Building Secure Resources to Ensure Safe Computations in Distributed and Potentially Corrupted Environments

Sebastien Varrette[1,3], Guillaume Duc[2], Ronan Keryell[2], Jean-Louis Roch[3], Franck Leprevost[1]

[1] University of Luxembourg, LACS Laboratory, Luxembourg
[2] SAFESCALE Project, Department of Computer Science, ENSTBr, Plouzané, France
[3] MOAIS/SAFESCALE Project, LIG-IMAG Laboratory, Grenoble, France

Abstract. Intensive parallel computing in grid environments are subject to various concerns, in particular in terms of security and fault-tolerance. In [1, 2], the authors present a robust and secure architecture able to deal with intensive parallel computing in such environment where resources could be corrupted. The corruption either comes from hardware issues (such as network disconnection) or from malicious act in order to alter the computation and consequently its result. The proposed approach uses dataflow graph to represent a parallel execution and to provide efficient result-checking mechanisms to certify the results of an execution. The architecture is based on a limited number of safe resources that host the checkpoint server (used to store the graph) and the verifiers which could securely re-execute piece of tasks in a trusted way. It remains to detail the effective construction of these resources and this is the purpose of this article. While this issue is generally treated with software solutions, we combine both software and hardware approaches to build strongly secure resources.

1 Introduction

Todays needs for intensive parallel applications require global computing platforms composed of scattered resources that are interconnected through the Internet. Such platforms, called *grids* in [3], are more and more used since the 90's. Two main categories can be distinguished:

1. the "*Desktop Grids*" [4] that typically steal idle cycles of desktop PCs and workstations through the Internet to compute parts of a huge problem. Such topology is illustrated on figure 1;
2. the "*Computing Grids*" which rather gather one or more dedicated clusters as depicted on figure 2. A *cluster* connects several computers through a local network in order to provide a coherent set able to deal with parallel computations, network load balancing, fault-tolerance, etc.

A new sub-classification of computing grids has been introduced in [5], based on the relations between the administrators of each of the involved clusters. It subdivided computing grids in three classes:

1. multi-clusters that bind together several nearby sites administrated by a single person and that manages around 10^2 users;
2. cluster grids that merge several scattered sites through the Internet. Each site is administrated by different persons, yet the administrative domains are sufficiently open to enable the settlement of conventions between the sites (for

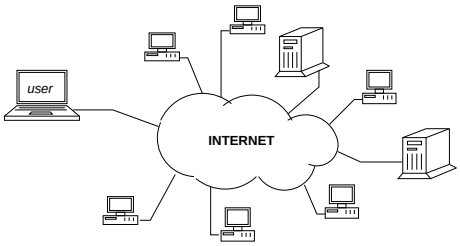

Fig. 1. Desktop grid topology.

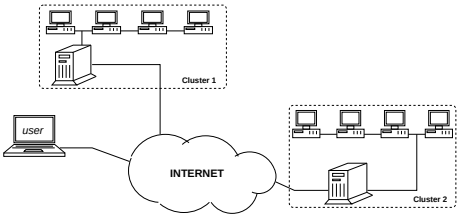

Fig. 2. Computing grid topology: aggregation of clusters.

instance when resolving the host name of the nodes, or for the choice of a common authentication system). Such a topology manages around 10^3 users and corresponds to the architecture of the Grid5000 project[1];

3. finally, the grid in the sense of Globus [6] that manages a huge number of users in sites administratively closed.

Terms introduced in this sub-classification will be used in the rest of this paper: depending on the size of the grid, some mechanisms are more or less adapted. Grid environments raise various concerns in terms of security and data privacy. In particular:

- users and resources should be authenticated;
- only authorized users should be allowed to access and use only the resources of the grid allocated for their own purpose;
- communications should be ciphered to ensure privacy but also integrity;
- data should be securely stored;
- the system should remain operative even in case of failure of some grid components or disconnections which are relatively frequent events. In other words, the integrity of the execution should be guaranteed;
- the resources of the grid should be protected from malicious code;
- the infrastructure should not fail even if some parts of the grid are under control of a pirate or even an evil system administrator, since on quite large grids it is no longer possible to know neither to trust every remote system administrator or computer owner.

Other constraints could intervene depending on the type of grid. Our talk is mainly oriented toward open-source solutions in computing grids based on Unix/Linux operating system which constitute major actors in this field[2]. In the following, section 2 details basic mecanisms that could be used to address the issues mentioned before. Section 3 demonstrates the limits of the classical approaches and explains why efficient result-checking techniques are required. In [1, 2], the authors proposed a novel approach based on data-flow graphs to represent a parallel execution with verifiers. While the graph is stored on a secure checkpoint server, verifiers are used to re-execute some tasks in a trusted way. Therefore, the resources that host either the checkpoint server or the verifiers should be located in a strongly secure area (authors of [2] refers to this area as the "safe resources domain"). One of the contribution of this article, in addition to the analysis of software solutions to address the

[1] The Grid5000 project aims at building a highly reconfigurable, controlable and monitorable experimental Grid platform gathering 9 sites geographically distributed in France and featuring a total of 5000 CPUs - `https://www.grid5000.fr`.

[2] For example, IBM, which is the most represented manufacturer in the list of the 500 most powerful machines in the world (`http://www.top500.org`), claims 161 Linux clusters in this list. That is three quarters of the IBM systems and near a third of all manufacturers.

security constraints in grids, is the effective construction of this area, using both a software and hardware approach. It will be developped in section 4.

2 Basic Security Mechanisms in Grids

Basic security constraints have to be addressed in grids and therefore, basic security features should be installed before any attempt to harden the security of some resources. This section details these constraints and the software solutions that could be used to solve these issues.

2.1 Authentication and Access Control Mechanisms

Designing a robust authentication system in distributed environments has been extensively studied [5, 7, 3]. Efficient solutions depend on the grid topology:

1. In multi-clusters environments, a Kerberos [8] approach should be privileged. This is an authentication system developed at the MIT based on tickets, authenticators and a trusted third party called KDC (for Key Distribution Center). Similar approaches like Kryptoknight [9] could also be considered as the use of Message Authentication Code makes it possible to reduce the size and the number of the messages exchanged during the protocol.
2. As for grids of clusters, the authors of [5] demonstrate an adapted and efficient solution based on LDAP servers that broadcast authentication information. In terms of security, LDAP provides various guarantees thanks to the integration of cipher and authentication standard mechanisms (SSL/TLS, SASL) coupled with Access Control Lists. All these mechanisms enable an efficient protection of transactions and access to the data incorporated in the LDAP directory. The proposed solution is currently used as the authentication system of Grid5000.
3. Finally, for grids of bigger size, the Globus [6] middleware is probably the most adapted. Globus relies on a PKI (Public Key Infrastructure) to establish certification chains between the entities of the grid (either users or services).

Access Control mechanisms prevent unauthorized access to data or resources. Two fundamental types of Access Control could be distinguished: Discretionary Access Control (DAC) and Mandatory Access Control (MAC). DAC permits the granting and revoking of access control privileges to be left to the discretion of the individual users. Unix/Linux file-system uses DAC through the concept of users and groups. This permits users to entirely determine the access granted to their resources, which means that they can (through accident or malice) give access to unauthorised users. MAC extends the previous approach by allowing administrators to enforce additional security for all subjects (e.g. processes or sockets) and objects (e.g. sockets, file system objects etc...). MAC secures information by assigning sensitivity labels on information and comparing this to the level of sensitivity a user is operating at.

If DAC is generally sufficient, we advice the use of Operating Systems that support MAC such as TrustBSD, SUSE or SELinux (this stands for Security-Enhanced Linux - an NSA project that recently added a MAC architecture to the Linux kernel). Debian and Redhat systems also incorporate modules to support MAC.

2.2 Isolating Mechanisms

Sandboxing is a popular technique for creating confined execution environments, which could be used to run untrusted programs. A sandbox limits the level of access the executed program have (a simple example is the `chroot` or `jail` system call that limits the file system hierarchy a process could access). Sandbox environments

could look like a complete operating environment or could only provide a minimum level of isolation (to a few system calls for instance). Among sandboxing techniques, one can distinguish two main approaches:

1. Virtual Machines (VMs) can be used to provide runtime sandboxes to enhance users level of confidence in the resources of the grid. This can either apply to a specific execution of a program written in a given language (we refer then to Programming Language Virtual Machine as the JVM for Java) or to a complete OS (one speaks in that case of virtualization). In the latter domain and in the particular interest of grids, we can mention Xen[3], an efficient virtual machine monitor. Xen can securely execute multiple virtual machines, each running its own OS, on a single physical system with close-to-native performances;
2. The applications could contain a sandbox mechanism within themselves or a system to protect themselves from the outside. Proof-carrying code (PCC) [10] is such a technique used for safe execution of untrusted code. The basic idea is to require the code producer to generate a formal proof that the code meets the safety requirements set by the code receiver. A main issue with this approach concerns the decrease in the performances and the lack of generalization.

Between the different parts of a grid, the isolation is also extended at the network level through firewalling to restrict the packets transiting on the network. This can be achieved, for example on Linux, by using Netfilter and its interface `iptables`.

Even if software and hardware mechanisms are used to isolate user processes from each other, it is difficult to avoid all the side-effect channels that can be used by a process to spy another process (disk activity, cache eviction between different threads...) and potentially extract some secrets.

In addition, there is an asymetry in the trust for all the previous techniques: the aim is to protect the computing infrastructure from the program execution and there is no way for the users to have a certified and protected execution. Military or industrial clients therefore prefers to have their own computing infrastructure (to protect their own code and perhaps even more data from theft, analysis or modification) rather than using public grids. We hope that the mechanisms that are now presented could modify this behaviour.

Software obfuscation is a software-only partial answer to software protection and data protection [11]. The code is transformed at the source or binary level to render more complex its readability and also change the data coding. Of course, often this slows down the execution and it is not impossible for someone very motivated to reverse-engineer the obfuscating process since the code is available for execution and can be exercised at will.

Another way is to transform an algorithm that produces data from input data into an isomorphous one that acts on ciphered input and produces ciphered output. It is also possible to add some authentication mechanisms in it to certify the computation done. For some simple algorithms there exists such efficient isomorphous algorithms but, unfortunately, this seducing approach has an untractable complexity for real life programs [12]. Since basic computations (as floating point operations that are highly optimized in modern processors), are transformed in elementary operations executed to emulate some enciphered circuits, the efficiency expected on grids are no longer here.

2.3 Monitoring and Intrusion Detection Systems

Monitoring the operations done on the grid is required to detect abnormal activities. R-GMA [13] has been developed in the framework of EGEE[4] (Enabling Grids for

[3] `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`
[4] `http://www.eu-egee.org/`

E-science in Europe) to provides a service for information, monitoring and logging in a distributed computing environment. It is suitable both for information about the grid (primarily to find out what services are available at any time and then to find details about those services) and for application monitoring in larger grids.

For multi-cluster and cluster grids, jobs and resources state can be sketched using OAR [14]. In such topologies, Nagios and Ganglia[5] can be used for network monitoring. Yet, as regards network monitoring, one could inspire from the infrastructure used in acacademic networks such as RENATER that exploits a set of passive and active measurement tools.

On the one hand, passive measurement consists in traffic analysis by capture. This can be done by packet monitoring or by using the standardized Simple Network Management Protocol (SNMP) to collect information (such as router CPU utilization, link loss, etc.) from MIB (Management Information Base). In RENATER, this is done using the open-source tools MRTG[6] and Cricket[7]. It typically enables the detection of Denial of Service attacks. A complementary approach is stream-oriented to analyse the packets by port, protocol, etc. In that case, NeTraMet or Cisco IOS NetFlow can be used. It makes it possible to detect virus which generally use a particular port for their diffusion.

On the other hand, active measurement injects predefined packets to analyse the efficiency of point-to-point communications (availability, route, packet delay, etc.). Basic active measurement tools such as `ping` and `traceroute` are common and many measurement infrastructures based on them are deployed. Among the various tools available, we encourage the use of Netmeter[8].

3 Remaining Threats and Result-Checking

The precedent section provided basic software mechanisms that have to be used to address basic security constraints inherent to grids use. We now focus on attacks that remain possible.

3.1 Remaining Threats

DDoS Distributed-Deny-of-Service attacks use large networks of Internet-connected systems, which send junk traffic to a victim network or server, crashing the device or slowing it enough to cause a denial of service. In our case, the front-end of each cluster are the main target of those attacks. DDoS are generally done using worms, such as `MyDoom` and/or, as experimented recently, using DNS Amplification [15].

Trojan Horse It is a malicious program that is disguised as or embedded within legitimate software. Trojan horse programs cannot operate autonomously, in contrast to some other types of malware, like viruses or worms. In practice, Trojan Horses often contain spying or backdoor functions that allow a computer to be remotely controlled independently from the owner. The effects on a grid varies: the attacker could send new jobs, alter the execution of jobs on the corrupted node and/or infect the other nodes. The two latter cases appears in the framework of the Seti@Home project [16]. We also mention Sub7[9] as an example of a more intrusive (end destructive) trojan. In essence, trojan horse programs could only be detected

[5] `http://www.nagios.org/` and `http://ganglia.sourceforge.net/`
[6] `http://people.ee.ethz.ch/~oetiker/webtools/mrtg/`
[7] `http://cricket.sourceforge.net/`
[8] `http://www.ccaba.upc.es/netmeter/`
[9] `http://www.hackpr.net/~sub7/`

by monitoring abnormal network activities. In that case, they could be detected using the tools mentioned in §2.3. Otherwise, the detection of a falsification could be done at runtime using the mechanisms briefly introduced in §3.2.

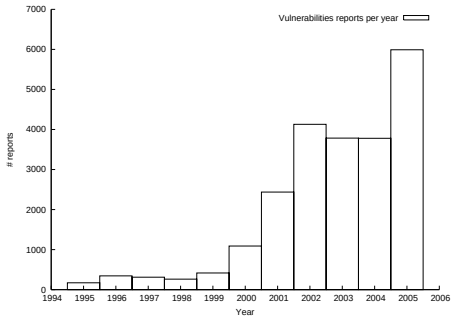

Fig. 3. Vulnerabilities reports per year.

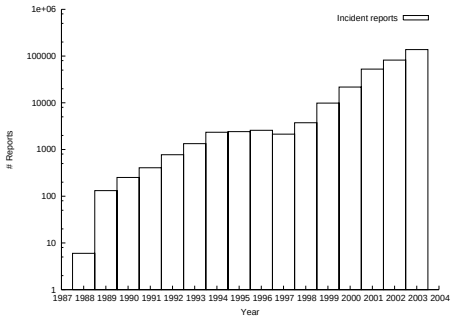

Fig. 4. Incident reports per year.

Applicative Vulnerabilities Statistics regarding the evolution of vulnerabilities and incident reports are presented on figures 3 and 4 (source: CERT[10]). One can notice that the number of reports is exponentially increasing in such a way that given the widespread use of automated attack tools, attacks against Internet-connected systems have become so commonplace that the CERT stopped to count the number of incidents in 2004. Exploitation of vulnerabilities leads in general to remote control with the consequences mentioned in the previous paragraph. Vulnerabilities in the softwares installed on the grid are unavoidable, and can be exploited in a more or less important time window. To limit them, there should exist a secure tool to update nodes environment and even reinstall them in a stable environment. For example, `kadeploy`[11] is such a tool used in Grid5000.

3.2 Ensuring Computation Resilience

Resilience in grid execution is a prerequisite that should be embedded in the application: at this scale, component failures, disconnections or results modifications are part of operations, and applications have to deal directly with repeated failures during program runs. In [1, 2], the authors present a robust and secure architecture able to deal with intensive parallel computing in such environment where resources could be corrupted. The corruption could be caused by DDoS attacks, virus or trojan horses, as expounded in section 3.1or even a DoS by a local administrator. In the latter case, the attack could be undetected by the monitoring tools presented in section 2.3. That's where this infrastructure is particularly interesting as it allows the detection of attacks that tamper the results of the tasks executed by efficient result-checking algorithms. The proposed approach uses a data-flow graph to represent a parallel execution This graph is both used to ensure resilience to crash fault on nodes and to provide efficient result-checking mechanisms to certify the results of an execution. The architecture is based on a limited number of safe resources that host the checkpoint server (used to store the graph) and the verifiers which could securely re-execute piece of tasks in a trusted way. The infrastructure is illustrated on figure 5.

[10] `http://www.cert.org/stats/`
[11] `http://www-id.imag.fr/Logiciels/kadeploy/`

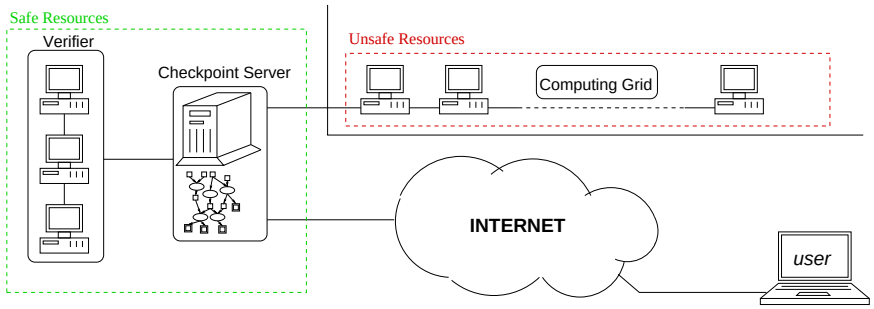

Fig. 5. Resources hierarchy and mandatory components for portable fault-tolerance and error-checking algorithms.

It remains to detail the effective construction of these resources. This is the purpose of the next section.

4 Building Strongly Secured Resources

4.1 Software Components

Sections 2 and 3 have been relatively developed as most of the solutions proposed in these sections will be used as software components for the resources we want to make strongly secured. We therefore construct the secure area with the following software components:

- Secure Authentication: using the most appropriate tools presented in § 2.1;
- OS: prefer TrustBSD that also manages package dependencies for update and various security mechanisms implemented in the kernel;
- Remote Access: only required port should be open (typically the SSH port) and root login should be disabled;
- Secured Storage: in [2], the authors mention the use of CryptStore architecture to ensure a secure storage of data. This approach is also advised here for storage servers;
- System state: Tripwire[12] is a security and data integrity tool useful for monitoring and alerting on specific file change(s) on the systems. TripWire first scans the file systems and computes digital signatures for the files therein. Later these signatures can be used to check those files for any changes. Signatures are deported on a secure storage server.
- Services: the minimum of services should be installed and run (typically, only a job scheduler and the tripwire daemon, without any X server installation).
- Network Monitoring: in the sub-network that constitutes the secure area, the tools presented in §2.3 in should be activated.

In addition to those components, an hardware approach is required to make the host strongly secured. The following section expounds the proposed approach.

4.2 Hardware Components

There is still a big issue with a full software approach: the operating system on the computing nodes have the full control of the hardware and the software running on the nodes. It is very useful to build very complex and powerful computing environment but it is also very dangerous for foreign users of these computing nodes.

[12] `http://tripwire.sourceforge.net/`

A computing node can easily discard a foreign process with all of its data if it seems to be malicious or exploits too many resources. In addition, if we guess that the operating system environment is bug free, foreign process needs to be completely confident in the local software environment: this one can discard the process, the data, read the program and the data, modify both, execute the program step-by-step, and so on. If DoS are unavoidable (the local administrator could decide to switch off the power supply of the computer anyway), computers should have mechanism to allow detection by the running process that a kind of DoS occurred.

In the real life, operating systems and distributed computing environment are huge pieces of software and it is unavoidable to have many bugs in them. Thus, adding pieces of hardware to protect some processes from other parts running out of their rails is quite interesting.

During the last few years, several hardware architectures (such as XOM [17–19], AEGIS [20, 21] and CRYPTOPAGE [22–24]) have been proposed to provide computer applications with a secure computing environment. This kind of secure and trusted computing receives more and more attention in the research and industrial community for different reasons that range from digital right management to secure distributed computing we are interested in here. These architectures use memory encryption and memory integrity checking to guarantee that an attacker cannot disturb the operation of a secure process, or can only obtain as little information as possible about the code or the data manipulated by this process. These secure architectures try to prevent, or at least detect, physical attacks against the components of a computer (for example, the Microsoft X-BOX video game console was attacked in [25] by sniffing the bus of the processor with a logical analyzer) or logical attacks (for example, the administrator of the machine tries to steal or modify the code or the data of an application).

Some secrets can also leak out through the address bus of the processor (an attacker can monitor the control flow graph of a running program and infer algorithms or ciphering keys for example), some approaches try to cipher the address bus more [26] or less [23]. Of course, a foreign process needs to ultimately trust the manufacturer of the processor which could have some wire-tapping or key-escrow features in it, or more probably hardware bugs too.

In our globally secure grid environment, tamper-proof hardware elements can be used to securely run some parts of the computation if we have enough of those trusted elements. As seen in section 3.2, if we cannot trust some parts of the nodes, i.e. we do not have enough trusted elements, we need to probabilistically run again some parts of the computation on some secure nodes that are the trusted element. If there is no trusted nodes at all, a grid user must choose to run the verification parts of the remote computation on her/his own nodes she/he is confident of.

5 Conclusion

Even if security in grid infrastructures is a major research area for a decade, the fact that resources that compose the grid cannot be fully trusted or can be corrupted prevents a wide acceptance as a cheaper computation for high-value applications.

The corruption either comes from hardware issues (such as network disconnection) or from malicious act (using malwares and software vulnerabilities) in order to alter the computation and consequently its result.

The challenge is then to ensure a correct and safe computation of a program despite potential corruptions of the resources that we have described in this article.

The solutions applies at different levels, beginning with users and resources authentication and ending with efficient result-checking algorithms.

Those solutions have been expounded through this article together with a global computing plateform able to address the security issues that could intervene during the execution of a parallel program.

We have first described some good practice to be used in the grid deployment, authentications and control but also in system and network audit and monitoring, as in any distributed computing environment.

Then, to certify the execution of an application on a set of untrusted nodes, we have introduced an original approach based on a combination of software and hardware techniques.

The software part uses some stochastic verifications by running again, on some trusted verification servers, some parts of the global computation chosen by studying the data-flow graph of the application. Since verifications can also be tampered, these tasks need to be run on different computers in different entities to have forgeries likely detected.

But by using a limited number of strongly secure hardware resources, the sensible and time-consuming task of verification can be executed on some remote tamper-proof nodes with no fear about an attacker changing these results.

According to the number of trusted resources available in the grid, our method needs more or less redundant verifications.

References

1. Krings, A., Roch, J.L., Jafar, S., Varrette, S.: A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments. In Verlag, S., ed.: Proceedings of the European Grid Conference (EGC2005). LNCS 3470, Amsterdam, Netherlands, LNCS, Springer Verlag (2005)
2. Varrette, S., Roch, J.L., Montagnat, J., Seitz, L., Pierson, J.M., Leprévost, F.: Safe Distributed Architecture for Image-based Computer Assisted Diagnosis. In: IEEE 1st International Workshop on Health Pervasive Systems (HPS'06), Lyon, France (2006)
3. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. International J. of Supercomputer Applications and High Performance Computing **11** (1997) 115–128
4. Fedak, G., Germain, C., Néri, V., Cappello, F.: Xtremweb: A generic global computing system. In: IEEE CCGrid2001 (Global Computing on Personal Devices). (2001)
5. Varrette, S., Georget, S., Montagnat, J., Roch, J.L., Leprevost, F.: Distributed Authentication in GRID5000. In Meersman, R., al., eds.: LNCS OnTheMove Federated Conferences - Workshop "Grid Computing and its Application to Data Analysis (GADA'05)". LNCS 3762, Agia Napa, Cyprus, Springer Verlag (2005) 314–326
6. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S.: A Security Architecture for Computational Grids. In: Fifth ACM Conference on Computer and Communications Security Conference, San Francisco, California (1998) 83–92
7. Dagorn, N., Bernard, N., Varrette, S.: Practical Authentication in Distributed Environments. In IEEE, ed.: IEEE International Computer Systems and Information Technology Conference (ICSIT'05), Sheraton Hotel, Alger (2005) Still waiting for precisions on proceedings.
8. Neuman, C., Ts'o, T.: Kerberos : An authentication service for computer networks. IEEE Communications Magazine **32** (1994) 33–38 `http://gost.isi.edu/publications/kerberos-neuman-tso.html`.
9. Molva, R., Tsudik, G., Van Herreweghen, E., Zatti, S.: Kryptoknight authentication and key distribution system. In: Proceedings of European Symposium on Research in Computer Security (ESORICS), Toulouse, France (1992) 155–174
10. Necula, G.C., Lee, P.: Proof-carrying code. Technical Report CMU-CS-96-165, School of Computer Science, Pittsburg (1996)
11. Collberg, C.S., Thomborson, C.: Watermarking, tamper-proofing, and obfuscation - tools for software protection. In: IEEE Transactions on Software Engineering. Volume 28. (2002) 735–746

12. Loureiro, S., Bussard, L., Roudier, Y.: Extending tamper-proof hardware security to untrusted execution environments. In: CARDIS. (2002) 111–124
13. Cooke, A.W., al.: The Relational Grid Monitoring Architecture: Mediating Information about the Grid. J. Grid Comput. **2** (2004) 323–339 http://www.r-gma.org/.
14. Capit, N., Costa, G.D., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., Richard, O.: A batch scheduler with high level components. In: IEEE Cluster computing and Grid 2005 (CCGrid05), University of Cardiff, UK (2005)
15. Vaughn, R., Evron, G.: Dns amplification attacks. Technical report, IsoTF (2006) http://www.isotf.org/news/DNS-Amplification-Attacks.pdf.
16. Molnar, D.: The SETI@Home Problem. http://www.acm.org/crossroads/columns/onpatrol/september2000.html (2000)
17. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural support for copy and tamper resistant software. In: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (Asplos IX). (2000) 168–177
18. Lie, D., Trekkath, C.A., Horowitz, M.: Implementing an untrusted operating system on trusted hardware. In: Proceedings of the 9th ACM Symposium on Operating Systems Principles (Sosp'03). (2003) 178–192
19. Lie, D.J.: Architectural support for copy and tamper-resistant software. PhD thesis, Stanford University (2004)
20. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: Aegis: Architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th International Conference on Supercomputing (Ics'03). (2003) 160–171
21. Suh, G.E., O'Donnell, C.W., Sachdev, I., Devadas, S.: Design and implementation of the Aegis single-chip secure processor using physical random functions. In: Proceedings of the 32nd Annual International Symposium on Computer Architecture (Isca'05), IEEE Computer Society (2005) 25–36
22. Lauradoux, C., Keryell, R.: CryptoPage-2 : un processeur sécurisé contre le rejeu. In: Symposium en Architecture et Adéquation Algorithme Architecture (SympAAA'2003), La Colle sur Loup, France (2003) 314–321
23. Duc, G., Keryell, R., Lauradoux, C.: CryptoPage : Support matériel pour crypto-processus. Technique et Science Informatiques **24** (2005) 667–701
24. Duc, G.: CryptoPage — an architecture to run secure processes. Diplôme d'Études Approfondies, École Nationale Supérieure des Télécommunications de Bretagne, DEA de l'Université de Rennes 1 (2004) enstb.org/~gduc/dea/rapport/rapport.pdf.
25. Huang, A.: Keeping secrets in hardware: the Microsoft XBox (TM) case study. Technical Report AI Memo 2002-008, MIT (2002)
26. Zhuang, X., Zhang, T., Pande, S.: Hide: an infrastructure for efficiently protecting information leakage on the address bus. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (Asplos-XI), Acm Press (2004) 72–84