard-wired mock certificates for the authorities involved (e.g., PCA).

## 2. Authenticated boot process

From the initial bootup process up to the start of trusted components, a chain of trust is generated: each component of the chain is measured prior to passing control to it. The component measurements, i.e. their fingerprints through cryptographic hashes of relevant binary and configuration data, are accumulated in Platform Configuration Registers (PCRs) of the TPM.

In order to generate the chain of trust, all components in this chain must be instrumented to perform integrity measurements of their successors in execution. For instance, the BIOS must include a Core Root of Trust for Measurement (CRTM), as defined by the Trusted Computing Group. Further modifications concern the Master Boot Record (MBR) and the boot loader. The latter has to measure the hypervisor, the kernel and initial ram disk images of privileged domains, or files or disk images of trusted compartments. For this purpose, the OpenTC prototype includes "tGRUB", an extended version of the familiar GRand Unified Bootloader (GRUB). The tGRUB boot menu offers a choice between the two different virtualisation layers, XEN and L4/Fiasco.

A second boot menu option concerns the execution mode. It offers a "normal user" mode that comes with a simplified interface and restricts access to management functions, geared towards showing how the user can perform a transaction in a real-life scenario. The "expert user" mode, on the other hand, enables full access to management features, permitting a peek “under the hood”. It allows interactive access and comes with a more complex interface. As a demonstration of the differences between successful and failing verification of integrity measurements, the boot menu also provides a "good domT" and a "rogue domT" option. In the "good" mode, all measurements match their expected values. On the other hand, the "rogue" mode simulates a modified compartment that could have been tampered with by an attack from a malicious program. In this case, at least one PCR contains an unexpected measurement value.

OpenTC’s tGRUB loader is constrained to supporting the so-called Static Core Root of Trust for Measurement (S-CRTM) model. In this case, all security and trust-relevant components must be measured, starting with the BIOS. To demonstrate the new Dynamic Core Root of Trust for Measurement (D-CRTM) approach introduced with TPM v1.2, OpenTC provides the Open Secure LOader (OSLO) boot loader implemented as a standard module for GRUB/tGRUB. OSLO implements the D-CRTM for AMD's CPUs, invoking the SKINIT instruction for re-initialising the platform in a trustworthy manner late at runtime.

## 3. Registration of the Known-Good Measurement Values

The client can now register with a server by uploading the measurements that represent its platform state (that is assumed to be trusted). During this procedure, the user (acting as a bank operator) registers his platform with a bank server. To this end, he uploads the integrity measurements of his platform (i.e., the expected value for the domT compartment), thus authorising the trusted compartment to connect to the bank server. In a realistic scenario, this process would be automated using secure communications and a dedicated registration protocol. Since the prototype does not yet implement this feature, the user instead launches a script (in "normal user" mode) or uses a browser (in the "expert user" mode) to upload a file containing a measurement digest to the front-end proxy running on the bank server. This digest corresponds to the current state of the client platform, i.e., its trusted virtualisation layer and the trusted domain domT. For simplification, the bank server (named "domS") measurements are already present on the client side (in a privileged compartment of the virtualisation layer named Domain-0 or dom0) of the OpenTC system: the PCR metrics defined to correspond to a "trusted" server state are currently hard-wired into the system.

4. Trusted communication setup

Trust status information is exchanged between the client and the server through a pair of proxy services running in a privileged compartment (dom0) on the client and as a front-end in the server compartment (domS). The proxies communicate through an SSL/TLS tunnel that can encapsulate any TCP-based protocol. For the PET scenario, HTTPS is used.

The communication setup is initiated when the browser is started in the trusted compartment (domT) and the link provided in its toolbar clicked on. The connection request is passed to the client proxy that runs in the privileged compartment (dom0). A dedicated component running in this compartment opens an SSL/TLS tunnel to connect to the server-side proxy running in the bank server compartment (domS). The permissibility of this connection must be stated in the policy of the virtualisation layer. Measurements of software components on both sides (client and server, see above) are represented by PCR values of the hardware and software TPM. These values are signed with the AIK acquired in step 3, and communicated to the respective peer system.

Communication between the client and the server will only be enabled if both the client and the server metrics suggest that they booted into (and still run) the expected configuration. This could prove a countermeasure to phishing attacks, preventing the user from following a false-lead URL to connect to a fake bank server. The mechanism might also improve the protection of the bank server against unauthorised connections.

## Platform components and behaviour
------------------------------------------

The authenticated boot process launches the selected virtualisation layer, which is responsible for controlling four dedicated compartments, namely:

- A privileged compartment (called "dom0") that directly accesses the physical platform and includes the drivers for hardware devices. This compartment is also used to perform management operations at the virtualisation layer. In "normal user" mode, this compartment is not visible, whereas the "expert user" mode allows client compartments (see below) to be started manually via scripts. Dom0 is part of the Trusted Computing Base (TCB).

- A compartment for the server side of the demonstrator (called "domS") which executes the banking application and its front-end. It locally simulates a remote server (web server and proxy). For the prototype, this removes the dependency on external communication which would require an additional, separate banking computer. The domain is accessed via the network name "domSbox", and a software TPM emulator is used to perform the integrity-related operations for this domain. The compartment is not visible to the user. It goes without saying that this domain could equally be executed on a remote physical machine, and future versions of the OpenTC prototype will support this.

- Two compartments for the client side of the system. The first compartment (called "domT") solely provides web browsing as its single functionality. It is considered trusted in that its integrity has been measured and the values correspond to a well-known configuration. Measurements, which include the configuration file, the kernel and the virtual disk image with the root file system, are accumulated within PCR[11] in the TPM through the “PCR extend” operation. The second compartment is an untrusted one (called "domU"). It is not measured and is intended for daily use, but specifically not for the PET transaction.

The platform policy ensures that only the trusted compartment (domT) is authorised to use the client proxy (executing in the privileged compartment dom0) for connecting to the bank server (executing in domS). By contrast, the untrusted compartment (domU) cannot connect to the bank server, since no measurements exist that can be used to attest to its trusted state.

Client compartments run a stripped-down Linux system (a Debian-based Damn Small Linux/DSL distribution), while the privileged compartment dom0 runs either a SuSE or a DSL distribution, depending on which version of prototype is used. The server compartment (domS) runs a standard Debian distribution.

The L4-based OpenTC implementation provides a secure GUI service for managing input and output. This GUI is provided as a trusted service started by tGRUB at boot time and running in a dedicated compartment. The top part of the display is reserved to indicate the number of active compartments and the name of the one currently hooked to the display. This section of the screen is considered trusted because it is under the exclusive control of the secure GUI. With the L4 virtualisation layer, a pair of hotkeys are used to switch between compartments.

For the Xen-based OpenTC implementation, the whole screen is under the control of a selected compartment. Each compartment is assigned a fixed hotkey. When a specific hot key associated to a compartment is pressed, the user can be sure that the desired compartment will actually be displayed since the key is under the exclusive control of the privileged compartment dom0.

## Shortcuts, simplifications, and limitations
--------------------------------------------------

Trusted Computing is an emerging technology, and infrastructure support, e.g., for issuing certificates, does not yet exist. The technology is also fairly complex. For example, first-time users are easily confused by the multiplicity of authorisation secrets. For the sake of simplicity, the OpenTC prototype, therefore, uses fixed passwords as authorisation secrets for the TPM and AIK keys (which can, of course, be changed for all operations by editing the configuration scripts in the "expert user" mode).

The prototype attempts to strike a balance here in providing all necessary components (including PKI mechanisms) as part of the distribution. In particular, the server-side mechanisms reside in a dedicated compartment on the same physical hardware that runs the client.

The PCR metrics corresponding to a trusted PKI server state are currently hard-wired into the PKI component running on the client system. The banking compartment domS is not actually measured since it employs a TPM emulator instead of a hardware TPM. Instead, we just communicate the initial PCR values of the software TPM. In a future release of the OpenTC prototype, the server side will run on a different physical platform, and integrity measurements of the server domain will be duly recorded.

In the "rogue domT" scenario, the measurement of the client banking compartment (domT) does not correspond to the one uploaded earlier to the bank server. Consequently, the attestation will fail, and communication with the bank server will be disallowed. Due to implementation specifics, the PCRs mismatch currently has to be simulated in that the values are not generated from actual measurements of the persistent compartment image but only from a different configuration file for domT. A future release will fix this issue.

Furthermore, the current prototype is also limited in that man-in-the-middle or "relay of attestation challenge" attacks have not yet been considered. For more information, please see [2].

## Outlook
---------
Although the current implementation of the PET scenario is relatively straightforward and cuts quite a number of corners, it is a concrete example of the possibilities of coupling TC with virtualisation technology. As the source code of the prototype is publicly available, the OpenTC prototype provides a working basis for any developer interested in implementing TC applications.

About the authors: Stephane Lo Presti is a research assistant working at Royal Holloway, University of London (UK), on mobile trusted computing and dissemination activities of the OpenTC project. Gianluca Ramunno is a research assistant working at the technical university Politecnico di Torino (IT) on security applications and PKI for trusted computing. In the OpenTC project, he coordinated the development of the proof-of-concept prototype related to the PET scenario. Dirk Kuhlmann is a senior research engineer for Hewlett Packard Laboratories in Bristol, UK, where he works as a member of the Trusted Systems Laboratory. He acts as the overall technical leader for the OpenTC project.

Download:
The prototype can be downloaded from the OpenTC website, URL:
http://www.opentc.net/index.php?option=com_content&task=view&id=45&Itemid=63

The source code for the components of the OpenTC system is referenced in the following webpage:
http://www.opentc.net/index.php?option=com_content&task=view&id=47&Itemid=65

References:
- [1] "The Roots of Trust - The RTM and the TPM ", Eimear Gallery. Second Lecture of the Trusted Computing course, MSc of Information Security, Royal Holloway, University of London, 2007.

- [2] Kenneth Goldman, Ronald Perez and Reiner Sailer: Linking remote attestation to secure tunnel endpoints. In: Proceedings of the first ACM workshop on Scalable Trusted Computing (STC '06), pp. 21-24. Alexandria, Virginia, USA, 2006. ACM Press, ISBN: 1-59593-548-7, http://doi.acm.org/10.1145/1179474.1179481.

Contacts: Stephane.Lo-Presti at rhul.ac.uk; Ramunno at polito.it

## Static analysis using Abstract Interpretation (part 2)

By: Pascal Cuoq, Commissariat à l'Energie Atomique (CEA-LIST), Paris, France

This is the second part of the article, the first part of which was published in the September 2007 issue of the OpenTC newsletter, see [1].

The sample program analysis used in the first part of the article showed how static analysis can work at first sight. This second and final part of the article aims to show that it is not always as easy as it seems, using various program examples.

The property of termination of a program

The programming example used in the first part of the article was simple, but not simplistic, because it contained a loop statement. Many research developments in theoretical computer science either have to do with the fact that loop statements are necessary for programming, or try to deal with the difficult questions they automatically raise. One such question is whether a given program is always guaranteed to end and return a result. This property is considered particularly interesting and is called "termination" of the program. However, it was not difficult to check the properties of the loop statement (a 'while' statement) used in the previous programming example. It is easy to see that the condition in the 'while' construct (x >= 2) would eventually become false and cause the loop to stop, meaning that the program would not run forever. Let us now consider the following C example instead:

```
1  for (n = 4 ; ; n += 2)
2  {
3    for (p1 = 2 ; p1 < n ; p1 ++)
4    {
5      p2 = n - p1;
6      if (is_prime(p1) && is_prime(p2))
7        break;
8    }
9    if (p1 == n)
10     break; // sum of two primes not found for n.
11 }
```

The function "is_prime" always terminates and returns "true" when its argument is a prime number. The implementation of this function requires some ingenuity in order to make it efficient for big numbers, but a simple implementation that repeatedly divides its argument "m" by every number smaller than m and returns "true" only if none was a divisor, would be enough for our purposes. Programming this function in C is left as an exercise to the reader.

The above program enumerates for each value of the variable "n" the even integers greater than 2 (4, 6, 8, 10, etc.). For each of these successive values of n, the program first (lines 3 to 8) enumerates the various ways of splitting n into the sum of two integers p1 and p2. If this split is such that both p1 and p2 are prime numbers, the program jumps to the next even value for n (line 7). If no pair of numbers p1 and p2 satisfying this property, i.e., n == p1 + p2 and p1 and p2 both prime, is found, the program stops (line 10).

This program is very simple, but in fact conceals a very difficult theoretical problem: deciding whether the program terminates or not is equivalent to proving or disproving the mathematical "Goldbach's conjecture", which at the time of this writing remains an open problem in number theory and has been so for more than 250 years. In practice, this means that it is in the worst case impossible to know whether a program will stop under any circumstances. Although the previous program does not use any complicated programming features, the question of whether it terminates or not is theoretically even harder to solve than Fermat's last theorem.

## Loop invariants

Even when the termination of a loop statement is easier to establish than in the previous example, the correct behaviour of the loop may rely on complicated hidden invariants. "Loop invariant" is the name given to a property that holds throughout the execution of a loop. This situation should be familiar to any developer who has ever had to modify code that he did not write himself: the program works for reasons that are not obvious in the code, for example because the value of a certain variable "p" can never be negative. The invariants are not always explicitly written in the program (experienced developers write them in comments or in assertions, but even with experience, it is not easy to think of all the invariants that should be written down). Taking the example of the variable p that is always positive, the invariant holds because one can check on each program statement that a negative value is never assigned to p. The difficulty lies in the fact that it is not easy to reconstruct the invariant by simply looking at the source code. Similarly to the situation of the termination of loop statements, it can in fact be arbitrarily difficult.

Analysis of actual source code does not always require proving or disproving Goldbach's conjecture, however. The above example is only intended to show the kind of difficulties that make the problem impossible to solve in all generality. In real life and for ordinary programs, invariants are often simple enough to be discovered automatically.

## Example of a program that is difficult to analyse

Besides loop statements, real programs also contain many statements that are difficult to handle because they do not fit well into the theoretical frameworks used for program analysis. The particular set of statements that causes problems depends on the analysis framework, but it typically contains some or all of the following: casts, union types, architecture-dependent statements, pointers and aliasing, integer overflows, floating-point arithmetic. The following example illustrates several of the above issues:

```
void shift_string (char *s, unsigned int src_offs, unsigned int size)
{
  unsigned int dst_offs = src_offs - 4;
  while (size > 0)
  {
    * (int *) (s + dst_offs) = * (int *) (s + src_offs);
    size -= 4;
    dst_offs += 4;
    src_offs += 4;
  }
}
```

This function makes particular assumptions about the computing architecture and the compilation model in order to shift a specified part of a string "s" (starting at "src_offs") by four characters. The function also assumes that the variable "size" is a multiple of 4, and that the parameters passed to it do not make it access memory out of bounds or in a misaligned way. For the purpose of the program analysis, we assume that these assumptions are well understood by the developer, and that the bug we are looking for (if any) does not lie in these assumptions. Proving that the above function satisfies its informal specification (shifting part of the string "s" by four characters) is difficult. Besides all the low-level statements that the analyser needs to understand (e.g., the conversion from "char*" to "int*"), there are several hidden invariants that ensure that the function works as intended.

To perform the analysis of this program, let us focus first on one of these invariants: at the beginning of the body of the loop (line 6), the current value of variable "dst_offs" is always equal to "src_offs-4".

It is necessary to establish this invariant in order to prove that the function "shift_string" operates as it is supposed to. For a 32-bit architecture, the property implies, in particular, that there is no overlap between the four bytes read at the address "s+src_offs" and the four bytes written at "s+dst_offs" (line 6).

The equality "dst_offs = src_offs - 4" can be automatically discovered by relational abstract interpretation. The idea behind relational abstract interpretation is to compute information about the relationships that are guaranteed to hold between the values of several variables, in addition to sets of possible values for individual variables. Sets of possible variable values are propagated from instruction to instruction in the same fashion that intervals were propagated in the first part of the article. But "values" here means possible values for the tuple of variables (size, dst_offs, src_offs). In the first part of the article, we restricted the "sets" of values to intervals. Here, we also need to make a choice regarding the representation of these sets of possible tuples, so that they can be computed in a reasonable amount of time and memory. There is a wide selection of options when choosing a representation for sets of tuples, but we are only going to consider linear relations between two variables (e.g., y=a*x+b), because this representation is space-efficient and simplifies the computations. Without precise knowledge of the values of the arguments "src_offs" and "size", we know that values that do not satisfy the above equality are impossible for the pair (src_offs, dst_offs) after executing line 3. In fact, the values that are possible for the pair can be represented as a linear relation between these two variables: src_offs = dst_offs + 4.

Given these values for the pair (src_offs, dst_offs), the condition of the 'while' -loop can both be true or false, so we have to assume that this state may propagate to both the inside of the loop at line 6 and the exit of the loop (line 11), which is also the exit of the function.

We will assume that the instruction on line 6 does not affect the values of variables src_offs and dst_offs (this would theoretically require proof that the memory access "*(int *)(s + dst_offs)" is not out-of-bounds). Line 7 does not change the relation between src_offs and dst_offs either.

The instruction on line 8 has an effect on the value of dst_offs and, therefore, on the relation between src_offs and dst_offs. Fortunately, the effect of this instruction on the relation between src_offs and dst_offs can be computed. Better yet, the resulting state can be represented as another linear relation, "src_offs = dst_offs".

The instruction on line 9 has an effect on the relation between src_offs and dst_offs, too. Since this assignment increments the value of src_offs by 4, its effect on a state where "src_offs = dst_offs" is to produce a state where "src_offs = dst_offs + 4" (since the new src_offs is the same as the old src_offs plus 4).

The state of the variables at the end of line 9 cannot be used to decide whether the condition of the 'while' loop (size > 0) is true, so this state must be propagated again to both the exit of the function and the beginning of the body of the loop. This state, which is identical to the state that occurred after executing line 3, has in fact already been propagated to the exit of the function and the beginning of the body of the loop. This means that this state is a fixpoint of the program (i.e., a state that leads to itself when the body of the loop is applied to it), and that the program analysis is finished. The analysis has determined that the property held, because the fixpoint found represents the invariant we wanted to prove.

The choice of the kind of sets that are used determines the types of programs that can be analysed accurately. In the above example, linear relations are a good choice because:

1. Some variables in the program are in a linear relation to each other, and
2. The instructions executed by the program (particularly, at lines 8 and 9) transform a linear relation into another linear relation, which can also be represented.

What else is necessary to analyse this program?

The other important requirement in order to be able to analyse this program is a memory model. Actually running the program would have allocated the string "s" to somewhere in the memory, and it would be possible to analyse the function "shift_string" with respect to this particular memory allocation (in this case, a particular choice of address where "s" is stored). But this is not satisfactory because the goal of static analysis is to check that there are no errors for all possible executions, which means not only for all inputs, but also for all addresses at which the system decides to allocate string "s". If two strings are allocated in the program, one goal of the verification process would be to check that one string is not modified via an out-of-bounds access to the other. This error may manifest itself for some allocations while it may not be visible for others, making it hard to find or debug solely by testing the program.

For this reason, the static analysis of a program proceeds with respect to an abstract memory model. An abstract memory model allows manipulation of a synthetic representation of all possible choices for the layout of a program's data in memory. As with the choices that must be made with respect to the representation of relationships between variables, there are compromises in the design of an abstract memory model. It goes beyond the scope of this article to delve into all the various possibilities. The purpose of an abstract memory model, on the other hand, is simple: the abstract memory model should make it possible to analyse a program without having to choose a particular memory allocation. Instead, when the memory model is appropriately devised, a program property proved in the abstract memory model holds for all the possible allocations that can occur at run-time. The adjective "correct", which was applied to analyses in the first part of the article, is applied to abstract memory models which have this property. Of course, one needs a correct memory model in order to build a correct analyser.

In the previous example, it is the task of the memory model to ensure that the instruction on line 6 does not have any effect on the relation between src_offs and dst_offs. Because the C programming language is a low-level language, an out-of-bounds access does not necessarily translate into a run-time error. Depending how the program's data is laid out in memory, the memory access could also silently change the value of another arbitrary variable. A reasonable memory model should exclude this possibility, and instead emit a warning concerning the out-of-bounds access, if one might exist.

To analyse this example precisely, the memory model also needs to describe the fact that an "int" variable takes up the same space as four "char" variables (on most current computers, integers are stored on 32 bits while characters require 8 bits), and what exactly happens when a "pointer to char" is cast into a "pointer to int", and subsequently accessed.

The Frama-C toolbox

The Laboratory for Software Safety at CEA LIST (Paris, France) is developing Frama-C, a toolbox for the static analysis of C programs. This toolbox is designed to enable different analysis techniques and features to be plugged in, and used with each other. One of the distinguishing features of the Frama-C toolbox is the specification of the memory model, which is especially adapted to the analysis of embedded (low-level) C code. Frama-C has been used to analyse parts of the Xen hypervisor in the context of the OpenTC project.

Editor's note: An article about the analysis of the XEN hypervisor will be published in a future issue of the OpenTC newsletter.

About the author: Pascal Cuoq has been playing with computers since he was 6. He is currently doing so at CEA-LIST, within the Software Safety Laboratory, where he is one of the architects of Frama-C, an assistance tool to improve confidence in critical C code.

Contact: pascal.cuoq at cea.fr

Reference:
[1] Static analysis using Abstract Interpretation. In: OpenTC newsletter of September 2007: http://www.opentc.net/publications/OpenTC_Newsletter_02.html#link3

---

**Recent OpenTC publications**

Since the publication of the last newsletter, the OpenTC project partners have produced one scientific publication:

- Murray, D.; Milos, G.; Hand, S.: Improving Xen security through disaggregation. In Proceedings of the 4th international conference on Virtual Execution Environments (VEE 2008), March 2008 (to appear).

---

Edited by the Institute for Technology Assessment and Systems Analysis, Forschungszentrum Karlsruhe, Germany, on behalf of the OpenTC research project consortium, in co-operation with all partners.
Editor: Arnd Weber, Forschungszentrum Karlsruhe GmbH, ITAS, Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Telephone: + 49 7247 82 3737.
Contact: editor at opentc.net

The OpenTC-project is a research project supported by the European Commission, project IST-027635. Its 23 partners are: Technikon Forschungs- und Planungsgesellschaft mbH (project coordination, AT); Hewlett-Packard Ltd (technical leader, UK); AMD Saxony LLC & Co. KG (DE); Budapest University of Technology and Economics (HU); Commissariat à l'Energie Atomique – LIST (FR); COMNEON GmbH (DE); Forschungszentrum Karlsruhe GmbH – ITAS (DE); Horst Goertz Institute for IT Security, Ruhr-Universitaet Bochum (DE); IBM Research GmbH (CH); Infineon Technologies AG (DE); INTEK Closed Joint Stock Company (RU); ISECOM (ES); Katholieke Universiteit Leuven (BE); Politecnico di Torino (IT); Portakal Teknoloji (TR); Royal Holloway, University of London (UK); SUSE Linux Products GmbH (DE); Technische Universitaet Dresden (DE); Technische Universitaet Graz (AT); Technische Universitaet Muenchen (DE); Technical University of Sofia (BR); TUBITAK – UEKAE (TR); and University of Cambridge (UK).

For more information about the project, see: http://www.opentc.net

Feedback to the consortium: http://www.opentc.net/feedback

Archive of newsletters: http://www.opentc.net/newsletter

Subscription: To subscribe or unsubscribe to the newsletter, write an email to <subscribe at opentc.net> or <unsubscribe at opentc.net>.