Experiments on COTS Diversity as an Intrusion Detection and Tolerance Mechanism

Frédéric Majorczyk, Éric Totel, Ludovic Mé
firstname.lastname@supelec.fr

ABSTRACT

COTS (Components-Off-The-Shelf) diversity has been proposed by many recent projects to ensure intrusion detection and tolerance. However using COTS in a N-version architecture presents some drawbacks, especially in intrusion detection, which have consequences on intrusion tolerance. COTS Diversity is prone to raise many false positives (false alerts). In this article, we explain what a COTS Diversity architecture can detect and propose a masking mechanism to reduce the false positive rate. We apply this method to web servers and provide some experimental results that confirm the necessity of this mechanism.

Keywords

Intrusion tolerance, design diversity, COTS diversity, intrusion detection

1. INTRODUCTION

Design Diversity, and more especially N-version programming, are techniques used to detect and tolerate faults. They have been studied actively [1, 9] and have been used in many industrial projects. N-version programming consists in the execution of a single function by two or more elements, called versions, and the comparison of the results of the different versions to make a decision on the result. The underlying hypothesis is that the different versions used are independent from the point of view of their faults. N-version programming has been proved to provide a high coverage of fault detection [10], although some common-mode failures may still exist [8].

Recently several projects have explored the idea of using COTS (Components-Off-The-Shelf) instead of specifically developed software both in the dependability [5] and in the security field [14, 4, 3, 6, 16, 15, 2]. Indeed, developing specific software several times is very costly, while many Internet services are already implemented by COTS. In the security field, three projects, DIT [15], HACQIT [6], and BASE [2], focus on intrusion tolerance while the others [14, 4, 3] focus on intrusion detection. The intrusion tolerance property relies heavily on intrusion detection. While false negatives (missing of intrusions) have a direct impact on the intrusion tolerance, false positives (false alarms) can decrease the performance of the architecture and lead in some cases to a self denial of service (DoS) ; the availability of the system is then not ensured. A binary comparison of the outputs of the COTS can lead to a very high false positive rate. A very simple comparison algorithm can lead to many false negatives and may yield some false positives. Depending on the choice of the algorithm, it is necessary to introduce some mechanisms in the architecture to counterweight its drawbacks. We suggest here to implement a masking mechanism to reduce the false positive rate.

In this paper, we study, through the example of web servers, the false positive and false negative rates in a COTS diversity based architecture, their potential influence on the intrusion tolerance property and the effectiveness of a masking mechanism. In Section 2, we briefly present the DIT, HACQIT, and BASE projects which use COTS diversity for intrusion tolerance. Then, in Section 3, we analyse the type of differences detected by COTS diversity, and show the necessity for a masking mechanism avoiding false positives to be generated by the detection algorithm. Finally, in Section 4, we present some results with relation to intrusion detection.

2. RELATED WORK

Three recent projects have brought design diversity to the security field in order to detect and tolerate intrusions. We present here these three projects: DIT, HACQIT, and BASE.

2.1 The DIT Project

DIT (Dependable Intrusion Tolerance) [15, 16] is a project that proposes a general architecture for intrusion tolerant enterprise systems and the implementation of an intrusion-tolerant web server as a specific instance. The architecture includes functionally redundant COTS servers running on diverse operating systems and platforms, hardened intrusion-tolerant proxies that mediate client requests and verify the behaviour of servers and other proxies, and monitoring and alert management components based on the EMERALD intrusion-detection framework [11]. The architecture was next extended to consider the dynamic content issue and the problems related to on-line updating [13]. Intrusion detection relies mostly on host monitors and network intrusion detection systems, but is also enforced through the comparison of md5 hashes of the servers outputs. Once a COTS server is considered as compromised, it is reconfigured from a backup and can reinserted again in the architecture.

2.2 The HACQIT Project

HACQIT (Hierarchical Adaptive Control for QoS Intrusion Tolerance) [6] is a project that aims to provide intrusion tolerance for web servers. The architecture is composed by two COTS web servers: an IIS server running on Windows and an Apache server running on Linux. One of the servers is declared as the primary server and the other one as the backup server. Only the primary server is connected to clients. Another computer, the Out-Of-Band (OOB) computer, is in charge of forwarding each client request from the primary to the backup server, and of receiving the responses of each server. Then, it compares the responses given by each server. The comparison is based on the status code of the HTTP response. In addition, host monitors, application monitors, a network intrusion detection system like Snort [12] and an integrity tool (Tripwire [7]) are also used to detect intrusions. A mechanism of rejuvenation is used to restart possibly compromised servers in a safe state. In case of an intrusion, a sandbox mechanism is used to replay requests in order to find the sequence of malicious requests that has lead to the intrusion. These requests can then be rejected by the firewall.

2.3 The BASE Project

BASE [2] proposes to use an abstract model of the protected service to allows to use COTS and thus to reduce the cost of Byzantine fault tolerance, and to improve its ability to mask software errors. A diversified NFS service is built as an example. The abstract model, which describe the normal functioning of the diversified service, is used to normalize the outputs of the diversified implementations, and thus to mask the output differences due to specification differences or nondeterminism. This approach can be used if the service to diversify is well documented, in order to allow the definition of the abstract model. If the implementations used to build the service behave very differently, the outputs of one or several implementations will deviate significantly from what is modeled in the abstract model.

2.4 Discussion

In these two projects, the intrusion tolerance property relies to a great extend on the detection mechanism. If an intrusion occurs and is not detected, the compromised server must be considered as a byzantine process. Then it can skew all the next results of the comparison algorithm and so the intrusion tolerance property is not ensured anymore. It is then possible to compromise other servers in the architecture without being detected. There are many ways to circumvent false negatives or their effects: using a very strict comparison algorithm, adding several others IDSes, reconfiguring regularly the servers. The reconfiguration of a server is obviously required when an intrusion is detected but, as the detection mechanism can miss some intrusions, it seems essential to reconfigure the servers periodically. DIT, HACQIT and BASE implement it. DIT and HACQIT add also other IDSes to reduce the probability of a false negative.

Too many false positives can decrease the global performance of the system and even lead to a self DoS. In the HACQIT project, in case of an alert generated by a request, a window of past requests is replayed in a sandbox to establish the requests that are responsible for the alert. These requests are then rejected by the firewall. If the requests considered as malicious are in fact sound, the server will not respond to normal requests and so is partly unavailable. To resolve the problem of false positives, we introduce a masking mechanism, which allows us to use a strict comparison algorithm and thus to be able to detect and tolerate intrusions without additional IDSes. The BASE project masking mechanism avoids some false positives. Nevertheless, this require an *explicit* model of the service, which may be difficult to build, as we propose to use an *implicit* model associated to an *explicit* model limited to the definition of known differences.

3. INTRUSION TOLERANCE BY COTS DIVERSITY

We present first some drawbacks of COTS Diversity with regard to classical N-version programming. Then, we detail a general taxonomy of the differences detected by a COTS diversity based architecture.

3.1 COTS Diversity Drawbacks

Using COTS in an N-version architecture leads to some drawbacks in the detection process. There is no proof that the assumption of independent failure is verified by the COTS used. Consequently, a study of the known vulnerabilities must be performed in order to ensure that this hypothesis is verified. This has been carried out in several studies, such as [5] that shows that there are very few common failures for COTS databases, or the one of [17] that Apache and IIS servers have no common vulnerabilities.

Moreover, the specification of the COTS neither precise what are the data to be compared, nor when it has to be compared. Thus, a choice has to be made about that two points. This choice will have a major impact on the detection. On one hand, a very strict comparison algorithm can generate many false positives but no false negatives. On the other hand, a loose comparison algorithm may generate comparatively few false positives but miss several intrusions.

Finally, the specifications of the COTS used may not be known exactly and some differences may exist though the COTS are supposed to implement the same service. The comparison may then lead to output differences that are not only due to software failures but also to design or specification differences. If not handled correctly, these differences may cause many false positives. We think that this is the main issue with COTS diversity based architecture. Thus we detail in the next Section what kind of differences such an architecture will detect.

3.2 Taxonomy of Detected Differences

In N-version programming, since the different versions have the same specification, an output difference means that a fault has been activated in one of the version. That is not necessarily the case when COTS are used. COTS software have indeed not exactly the same specification. The specification of a COTS with respect to other COTS can be viewed as a common part and a specific part that differs from other variants specific parts.

Thus, the output differences that are detected are the results either of design differences that are due to design faults in the part of the program covered by the common specification, or design differences that are due to differences in the specific parts of the specifications. These later design differences are not necessarily (but can be) design faults;

Design faults can in their turn be divided in two different sets: classical design faults and vulnerabilities. Classical design faults are faults in the system that cannot lead to a violation of the security policy of the system while vulnerabilities are faults that can be exploited to breach the security of the system.

Clearly, without an additional mechanism, the comparison algorithm would detect all these kinds of differences. The output differences detected that are due to classical design faults or specification differences would actually be false positives, because they do not imply any violations of the security policy. These false positives must, of course, be eliminated. This elimination can be performed by masking the legitimate differences. The masking functions are applied to modify the request before it is processed (pre-request masking: proxy masking function) or after the request has been performed (post-request masking: rule masking mechanisms). In both cases, an off-line experimental identification of the specification differences is needed. It is not easy to evaluate theoretically the sets of differences detected and the need of masking rules since it depends on the COTS and the comparison algorithm used. We decided to evaluate them experimentally on a web server implementation, presented in the next Section.

4. APPLICATION TO WEB SERVERS

First, we present briefly the detection algorithm and the output difference masking mechanism. We expose then the results with relation to detection.

4.1 Detection Algorithm

The detection algorithm depends on the application monitored and must be developed specifically for each application considered. Here, we compare the HTTP responses from the web servers, since HTTP is obviously part of the common specification between the COTS.

The detection algorithm is composed of two phases. First, a watchdog timer provides a way to detect that a server is not able to answer to a request. All servers that have not replied are considered to be unavailable, and an alert is raised for each of them. Then, the comparison algorithm is applied on the set of answers that have been collected.

When all server responses are collected, we first try to identify if these answers are known design differences. In this case, we mask the differences by modifying some of the headers. Then, we begin the comparison process itself. As the comparison of the body can consume a lot of time and CPU, the detection algorithm compares first the status code, then the other headers in a given order (Content-Length, Content-Type, Last-Modified), and eventually the body. If no majority can be found amongst the responses from the servers, the algorithm exits and an alert is raised. It is useless to compare the body and the other headers of the responses if the status code is not of type 2XX (i.e., the request has not been successfully processed). In this case, the response is indeed generated dynamically by the web server, and may differ from one server to the others. (If these bodies were compared, it would generate a large amount of false positives.)

4.2 Output Difference Masking

The recognition of the output differences that are not due to vulnerabilities is driven by the definition of rules. These rules define how such differences can be detected. They currently take into account several parameters, such as: a characteristic of the request (length, pattern matching, etc.), the status code, and the Content-Type headers. For example, a rule can define a relation between the outputs, e.g., between the status code of the several outputs. Another example would be to link a particular input type to its expected outputs.

It is not possible to define all differences using these rules. For example, Windows does not differentiate lower case letters from upper case letters, and thus we had a lot of behaviour differences due to this system specification difference. Thus we added a mechanism in the proxy which processes the requests to standardize them before they are sent to the servers. Thus, all web servers provide the same answers.

The output difference masking mechanism is thus divided in two parts: pre-request masking mechanisms that standardize the inputs and post-request masking mechanisms that mask the differences that are not due to errors in the servers.

4.3 Experimental Results

The objective of the tests that have been conducted is to evaluate the COTS diversity based detection mechanism in terms of both reliability and accuracy of the detection process. The reliability of the approach is its ability to detect correctly all the intrusions. The accuracy refers to its capacity to avoid false positives generation.

It is not easy to evaluate detection reliability for practical reasons (vulnerabilities affect specific versions of the servers in specific configurations). However, our previous work [14] shows that the detection mechanism is valuable since it is able to detect several intrusions launched against our web server implementation.

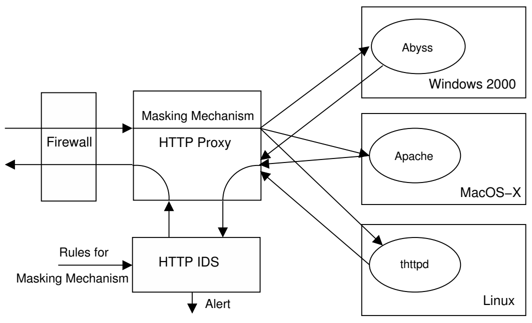

Figure 1: Architecture for the Accuracy Test

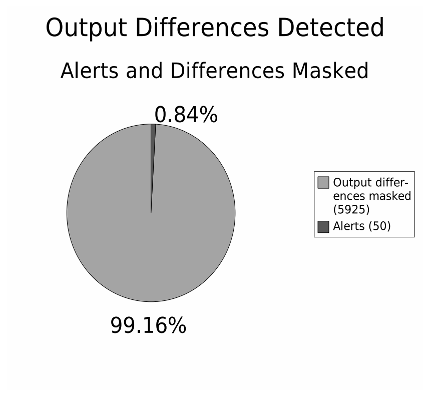

Figure 2: Analysis of the Detected Differences

In order to evaluate the detection accuracy, we set an architecture with three servers shown on Figure 1. We use two sets of HTTP requests. Both contains HTTP requests logged on the website of our campus during a week. The first one is composed of 71,596 requests and is used to write the masking rules. The second one is composed of 105,228 requests and is used to evaluate the detection mechanism. On the test set, 50 alerts are raised, which represents about 7 alerts a day. After analysis, all these alerts are false positives. Without masking mechanisms, 5975 alerts would have been raised or in other words, the comparison algorithm detects 5975 output differences. 99.16% of the output differences are then masked by masking rules.

It must be noticed that, in case of an intrusion or an attack, the localization of the server attacked is often not possible since there is no majority in the responses of the servers. A request can indeed activate a design fault (especially a vulnerability) and induce an output difference due to design differences between all COTS used. If the localization is not possible, it is necessary to reconfigure all servers.

5. CONCLUSION

As a conclusion, we can state that the COTS Diversity approach provides a high coverage of detection (consequence of COTS diversity and hypothesis of decorrelation of vulnerabilities).

However using COTS in a intrusion tolerant architecture must be done carefully: the choice of the COTS, the comparison algorithm and the masking mechanisms have an impact on the false positive rate. By our experiments, we have shown that a high amount of output differences are actually due to design differences and not to exploits of vulnerabilities. We conclude that a masking mechanism is mandatory for COTS Diversity being effective in intrusion detection and tolerance.

6. REFERENCES

[1] A. Avizienis and J. P. J. Kelly. Fault tolerance by design diversity: Concepts and experiments. *IEEE Computer*, 17:67–80, August 1984.

[2] M. Castro, R. Rodrigues, and B. Liskov. Base: Using abstraction to improve fault tolerance. *ACM Trans. Comput. Syst.*, 21(3):236–269, 2003.

[3] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: A secretless framework for security through diversity. In *Proceedings of the 15th USENIX Security Symposium*, Vancouver, Canada, August 2006.

[4] D. Gao, M. K. Reiter, and D. Song. Behavioral distance for intrusion detection. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, pages 63–81, Seattle, WA, September 2005.

[5] I. Gashi, P. Popov, V. Stankovic, and L. Strigini. *On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers*, volume 3069 of *Lecture Notes in Computer Science*, pages 196–220. Springer-Verlag, 2004.

[6] J. E. Just, J. C. Reynolds, L. A. Clough, M. Danforth, K. N. Levitt, R. Maglich, and J. Rowe. Learning unknown attacks - a start. In A. Wespi, G. Vigna, and L. Deri, editors, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 158–176, Zurich, Switzerland, October 2002. Springer.

[7] G. H. Kim and E. H. Spafford. The design and implementation of tripwire: a file system integrity checker. In *in Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 18–29, Fairfax, VA, November 1994.

[8] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *Software Engineering*, 12(1):96–109, 1986.

[9] M. R. Lyu and A. Avizienis. Assuring design diversity in N-version software: A design paradigm for n-version programming. *Dependable Computing and Fault-Tolerant Systems*, 6:197–218, 1992.

[10] M. R. Lyu and Y.-T. He. Improving the N-version programming process through the evolution of a design paradigm. *Transactions on Reliability*, 42(2):179–189, June 1993.

[11] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. of the 20th National Information Systems Security Conference*, pages 353–365, Baltimore, MD, October 1997.

[12] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the USENIX LISA'99 conference*, pages 229–238, Seattle, WA, November 1999.

[13] A. Saidane, Y. Deswarte, and V. Nicomette. An intrusion tolerant architecture for dynamic content internet servers. In P. Liu and P. Pal, editors, *Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems (SSRS-03)*, pages 110–114, Fairfax, VA, October 2003. ACM Press.

[14] E. Totel, F. Majorczyk, and L. Mé. COTS diversity based intrusion detection and application to web servers. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, pages 43–62, Seattle, WA, september 2005.

[15] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saïdi, V. Stravidou, and T. E. Uribe. An adaptive intrusion-tolerant server architecture. In *Proceedings of the 10th International Workshop on Security Protocols*, pages 158–178, Cambridge, United Kingdom, April 2002.

[16] P. E. Veríssimo, N. F. Neves, and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*. Sptringer-Verlag, April 2003.

[17] R. Wang, F. Wang, and G. T. Byrd. Design and implementation of acceptance monitor for building scalable intrusion tolerant system. In *Proceedings of the 10th International Conference on Computer Communications and Networks*, pages 200–5, Phoenix, AZ, October 2001.