



ENISA Quarterly

IN THIS EDITION

Secure Software

	Page
A Word from the Executive Director	1
A Word from the Editor	2
From the World of Security – A Word from the Experts	3
Cycles of Software Crises	3
The Whys and Hows of Assuring Secure Software	5
Technology Leaders Tackle Software Assurance	9
The 10 Most Common Sins of Software Developers	9
Security Skills of Software Developers	11
Leading the Way to More Secure Software	14
Providing Assurance for Security Software – Insights into the Common Criteria	15
From our Own Experts	17
ENISA Position Papers	17
Towards an EISAS	21
Food for Thought	23
Stop Using the Traffic Analogy	23
ENISA Short News	24

As we reach the end of another year, it is a good time to look back at the little more than two years since we began operations. It is valuable to revisit the roots of ENISA: i.e., our founding regulation, EC460/2004. Article 3 of the Regulation details our main tasks, which, put simply, are to:

- Give **independent, expert advice** to the EU, as the first step towards the drafting of legislation
- **Respond to requests** from Member States and the EU
- **Collect and analyse** data on security incidents and emerging risks
- **Promote best practices** in risk assessment & risk management, awareness-raising and computer security incident response.

Comparing the Regulation with our Work Programmes for 2006 and 2007, I am delighted to see that the Agency has delivered its operational and administrative tasks in full.

Our activities by the end of 2007 were challenging and numerous. Let me mention but a few of our achievements spread across Europe. We organised workshops on 'Risk management – Why Business Needs it?' in Barcelona, on 'A Data Collection Framework on security incidents and consumer confidence', co-located with eChallenges in The Netherlands, and on the 'Barriers and Incentives for NIS in the Internal Market for eCommunications' in Brussels. We held a meeting of our Permanent Stakeholders' Group in Brussels on the Work Programme 2008, focusing on resilience, and our feasibility study into a European Information Sharing and Alert System (EISAS) was finalised.

ENISA has also recently launched Position Papers on Botnets, Social Networking Sites, and Online Reputation Systems, which have already had an impact on both Network and Information Security (NIS) and international media.



Andrea Pirotti, Executive Director of ENISA

The process of preparing next year's Work Programme, with decisions on the Agency's priorities being taken jointly with all the stakeholders, guarantees that our activities are in line with the NIS needs, expectations and demands of the whole of Europe. The adoption of ENISA's Work Programme 2008 by our Management Board is a positive acknowledgement that a multi-stakeholder process for defining activities and establishing priorities is the way forward for Europe. I thank all our staff and stakeholders for their hard work which made this achievement possible.

The Work Programme 2008 title is 'ENISA driving for impact', with the focus on a number of Multi-Annual Thematic Programmes (MTPs):

MTP 1 'Improving resilience in European e-Communication networks' focuses on the identification of current best practices, gap analysis, analysing Internet integrity technologies, and the stability of networks. This MTP will support the review of the EC Electronic Communication Directives. MTP 2 will develop and maintain co-operation models, in order to use and enhance the existing networks of actors in NIS. In 2008 this MTP will be devoted to a) the identification of Europe-wide security competence circles in Awareness Raising & Incident Response, b) co-operation on the

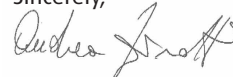
interoperability of pan European eID, and c) the European NIS good practice Brokerage. MTP 3 will identify emerging risks for creating trust and confidence. The Agency will develop a 'proof of concept' of a European capacity for the evaluation of emerging risks, linked to a Multi-Stakeholder Dialogue Forum for public and private sector decision-makers. Finally, the Agency will undertake a 'Preparatory Action', which includes a feasibility study

into the needs of and expectations for NIS in micro-enterprises.

These MTPs will guide us to the doorstep of the New Year and lead us throughout 2008, but reflections regarding the Agency's future role will also continue; we shall tackle new challenges and work projects in the months to come with the same commitment and team effort that has already brought us so far in such a short time.

You are all warmly invited to follow the Agency's activities and achievements throughout 2008 on our website and in the ENISA Quarterly. Wishing you all the best for a safe 2008 and beyond.

Sincerely,



Andrea Pirotti
Executive Director, ENISA

A Word from the Editor



Most IT security problems today stem from vulnerabilities in software. The fact that only a small fraction of these vulnerabilities are being exploited by attackers shows that there is still a great risk that more will take advantage of the constantly increasing number of newly found vulnerabilities.

By 'Secure Software' we mean software that is robust in the face of both intentional and unintentional incidents. To minimise risk it is important that the software industry re-examines the way software is built, shipped and deployed. The traditional approach of securing the perimeter and applications via security product add-ons such as firewalls and intrusion detection systems has proved to be inadequate. It is therefore clear that software developers will have to take their share of the responsibility for improving application security.

The challenge to the application development community is to create the conditions where a focus on functionality does not imply insecure applications. The key question is how we can be reasonably confident that the applications we run are secure if developers do not pay adequate attention to security issues.

Software assurance encompasses a developing set of methods and processes for ensuring that software functions as intended without introducing vulnerabilities, malicious code or defects that could harm the end-user. Put into practice, software assurance can reduce IT vulnerabilities, improve resistance to attack and protect supply chain integrity.

In this issue of EQ we shed some light on the practices that could help create more secure software. Markus Bautsch, the consumer

representative on ENISA's Management Board (MB), opens this EQ by looking into the issue of insecure and uneconomic software and how to avoid it. Sachar Paulus, a member of ENISA's Permanent Stakeholders' Group (PSG), provides a summary of best practices for developing secure software that complement and go beyond the standard certification approaches, advocating the need for self-regulation in the software industry.

Paul Kurtz presents the industry initiative SAFECode, the Software Assurance Forum for Excellence in Code, which was launched in October 2007. SAFECode is a non-profit organisation dedicated exclusively to increasing trust in information technology products and services through the advancement of proven software assurance methods. The organisation has a technical and educational focus.

The next two articles focus on enhancing the security knowledge and awareness of software developers. Shay Zalalichin presents the 10 most common sins committed by software developers, as identified by the Open Web Application Security Project (OWASP). Johan Peeters, Ken van Wyk and Frank Piessens present their vision of how software developers can work towards meeting security challenges without cutting themselves loose from their economic basis. This vision has inspired the curriculum design of the secure application development course that they have been running since 2005, and which this year is endorsed by ENISA.

Finally, David Ochel's article demonstrates how certification, namely the Common Criteria, can be effectively used to provide assurance of a software product's purported security functionality, and describes a means of maintaining a high-level view of a security evaluation.

Our own expert Giles Hogben introduces the three Position Papers that ENISA has recently published representing expert opinion on important emerging Network and Information Security risks affecting Social Networks, Reputation Systems, and Botnets. Each paper has been produced by independent expert groups from industry, academia and Member State governments

who have been selected for their expertise in the relevant area and do not represent any specific corporate interests. His article summarises the findings of these groups.

Marco Thorbruegge discusses the main points of the feasibility study into a Europe-wide Information Sharing and Alert System (EISAS) that ENISA took over in 2007. In order to provide thorough and responsible advice to the European Commission, ENISA first conducted an analysis of the current state of play in both the public and private sectors in all EU Member States, and identified possible sources of security information which could contribute to an EISAS.

Last but not least we have the regular 'food for thought' column provided jointly by our MB member Pernilla Skantze and our PSG member Nick Coleman, who urge us to stop using the traffic analogy when discussing security issues.

All in all, 2007 was a very good year for EQ, with three special issues and a large number of downloads. Within the first months of 2008 we will provide a comprehensive article with interesting statistics about EQ. In the meantime please continue to send us your articles. Generally we are open to articles in *all* areas of NIS. For the next edition, which is due at the end of March 2008, we would particularly like to receive contributions related to the first Multi-annual Thematic Area of ENISA's work programme for 2008, which is:

• Improving resilience in European e-Communication networks.

We trust this issue will increase awareness of the need for secure software, and serve as a good source of information for the interested reader. I hope you enjoy reading it as much as I enjoyed editing it!

Sincerely,

Panos Trimintzios
Editor-in-Chief, ENISA Quarterly

Dr. Panagiotis Trimintzios is an Expert at ENISA responsible for Relations with Industry, Academia and International Organisations.

ENISA Quarterly Vol. 3, No. 4, Oct-Dec 2007

Cycles of Software Crises

How to avoid insecure and uneconomic software

Markus Bautsch



Two sides of the same coin

Let's be honest – it can be difficult to find measures to ensure secure software. On the other hand it is relatively easy to name many properties and characteristics of insecure and bad software. Unfortunately a lot of responsible managers in software developing companies, and also a large number of professional programmers, seem to be absolutely unaware of issues related to secure software – or else they quite simply ignore them. In addition, there is a common assumption that well-established tools must be secure and efficient. This can be a fatal fallacy.



We have seen it all before

Every paradigm persists till new phenomena occur that no longer coincide with the predominant doctrine and circumstances. In the sixties it became obvious that the state-of-the-art of machine-oriented software development had reached a dead end. The evolution of tools could not keep pace with the technical innovation of hardware. This was called a 'software crisis' because software projects ran out of time and budget, and the programmes produced were of poor quality and no longer maintainable.

The result was that engineers invented high-level programming languages with interpreters and compilers, such as Pascal and C, which made programming more

abstract and more or less portable. The improved performance of hardware even allowed the establishment of multitasking operation systems.

Later on, in the 'eighties, the information technology community started to establish another programming paradigm, whereby software could be modular and object-oriented. This approach was even more abstract, and finally it became possible to manage larger software projects.

What we observe

Whenever a paradigm shift in programming became necessary, we observed that many experts tried to hang on to their traditional knowledge and tools for as long as possible. At the very best they adapted their existing tools to the new challenges, but this was always at a price, namely an increasing number of problems; furthermore it finally led to improper and outdated solutions and products. Even nowadays – many years after the last large scale paradigm shift in information technology – we find many software products which still suffer from ambiguity, non-modularity, type-unsafeness or memory corruption. The latter regularly leads to memory leaks, dangling pointers or buffer overflows, which are the cause of many severe software problems. The list of drawbacks could be extended, for example, by SQL- or http-header-injection, which are related to insecure input, output or exception handling.

Many apparently evolved programming languages and development tools only fixed a small number of minor issues, without addressing the real challenges. It would

have been better to use really new and improved tools which, incidentally, do not necessarily have to break with all traditions. Outdated programming languages are still in use today and a pithy comment made by the famous Swiss computer scientist, Niklaus Wirth, is appropriate in this context: *"It is indeed absolutely surprising with which equanimity the notational monster C was accepted by the world-wide programmer's community."* There can really be no debate about this because, for example, there are still job adverts asking for programming skills in C for automotive security systems today. Who really wants to drive a car with such an outdated security system?

What we need

Today we are in the middle of a new software crisis. The working day of many programmers is dominated by endless run-time debugging sessions with trials to fix the misbehaviour of thousands of lines of complex software. The widespread use of run-time debuggers cannot really do much to achieve more secure software. Run-time debuggers are not only dispensable, but also counterproductive and they are not economical because it is not possible to completely determine the behaviour of complex polymorphic objects in modern run-time environments.

It would be better to use tools which do not allow insecure structures and hierarchies, such as weak typing, ambiguous name spaces or scopes, as well as unchecked open array pointers. Moreover, these tools must generate secure machine code by default with run-time checking of array bounds,



buffer and stack sizes as well as exception handling. Furthermore, it is not only possible, indeed it is a necessity to leave dynamic name binding and memory disposal to the run-time environment. Last but not least, the less code is reused and instead is recreated because of bad software architecture, the larger the software programmes become. This means that there are more errors for the programmers to check and fix – an expensive and unnecessary nightmare. To cite Orfali et al. in their book, *The Essential Distributed Objects Survival Guide (1996)*: “Only the consumer gets freedom of choice; designers need freedom from choice”.

It is a myth that security measures lead to performance problems. This is not the case at all because, with modern hardware memory disposal, security checks and method calls take less than a microsecond. This is not noticed by the user, but these features provide substantially safer and more secure executable code, and this reliable code can even be reused in appropriate architectures. It is important to mention that this even holds true for embedded systems and real-time applications. On the other hand, development cycles are substantially shorter – and time is money, isn't it?

Examples

Let us have a closer look at the four programming languages, C and its predecessor C++, Component Pascal (CP), Java and C Sharp (C#). The table right compares nine properties which are important, among other things, for easy, safe and secure programming.

Niklaus Wirth classifies C++ as “an insult to the human brain”, not only because of the shortcomings illustrated in the table; C++ has inherited too many unsafe features from its predecessor, C. According to the table, the only advantage could have been the inheritance of static objects, which can be implemented very efficiently. Unfortunately static objects are also very efficient for garbage collection, which is neither mandatory nor safe in C++.



Static objects are not possible in Java at all, but Java also suffers from too many standard libraries of classes and methods, which result in rather too high a threshold for a good command of all possibilities. Unfortunately it allows multiple

Comparison of example programming languages

	C / C++	CP	Java	C#
Publication	1972/1985	1994	1995	2001
Structured syntax	no	yes	no	no
Simplicity and regularity	no	yes	no	no
Cyclic imports impossible	no	yes	no	no
Static objects	yes	yes	no	yes
Only simple implementation inheritance	no	yes	no	yes
Full module safety	no	yes	no	yes
Full type safety	no	yes	yes	yes
Automatic and mandatory garbage collection	no	yes	yes	yes
Dynamic loading	no	yes	yes	yes

implementation inheritance which is dispensable and leads to ambiguities and a large amount of overhead for the compilers. Last but not least many programming languages still suffer from complicated or unstructured syntax, such as too many hierarchy levels, the possibility of operator overloading, and missing or confusing definitions, as well as badly motivated rule exceptions, which are easily avoidable. Strangely enough, all these facts do not seem to have too much impact on the programming community.

So what should we do about this?

There are best practices for developers, which should be co-ordinated, propagated and taught much more frequently and intensively. If software project leaders do not understand these vulnerability problems, they urgently need support and advice, which should be given to them not only by academic partners, but also by standardisation bodies and public authorities. Lifelong learning is essential; it is not only students who have to be educated, but also teachers and decision-makers.

In addition to extensive and continuous security education for managers, designers and developers, there are many interesting and innovative technological solutions to help to improve the security properties of software.

Service-oriented architecture (SOA) is a good approach for achieving better software solutions, for example. It supports distributed computing, code reuse, data and code encapsulation as well as standardised and secure interfaces. However, it is essential that the underlying software architecture is sophisticated and can be assessed with regard to risk and quality. This is often not the case or is inadequately realised.

This assessment can even be carried out by the software itself; **proof-carrying code (PCC)** is one noteworthy example. This is an integrated mechanism that allows verifying software properties via formal proof. The run-time system can derive information itself and can decide whether a software component is safe and may be executed. The programmer does not have to be concerned about the safety of a method or a module – this can be done automatically by reliable and well-established code; he only has to make sure that he uses qualified development tools and run-time environments. Many researchers are already studying the capability and benefit of proof-carrying code, such as in the international project **Mobility, Ubiquity and Security (MOBIUS, <http://mobius.inria.fr>)**, but we should all intensify our efforts and improve our knowledge about inherent information and network security.

Mutual benefit

With technologies which really deserve to be called new technologies we will benefit from slimmer and more reliable software, which is easier to use and to maintain. Clemens Szyperski states in his book on *Component Software – Beyond Object-Oriented Programming*: “With abstraction, less becomes possible in theory, but more becomes possible in practice”.

In mature and proper **model-viewer-controller architectures (MVC)** within a component-oriented framework, for example, it can be very easy and efficient to implement generic undo-redo-functions, automatic garbage collection and embedded media or compound documents without any ballast and performance issues.

Another principle, which is also known as ‘Wirth’s law’, states: “*Software gets slower faster than hardware gets faster!*” Component-based software engineering not only has the potential to disprove this law,

but has already done so. A side-effect of the new programming paradigm of component orientation is improved security and better portability, but it has to be implemented by skilled and aware engineers and programmers. We should not trust in information and communication technologies without checking whether the development tools meet the requirements of an economic and secure future. ENISA will support us in our attempt to do this.

Markus Bautsch (M.Bautsch@stiftung-warentest.de) has worked on developing component-oriented software and has lectured on programming languages at the University of Applied Science for Technology and Economy in Berlin; he is responsible for international consumer testing of high-tech products; he is also a member of the End User Panel of MOBIUS and is the consumers’ stakeholder representative on the ENISA Management Board.



The Whys and Hows of Assuring Secure Software

Sachar Paulus



This article provides a summary of best practices for developing secure software. Existing certification approaches, such as the Common Criteria, have their drawbacks so one should identify the techniques and principles which are most useful and which make most sense. This activity has been driven by the software industry throughout the last two years and thus, to some extent, these findings represent the software industry’s view of the problem. The result is: **we need to work together on best practices and identify practical implementation regimes.**

Why a new approach?

There are basically two existing approaches: either the certification of products or the adoption of best practices limited to implementation only.

The security certification – notably using the Common Criteria framework – of a product works perfectly well for products that are built once and will not be changed during their lifecycle, such as smart cards or mission control software. It does not work for common software, since changes, improvements and customisation are the norm rather than the exception, hence innovation speed overrules the processes needed for certification. Moreover, standard software is often used in a context for which it was not made, i.e., the certification cannot take all possible usage scenarios into account, including change/customisation of the software for specific purposes. In addition, for a significant part of the overall software industry, such as custom solutions, small independent European software vendors, open source development etc., it is not economically feasible as the certification costs are very high. Therefore, the value of the security certification of a software product is questionable – by definition it will lag behind the actual version of a product, taking into consideration hot fixes, updates, enhancements etc. Moreover, ‘static’ certifications do not allow for the potential future risk which results from social engineering, and the constantly changing assumptions about people’s behaviour.

Instead we need another approach which should be easy and flexible, so that companies that strive for innovation have no – or only minimal – difficulty in achieving the goal of secure software. It should also be process-oriented, so that it is applicable for

all ‘versions’ of a product, including patches and customisations of user installations.

Limiting remedies to the development phase only is often another mistake; ensuring that security issues will not be created during implementation does not protect the software from potential security risks. Often software or pieces of software are used in contexts that the developer did not anticipate. Thus looking only into the development phase will not prevent security issues from occurring, which incidentally is often the reason for security issues in open source software. Secure software best practices should be applied to all phases of the software lifecycle: from requirements definition to maintenance.



Therefore the approach to secure software should be neutral in three dimensions:

- Technology neutral – independent of the development process
- Business model neutral – independent of the way that users use the software, for example standard software, software embedded in products, custom developed software or online software provisioning
- Size neutral – applicable by small as well as large organisations.

How to reach the goal of secure software

As a first step, a best practices guide should summarise the good experiences of organisations in attempting to deliver secure software. 'Secure software' means that it is robust in the face of both intentional and unintentional incidents. The purpose is first and foremost to find a unified approach based on best practices to withstand intentional incidents, i.e., malicious attacks, but at the same time this may also improve the reliability of software.

In addition, a *baseline* recommendation is needed; it should be applicable for the whole software industry, independent of the development model, the business model or the size of the company. During the whole development cycle, high security environments may need additional safeguarding principles which go well beyond the baseline recommendation; in this case one may even require completely different approaches. The baseline recommendation should consist of principles to be followed and should encompass simple and easy-to-understand hints and recommendations as to how to implement the principles through all phases of the software lifecycle:

- product definition/requirements gathering
- design
- development
- testing
- roll-out
- documentation
- updates (product enhancements, product fixes)
- emergency hot fixes.

The approach should be inclusive, not elitist. Users often use a combination of products/solutions from different suppliers, affecting the overall security experience. There are roughly 37.000 independent software vendors in Europe and more than 3 million software developers, working in large standard software/product companies, system integrators, in-house development departments and many single-person companies. All these should be able to apply the principles formulated in the best practices guide. It should also contain a timetable as to how and when to communicate security issues to users. The hints and recommendations should be concrete and easily implemented, repeatable and, ideally, comparable. They should focus on process steps towards the goal of secure software.

Some thoughts on relevant regulatory policy

There is a lively ongoing discussion about whether software security should be regulated. Two prominent examples are often cited: the pharmaceutical sector and the automotive sector. Whereas the pharmaceutical sector suffered heavy regulation in its early days to ensure the security of its products, the automotive sector managed to make security a market

differentiator without regulation imposing requirements on the products. There is one major difference though in the software environment: it suffers from attackers who deliberately try and break software for personal or organisational benefit. This is simply a fact of life and must be addressed with additional vigour by the software industry.



It makes sense to ensure that the risk is taken by those who have the capability to deal with it. But this should not be imposed by regulatory means; it could also be in the form of 'social capital', i.e., trust by the user. In this situation, self-regulation would work just as well as regulation.

Since the policy bodies' approaches to making software more secure are perceived by the industry as unrealistic and restrictive in their application, the industry itself should propose how it should deal with the issue: industry should take responsibility for the security of its software (during its whole lifecycle) and drive general software development methods by the dissemination of best practices, to reach a security level that is acceptable throughout the industry.

Regulatory processes are normally static activities, and the adoption cycles for modifications can be quite long – the field of security understanding, the evolution of software development, the evolution of threats and attacks are a reality – so the processes to address these issues need to evolve. This is the primary benefit of an industry-driven initiative. Regulation could be outdated at its inception.



Following best practices could be marked by issuing labels of trust, eventually perhaps even by certification of the organisation that delivers the software similar to ISO 9000, but specifically targeted for secure software practices.

Whether or not the policy-makers should then decide to make such a certification mandatory is a matter for future debate. Hopefully the industry will have adopted the best practices by then so that policy intervention would be superfluous.

Non-goals

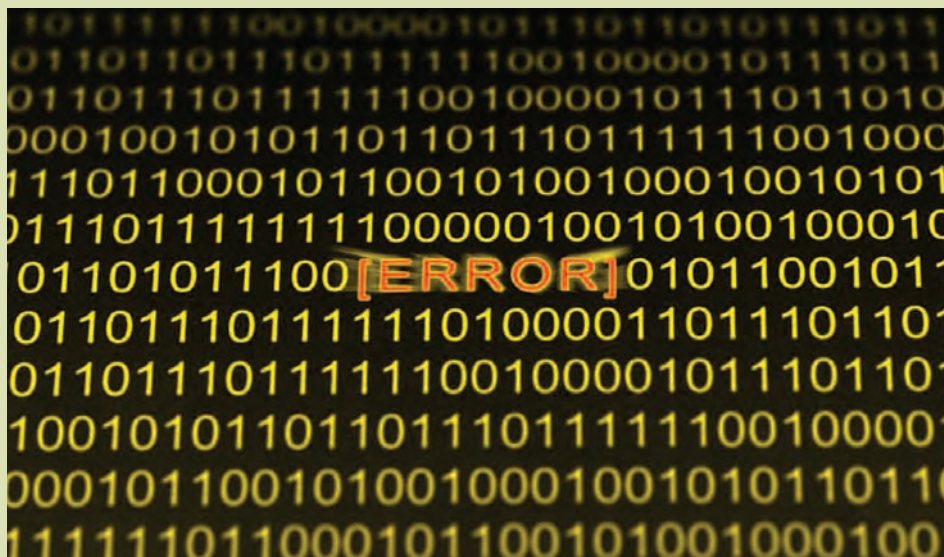
The best practices should be limited to the process aspects of secure software development. Goals that should *not* be followed are:

- To try and describe all potential vulnerabilities and remedies and request proof against these vulnerabilities. This could never be complete and up-to-date. Moreover, this knowledge is already available.
- To define the process steps in detail. The implementation of the processes should be left to each manufacturer; otherwise it would put too much pressure on the specific software producer. (Technology neutrality)
- To set up an exchange forum to discuss specific vulnerabilities. This might lead to the infringement of IPR or confidentiality agreements. (Business model neutrality)
- To enforce the use of specific security technologies.
- To address compliance issues around security such as, for example, export issues, i.e., potential back doors that may be necessary for shipping to specific regions. It might be a goal though, to ensure that such back doors can only be used by authorised bodies.
- The creation of a vibrant academic environment for the secure software development process.

A first set of best practices: the '10 Principles'

The following is a first attempt to list the principles that should be adhered to by the software industry if it is to deliver secure software. This should be an all inclusive approach; it should apply to standard and customised software manufacturers and IT project consultants, as well as hosting operators – named simply 'supplier' in the following. The supplier may produce standard software, project solutions or hosted applications – named 'product' in the following.

The principles listed right describe the general characteristics of risk mitigating activities; they do not explain how these activities should actually be achieved, since this might be very much subject to individual organisations, processes and the business models of individual IT companies.



The 10 Principles for Secure Software

1. During the requirements gathering process, the supplier should define the security assurance properties of the product.

This is a way of improving the likelihood that the product will incorporate the features and software quality necessary to block hostile attacks. The security assurance properties of the product, as designed, should ideally include resistance to potential threats, assuming a well defined environment. Checking the status of the predefined properties in subsequent development phases is a first indicator as to whether the product will meet the user's security demands. It also makes sense to check the planned features of the product itself against user demand, to compare historic attack vectors with earlier versions of the product or similar products, and to look for possible exploitations.

2. The design and implementation of the product should be constantly reviewed to check that it does not jeopardise the security assurance properties.

It is very useful to constantly review the design and the implementation of the software product in source code to check whether the promised/targeted security properties are actually delivered. There are different approaches to implementing this rule. A relatively complex – but very successful – one consists of using automated 'static analysis' tools to detect potential vulnerabilities in source code. Design checks can be made using so-called 'threat modelling'.

3. Software should be tested to check whether the desired security assurance properties are present, before shipping.

Testing the product is essential. This can be performed in-house, by external specialists or third-party companies. Tests can consist of black-box testing, i.e., trying to break into

the software without any information, white-box testing, i.e., with knowledge of how the software works, or even with careful code review. Tests can often be performed in an automated way, especially bad parameter mutation or 'fuzzing' tests, but some tests may need manual intervention. Tests must be flexible, appropriate and economically feasible but they can never be considered exhaustive.

4. The software should be able to run in 'secure mode' after installation.

Users expect to 'install-and-run' the given software product and that it would then be secure. Since this cannot be achieved in all cases, it should at least be easy to achieve a secure configuration status. Ideally, deployment should be 'secure by default', which means that capabilities that are not required by the vast majority of users are turned off by default, reducing the attack surface. Users would then turn on the capabilities they need, thereby controlling the security level of the installed software.

5. The security administration interface of the product should be easy to understand and simple to use.

This is important because otherwise there is a high risk that existing options for securing the product would not be used due to their complexity. The interface for users as well as administrators must be as easy as possible – not all users or organisations have security specialists.

6. The required secure configuration of the software environment of the product should be documented and actively communicated to users.

Not all security requirements can be fulfilled by the product itself; additional conditions may need to be fulfilled in the environment. This should be documented in such a way that the user is made aware of these requirements.

7. During the product enhancement process, the supplier should check that new requirements do not jeopardise the existing security assurance properties of the product.

This is especially important if products are developed following an incremental approach, such as, for example, in the case of most standard software development. It is an industry practice to roll out products in versions; this principle should reflect that requirement. It might be helpful to classify code segments or libraries as 'critical' so that, if the supplier changed these parts, he would know to review any potential impact of the change on the security assurance properties.

8. Software updates should not jeopardise the security properties of the product.

The supplier should make sure that, when updating a product, existing properties, including security configurations that have been made by the user, are not changed, or that users are notified appropriately if they need to change configuration to retain the security properties.

9. The supplier should implement a response process for effectively addressing security issues.

In the case of a security issue becoming known to the supplier, standard bug-fixing processes might not be successful in resolving the issue – not only in existing software versions, but also in similar products and future releases. Therefore, a dedicated tracking process for success in issue-fixing should be implemented.

10. Security issues detected in the product should be communicated responsibly to users.

'Communicating responsibly' means that the supplier must decide how and when he communicates about security issues. The most important aspect of this is that the user should be helped to maintain the security of the product, despite the discovery of the security issue. Other users could include partners who modify the product on behalf of the software owner. Ideally, the communication should include an easily understandable recommendation as to how to minimise the risk. This could consist of applying patches or describing workarounds. For products that are used by a wide variety of users or unsophisticated users, the supplier should offer automated mechanisms and processes that allow users to preserve the security of the product without undue analysis or effort.

The software industry should strive to develop measurement techniques for the principles mentioned above to continually improve the security of their products.

To make them manageable during the software lifecycle, it is important that the security measures can be quantified. There is ongoing research within the security community to achieve easily usable results. Eventually, it might be useful to use key performance indicators for secure software development that have been agreed on an industry-wide basis.

Prof. Dr. Sachar Paulus (sachar.paulus@sap.com) is SVP Product Security at SAP, he is member of the board of 'TeleTrust', a non-profit organisation for promoting secure electronic processes, a member of ENISA's Permanent Stakeholders' Group, and a founding member of the 'Application Security Industry Consortium'. He also currently co-organises the curriculum for a Masters degree on 'Security Management' at the Technical University of Brandenburg, Germany.



World Class Standards

ETSI Security Workshop

As innovation in information and communication becomes ever more essential for business, public administration, public safety and commercial needs, a vast number of new technologies are being developed. Nowadays security is taken into account right from the design phase, rather than being introduced after the adoption of a technology. In fact, the correct use of security by providers and vendors of services and products is proving to deliver a positive return on investment.

Security standards, sometimes in support of legislative actions, are essential to ensure interoperability between products using the same security mechanisms, to provide globally acceptable security metrics and to ensure an adequate level of security for these products.

ETSI is hosting the 3rd ETSI Security Workshop from 15-16 January 2008, at its Headquarters in Sophia Antipolis, France.

The annual ETSI Security Workshop is rapidly becoming the greatest annual international security standardisation workshop, bringing together international Standards Development Organisations (SDOs) and security experts to discuss recent developments, share knowledge, identify gaps and co-ordinate future actions and work areas.

The workshop will include overviews of work being carried out in the area of security across standards and technical bodies, along with presentations from major organisations involved in security initiatives.

Participation in the workshop is free of charge, and open to everyone. For more information please visit:
<http://portal.etsi.org/securityworkshop/>.

Technology Leaders Tackle Software Assurance

Paul Kurtz



In October 2007, EMC Corporation, Juniper Networks, Inc., Microsoft Corporation, SAP AG and Symantec Corp. came together to announce the formation of the Software Assurance Forum for Excellence in Code (SAFECode). SAFECode is a non-profit organisation dedicated exclusively to increasing trust in information technology (IT) products and services through the advancement of proven software assurance methods. The announcement was particularly significant because SAFECode represents the first global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services.

Software assurance encompasses a developing set of methods and processes for ensuring that software functions as intended without introducing vulnerabilities, malicious code or defects that could harm the end-user. Put into practice, software assurance can reduce IT vulnerabilities, improve resistance to attack and protect supply chain integrity.

In fact, many individual companies, including the founders of SAFECode, have implemented effective methods for developing and delivering more secure and reliable software, hardware and services, and have achieved significant and positive results from their efforts. However, there has been no co-ordinated, industry-led initiative to build upon this positive work and promote best practices to advance software assurance more broadly – until SAFECode.

SAFECode brings together experts in subject matter to identify and share proven vendor software assurance practices, promote broader adoption of such practices into the cyber ecosystem, and to work with governments and critical infrastructure providers to leverage vendor practices to manage enterprise risks. Not surprisingly, there is no single method for driving security and integrity into and across the globally distributed processes that yield the products and services that we all rely upon. Yet, despite the differences in approaches, a productive software assurance dialogue between and among vendors and governments/critical infrastructure providers can amplify the positive results individual companies are achieving, thereby improving the overall security and reliability of the IT ecosystem.

SAFECode's specific objectives are to:

- Increase understanding of the secure development methods and integrity controls used by vendors
- Promote proven software assurance practices among vendors and customers to foster a more trusted ecosystem
- Identify opportunities to leverage vendor software assurance practices to better manage enterprise risks



- Foster essential university curriculum changes needed to support the cyber ecosystem
- Catalyse action on key research and development initiatives in the area of software assurance

One question SAFECode members are often asked is whether or not one of the goals of the organisation is to impact government regulations related to software security in Europe and the US. As evidenced by its objectives, SAFECode is not a lobbying group, nor does it have political aspirations. Rather, the organisation is one with a technical and educational focus – it is a place for IT and communications technology providers to come together to tackle some of the most challenging problems facing developers of technology products and services. In fact, SAFECode encourages the participation of government and academic organisations in its discussions.

Early in 2008 SAFECode will release its first paper, outlining its members' best practices for developing, producing, testing and maintaining secure software.

Membership in SAFECode is open to any information and communications technology company committed to advancing the art of software assurance. More information can be found at www.safecode.org.

Paul Kurtz (paul@safecode.org) is the Executive Director of SAFECode and a partner in Good Harbor Consulting LLC. Previously he was the founding Executive Director of the Cyber Security Industry Alliance (CSIA) and, prior to this, he served in senior positions on the White House's National Security and Homeland Security Councils in the US under Presidents Clinton and Bush.

The 10 Most Common Sins of Software Developers

Shay Zalalichin



We hear more and more nowadays about Cross-Site Scripting (XSS) security vulnerabilities being discovered in systems and websites. This is just one common example of vulnerabilities at the application level – security problems caused by development mistakes. These mistakes create code which is not secure and which can be exploited by malicious users. Many surveys suggest that, during the last few years, the trend in discovered security incidents has moved from the infrastructure to the application level.

The following are the top 10 most common mistakes made by software developers that result in dangerous code, as listed by the Open Web Application Security Project (OWASP) – <http://www.owasp.org/>.

1. Not Validated Input

There is a saying in the world of information security that: "User input is the source of all evil". It is no secret that numerous security vulnerabilities are caused by the fact that many systems do not handle inputs received from users properly.

An intruder who tries to exploit a lack of input inspection will insert malicious code which will disrupt the system and even utilise the trust relation with other internal systems to receive unauthorised access to the data.

2. Broken Access Control

Many information systems implement different mechanisms to allow users to access authorised information in an appropriate way (access to read, write etc.). These mechanisms are implemented in different levels and usually include examinations to enforce authorisation policy. Statistically, it seems that enforcing access control is a very complicated process, and implementation mistakes can have very significant implications. For example, a mistake in implementing an access control mechanism in Internet banking applications could allow users to transfer money from accounts that are not their own; the consequence is obvious.

3. Broken Authentication and Session Management

In a sensitive application system, strong and unique authentication must be used in order to provide users with information they are authorised to access with maximum security. The importance of this mechanism is clear – a mistake in implementation could allow external attackers to spoof other user identities and receive access to unauthorised information.

An authentication mechanism governs not only the way an application receives a username and password, but also the way a user is identified throughout his/her use of the system, using a mechanism called 'Session Management'.

4. Cross-Site Scripting (XSS)

This kind of flaw is considered the most complicated and difficult to understand, sometimes even by technical individuals. It exploits the fact that a system does not validate the input received from a user and uses it to create responses which are sent to the user. This problem, which at first glance seems to be innocent, supplies an applicative platform to carry out 'Site Defacement' and it is one of the popular ways to perform phishing attacks. With XSS, an application takes data supplied by the user and sends it to a web browser without first validating or encoding the content. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface websites, possibly introduce worms etc.

5. Buffer Overflows

In many cases where 'low level' development languages are used, such as C

and C++, input that is not validated could lead to another set of security problems called 'Buffer Overflow'. This is usually caused by a long user input which overruns its allocated memory. At best, this memory trampling will lead to memory leakage and crashing of the system; in the worst case, it can help an attacker to gain full control over the system and allow malicious code to run.

There are a number of low level programming functions which are considered dangerous. In addition, many systems that were not developed using low level languages but use third party libraries that were developed using these languages, may also be vulnerable to the same or similar problems.

6. Injection Flaws

Usually, information systems use different services (such as operation system, database etc.) to achieve their goals. The injection flaws exploit the 'Trust Relation' between the components of the system in order to change the way services are used to allow the attacker access to unauthorised information.

A simple example of this problem is 'SQL Injection'; systems which include web servers, applications and database servers could be vulnerable to this problem if they are using dynamic SQL queries. By injecting malicious input, the user could cause the application to run different queries which could change data or allow access to other information in the database.

7. Improper Error Handling

'Error Handling' mechanisms allow systems to handle extreme or unpredictable events in order to provide the user with a friendlier environment. From a security point of view, the failure of such a mechanism could lead to information leakage from the system or uncontrolled system crash and, as a result, endanger the information that system stores.

Thus, it is important to understand that attacking the system usually equates with finding ways in which a system cannot properly handle an attack and thus crashes in such a manner that it is endangered. This makes 'Error Handling' mechanisms a favoured target for potential attackers.

8. Insecure Storage

This flaw is extremely comprehensive. It involves the methods that a system uses to secure sensitive information (credit card details, passwords etc.) by encryption, key management, saving the data in a secure location etc. Failure of these mechanisms could expose sensitive data managed by the system to external factors or to internal

attackers who would receive unauthorised access to sensitive information.

9. Denial of Service

Denial of Service is a well known attack in the world of IT infrastructure and operation systems security. In the context of application security, it usually refers to the source code which is written in an unsafe way, allowing attackers to run the application in a 'greedy' manner that would consume system-critical resources to the point where the availability of the system would be affected.

An example of this problem is bringing the system to a situation where it runs heavy SQL queries against the database in order to damage the response time of the entire system.

10. Insecure Configuration Management

This flaw appears on the thin line between the infrastructure and application level and derives from the fact that application systems are eventually integrated on servers that run different infrastructure services, such as operation systems, application server, queue server etc., to provide the application with an operating environment. The problem is caused by the fact that the application is based on infrastructure services it receives but, in many cases, these infrastructure services are not hardened (or secured), or are hardened in a way that does not match the security level required by the application.

This failure could cause different security problems, from information leakage (for example, by simple directory browsing) to completely gaining control of the system by exploiting the non-secured administration interfaces of one of the infrastructure services.

To sum up...

This list is assumed to be the 'Top 10' of the most 'popular' problems caused by a non-secured development cycle. Application design, development and testing for these problems could significantly reduce the possible damage caused by the exploitation of the security weaknesses at the application level. In addition, every technology/development environment has its own unique problems which need to be solved as an integrated part of the development process.

Shay Zalalichin (shayz@comsecglobal.com) is Application Security Division Manager at Comsec Consulting.

Security Skills of Software Developers

Johan Peeters, Ken van Wyk and Frank Piessens



Today's business software is more capable, powerful and vital to our businesses than ever before. Despite this, however, we find it often wilts under some of the most basic security scrutiny – often succumbing to decades-old security defects. At the same time, attackers have grown more ruthless and profit-motivated than ever before. This confluence of extremes has resulted in unprecedented losses due to security breaches, and threatens to undermine consumers' confidence in business systems.

What are the underlying causes of this dangerous situation? One factor is that application developers tend to focus on providing functionality, not on security. This is in stark contrast to security personnel, who are paid to stop bad things from happening. That is, developers' contribution to the value chain is building enabling technology. This is exactly as it should be in a successful business, despite increasing malicious targeting and the exploitation of applications.

Nonetheless, these trends are forcing the software industry to re-examine the way software is built, shipped and deployed. The traditional approach of securing the perimeter and applications via security product add-ons such as firewalls and intrusion detection systems has proved to be inadequate. It is therefore apparent that developers will have to take their share of the responsibility for improving application security.

In this article we present a vision of how developers can work towards meeting security challenges without cutting themselves loose from their economic basis. This vision has inspired the curriculum design of the secure application development courses that we have been running since 2005 (<http://secappdev.org>). The course will be presented next in Leuven, Belgium, in March 2008. It is organised by

secappdev.org, a non-profit organisation which is dedicated to improving security awareness and skills in the developer community, in collaboration with K.U. Leuven, Solvay Business School and L-SEC, and is endorsed by ENISA.

Design and architecture

The challenge to the application development community is to create the conditions where a focus on functionality does not imply insecure applications. The key question is how far can a company be reasonably confident that its applications are secure if the developers are not security experts. We illustrate below how some architectural patterns separate security concerns from functionality.

• Isolation of Critical Modules

If the security-critical portions of the code can be isolated in a few discrete modules, they can be assigned to senior developers with good security skills and can undergo rigorous review. At secappdev we therefore examine architectural patterns that afford such separation of security concerns. One

well-documented pattern is AOP (Aspect-Oriented Programming). Its particular attraction is related to the observation that security is a cross-cutting concern: security guarantees need to apply across large parts of the code. Rather than tangling security concerns with business logic, they are separated into a distinct module that is woven into the business logic. However, it has transpired in some cases that AOP has introduced new vulnerabilities and should therefore be applied with the utmost caution.

• The role of the platform

This duality present in AOP clearly illustrates the role of the platform in application security: it is both the developer's friend and foe. On the one hand, judicious use of the infrastructure, especially of its security features, can greatly alleviate the burden of writing secure applications. On the other hand, platform elements may themselves be the source of vulnerabilities if used carelessly. So at secappdev, we spend a good deal of time studying the security features of the environment in which



applications run. Using inherent security services, whether of the OS, the language environment, middleware or some specialised bought-in components, is often the right thing to do; it is likely that more resources were available for their design and implementation than a typical application development project would have at its disposal.

• Use of the classical methods

Although ideally security concerns should be separate from business logic, sometimes this is difficult to achieve and some security requirements might need to be dealt with in the application. In such circumstances, pluggable service providers should be used where possible. Notable areas for which the developer can call on a service via an API that is provider-agnostic are authentication and cryptography. This isolates developers from some of the more arcane details and makes it easier to replace providers and algorithms as their sell-by date lapses; in the field of cryptography, for example, key lengths will eventually be judged inadequate and flaws will be found in cryptographic libraries or even in algorithms. DES and SHA-1 spring to mind.

Process

In the previous section we demonstrated that good architecture allows the development team to get on with the core activity, which is implementing the functionality. Nonetheless, security concerns still need addressing. But before that, we need to be sure that security concerns are complete and well understood. Conversely, we must avoid tilting at windmills and focus on real risks.



We believe that cost-effectiveness is key. No system is absolutely watertight and hence resources need to be spent wisely. Therefore, a good understanding of the application's security requirements is paramount. Security requirements are closely related to the attendant risks. These derive partly from the value of the assets the application should protect and partly from the resources an adversary is willing to commit to break through its defences.

Formal or informal approaches to the engineering of security requirements may be more appropriate depending on the different situations. One informal approach that has been widely adopted is based on abuse cases, while goal-oriented methods could bring more rigour and formalism to the process. The two approaches can be used in a complementary fashion, with the former providing the large brush strokes of the problem space and the latter drilling down in particularly sensitive areas.

While, in principle, requirements engineering precedes architectural notions, threat modelling assesses risk with the proposed architecture in mind. This may lead to architectural revisions or to additional requirements at a lower level of abstraction.

Our discussion so far is in sharp contrast to the commonly observed development process that concentrates effort on security assurance in the latter part of the cycle. Relying solely on 'Penetrate and Patch' at best delivers patchy security. Testing an application does not add security that was not already embedded by design. Nonetheless, it would be foolhardy to take a system into production without testing. Yet testing should not be consigned to the last phases of the development cycle nor should it be the exclusive preserve of specialists outside the project team. In the functional sphere, test-driven development (TDD) has resulted in quality improvements and

enhanced code maintainability. TDD is enabled by toolsets that integrate well with development environments and makes testing easy and fun.

In the security sphere, progress has not been so rapid, but tools are emerging that enable a tighter integration of assurance and development activities. An interesting class of assurance tools that has seen tremendous progress in the last couple of years is static code analysers.

Secure coding

In recognition of the need to address the software vulnerability problem, secure coding practices have quickly gained attention. As well as the many publications and training courses that exist, certification for secure programmers has also started to gain a hold. The SANS GIAC Secure Software Programmer examination, for example, was launched in March and ran for the first time in Washington DC in August 2007. The first chance to take this examination in Europe was in London, in December 2007. The second opportunity for Europeans to sit for the exam will be together with secappdev, the course offered in Leuven in early March. We encourage our students to take the examination by offering favourable rates.

Rather than simply preparing students for the examination, the secappdev course is complementary to its concerns. Secure coding has probably received more attention than any other aspect of secure software engineering so far. The secappdev course reverses this focus, for two reasons. Firstly, there is so much good material on secure coding that there seems little point duplicating it. Secondly, secure coding practices are specific to their environment. So, rather than, for example, dissecting the intricacies of buffer overflows or SQL injection and the specific APIs to avoid in a given language, the course explores threats of code injection or return-to-libc attacks and mitigation strategies.



A prime architectural concern is the choice of the technology on which the application is built and language is the most obvious factor to consider. Programming languages are not created equal and judicious choice is an important success factor in delivering a secure application. Thus it behaves application architects to have a working knowledge of the security characteristics of programming environments. A good understanding of underlying principles is essential; detailed knowledge is not required.

Security as a mindset

The focus should be on sound secure software engineering principles, rather than guidelines for use in particular technologies, and should go a long way towards ensuring that students acquire the appropriate knowledge and skills. Above and beyond the commitment to teaching principles rather than how to survive the vulnerabilities of the most recent technological fad, one should also try to instil a security mindset.

Developers' usual mode of operation is to make things work. Shifting attention to how they might break, specifically how a malicious agent might cause them to break, initially seems unnatural and even perverse. Yet the insight gained by studying failure modes leads to a deeper understanding of the technology and thus to improved designs.

The security mindset challenges received wisdom and explores what may happen beyond the boundaries of expected use. It encourages developers to exploit their subversive streak and is crucially important in the ongoing arms race between cyber-attackers and -defenders. As defenders, developers are on shifting sand, not only because they have to contend with black hats whose community has proved itself to be tremendously resourceful and inventive, but also because of the ever-quicken pace in the adoption of new technologies offering novel opportunities to shoot oneself in the foot.

A security mindset is not the result of formal study alone. Hence secappdev supplements

theory with practical examples and case studies. It encourages active engagement with security concerns through interactive sessions in small groups. There are also plenty of opportunities for informal discussions with faculty and other participants. The faculty is drawn from leading practitioners in industry and academia. The audience consists of developers with several years' experience. Hence participants are mostly senior developers from a wide range of backgrounds. The variety of responsibilities and industries represented on the course is enriching, as each individual lends a unique perspective to the security landscape.

Conclusions

We believe existing software development teams can produce products that meet

rigorous security requirements, but doing this requires more than just a passive desire to do better. Exposing development teams to the essential components of security is a crucial step towards that end. It is also vital to put that knowledge into practice, but understanding software weaknesses, attacks and mitigations is absolutely essential. What is more, properly presenting that knowledge must involve more than the simple superficial reciting of facts. It must include case studies, exercises and laboratory time so that developers can appropriately synthesise their new-found knowledge in the form of actionable results.

We believe the examples such as the curriculum at secappdev.org can be a significant component in helping to create a secure software community.

Johan Peeters (yo@johanpeeters.com) is an independent software architect and Program Director of secappdev.org.

Ken van Wyk is one of the founders of the Carnegie Mellon CERT/CC, and a lecturer on security technology. He is also a partner at KRVW Associates.

Frank Piessens is a Professor in the Department of Computer Science of the Katholieke Universiteit Leuven.

Course on Secure Application Development

3-7 March 2008, Leuven, Belgium

ENISA is endorsing an intensive secure application development course (secure software engineering principles and techniques for countering threats and vulnerabilities) for experienced software practitioners.

Organised by **Katholieke Universiteit Leuven** in partnership with **Solvay Business School** and **L-Sec** (Leuven Security Excellence Consortium), the course will take place from 3-7 March 2008 in Leuven, Belgium.

It is aimed at software architects, designers, developers, testers and technical project managers. A limited number of places is available.

For more information, visit:
<http://secappdev.org/2008/FrontPage.html>

Leading the Way to More Secure Software

Wesley Higaki



Around the world, we have grown to depend on the Internet, computers and networks almost like a utility such as electricity, water and the telephone. Businesses routinely use the Internet to conduct transactions with customers and partners. Individuals use the Internet to communicate and share information. Governments and critical infrastructure providers have improved their ability to serve their citizens through the use of the Internet and associated computing and communications technologies.

However, the Internet is a hostile environment where cyber attacks and exploits on systems and networks are commonplace. In the face of these threats, customers are becoming increasingly concerned about the integrity, security and reliability of the software they use.

Software vendors recognise the need to reduce software defects which lead to exploitable vulnerabilities and to improve resistance to an attack. Individual software providers have implemented methods and tools for developing and delivering more secure and reliable software. Monitoring statistics indicate that the efforts of software providers to improve the security of their software are working. However, the attacks grow unrelentingly in complexity and sophistication, so constant improvement is necessary. There are some fundamental best practices that have been identified and which could be adopted to improve the security and integrity of software.

Security Training

Security awareness, education and training are the first steps towards improving the

security of software products. Software managers need to understand both the costs and benefits of a secure development programme. They must then allocate appropriate resources to meet the needs of their customers. Developers must learn and exercise secure development and test techniques.

One training resource Symantec has used effectively comes from Purdue University's Center for Education and Research in Information Assurance and Security (CERIAS) (<http://homes.cerias.purdue.edu/~pmeunier/secprog/sanitized/>), where one can find a series of secure programming training modules that cover a wide range of topics. This training material is constantly updated to leverage the latest techniques for mitigating risks and reducing defects.

Secure Development Processes

- **Setting security requirements** at the beginning of a product release cycle sets the tone and establishes the need to address security concerns. These requirements must be addressed by the secure design of the product. Secure coding practices should be exercised during the implementation phase and security testing should be performed at the same time as the rest of the product testing. Security reviews are an essential part of the secure development process as they help ensure consistency throughout the process. With the ever-increasing complexity of attacks, developers must continuously augment and enhance their techniques to address new threats.

- **Penetration testing** is performed by security experts to try to 'break into' software by exploiting vulnerabilities discovered in the code. These intensive tests can help uncover vulnerabilities that might not be discovered any other way. Fuzz testing is also a technique that is growing in popularity to combat issues such as buffer overflow vulnerabilities. Fuzz testing focuses on testing the interfaces to the software by trying a wide variety of inputs to exploit input handling errors.

While we strive to develop and deploy security defect-free code, vulnerabilities are being discovered in released code and need to be repaired in the field. Timely and effective **patching processes** and technologies play an important role in protecting deployed software. Certainly as we issue patches to repair security defects, software providers are aware of the burden this can place on customers, especially

those managing large, distributed deployments. Making patching easy and using automated tools helps to reduce this burden while it provides the protection customers need.

Development Security Tools

Within the software development community, growing attention has been drawn to the development and use of code analysis tools. These tools scan source and binary files for potential security defects. This automation is proving to be valuable in improving the reliability and efficiency of secure software development. Today, although considerable human intervention is still required to make full use of static code analysis tools, the tools are constantly improving. The use of security testing tools and standard testing tools to identify security defects is also growing. There are now fuzz testing tools available to improve the speed and efficiency of these tests.

Promoting Best Practices

While individual companies have implemented effective methods for developing and delivering more secure and reliable software, until now there has been no co-ordinated effort to build upon this work and promote best practices to advance secure software development more broadly. A group of leading information technology companies has recently formed the Software Assurance Forum for Excellence in Code (SAFECode), a non-profit organisation (www.safecode.org/) dedicated to developing and promoting secure software development practices. Promoting best practices and developing new, effective techniques collaboratively can be an effective means to address new threats and improve software security for all customers.

Leading software developers promote secure software development practices to demonstrate that the software industry takes this issue seriously and is encouraging all developers to employ best practices to learn from each other and improve the security of their software. Governments, critical infrastructure providers, enterprises and consumers will benefit from this effort because sharing best practices will improve the integrity, security and reliability of the software that they depend on.

Wesley H. Higaki (whigaki@symantec.com) is the Director of Product Certifications in the Office of the CTO at the Symantec Corporation.

Providing Assurance for Security Software – Insights into the Common Criteria

David Ochel



The Common Criteria for Information Technology Security Evaluation (CC) standard, version 3.1, adopted as ISO/IEC 15408, has succeeded the Information Technology Security Evaluation Criteria (ITSEC) as a means of providing independent assurance that the security functionality of a (software) product actually works. Although there are a number of perceived limitations to the Common Criteria approach, more effective alternatives to evaluate and certify security software are not readily available, while some of the perceived limitations can be addressed by embracing the CC in the larger context of risk management. This article will demonstrate how the CC can be effectively used to provide assurance of a product's purported security functionality, and describes a means of maintaining a high-level view of the purpose of a security evaluation.

At the time of writing, 13 of the 27 European Union (EU) Member States are also members of the international Arrangement on the Recognition of Common Criteria Certificates (CCRA), which stipulates that its signatories mutually recognise certificates up to Evaluation Assurance Level 4 (EAL4) that have been issued by each other, unless a matter of national security prevails. France, Germany, the Netherlands, Spain and the UK actually operate schemes for the issuance of certificates, Sweden is in the process of joining them, and the other nation members have agreed to simply recognise the validity of certificates issued by other nations. Non-EU members of the arrangement include Australia, Canada, Japan and the United States. Altogether there are a total of 24 participating countries. In addition, a Mutual Recognition Agreement (the 'SOGIS-MRA') exists within the EU whereby members recognise certificates issued by other states at all assurance levels.

'Certificate Authorising' members	'Certificate Consuming' members
Australia and New Zealand Canada France Germany Japan Republic of Korea Netherlands Norway Spain United Kingdom United States	Austria Czech Republic Denmark Finland Greece Hungary India Israel Italy Malaysia Singapore Sweden Turkey

CCRA members as of 8 January 2008 Source: www.commoncriteriaportal.org

The national schemes participating in the CCRA are operated by institutions representing the respective governments (as opposed to private schemes, which are not allowed to join the arrangement), such as the Organismo de Certificación de la Seguridad de las Tecnologías de la Información of the Centro Criptológico Nacional in Spain and the Bundesamt für Sicherheit in der Informationstechnik (BSI) in Germany. These agencies accredit and license commercial Information Technology Security Evaluation Facilities or, simply, 'labs', which perform the actual evaluation of products against the Common Criteria. While the labs charge the sponsor of an evaluation – typically the product vendor – for their services, they have to maintain

independence in their decisions. The evaluation results are recorded in detailed reports and are subject to scrutiny by the government-run Certification Body of the scheme, and only after an in-depth review cycle will the Certification Body issue a certificate for a product that has been successfully evaluated.

The member states of the CCRA participate, more or less actively, in the ongoing interpretation, development and improvement of the Common Criteria. The standard itself consists of three volumes: an introduction, a catalogue of Security Functional Requirements and a catalogue of Security Assurance Requirements. The functional requirements comprise a toolkit



CC 3.1								
Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		1	1	1	1	1	1
	ADV_FSP	1	2	3	4	5	5	6
	ADV_IMP				1	1	2	2
	ADV_INT					2	3	3
	ADV_SPM						1	1
	ADV_TDS		1	2	3	4	5	6
Guidance documents	AGD_OPE	1	1	1	1	1	1	1
	AGD_P RE	1	1	1	1	1	1	1
Life - cycle support	ALC_CMC	1	2	3	4	4	5	5
	ALC_CMS	1	2	3	4	5	5	5
	ALC_DEL		1	1	1	1	1	1
	ALC_DVS			1	1	1	2	2
	ALC_FLR	(ALC_FLR subactivities optional at all EALs)						
	ALC_LCD			1	1	1	1	2
	ALC_TAT				1	2	3	3
Security Target evaluation	ASE_CCL	1	1	1	1	1	1	1
	ASE_ECD	1	1	1	1	1	1	1
	ASE_INT	1	1	1	1	1	1	1
	ASE_OBJ	1	2	2	2	2	2	2
	ASE_REQ	1	2	2	2	2	2	2
	ASE_SPD		1	1	1	1	1	1
	ASE_TSS	1	1	1	1	1	1	1
Tests	ATE_COV		1	2	2	2	3	3
	ATE_DPT			1	2	3	3	4
	ATE_FUN		1	1	1	1	2	2
	ATE_IND	1	2	2	2	2	2	3
Vulnerability assessment	AVA_VAN	1	2	2	3	4	5	5

Composition of EALs. Higher component numbers represent increased evaluation effort.

of sorts that allows product vendors to select from templates in order to describe the security functionality that their product implements, thus providing for a certain amount of comparability between evaluated products (as described in the products' Security Targets). These 'templates' also allow for a common language that consumers can use to describe their desired security functionality to vendors in Protection Profiles. Assurance requirements formulate the required evaluation activities that are performed to demonstrate a product's proper working. These assurance requirements come in pre-packaged Evaluation Assurance Levels (EALs) with increasing levels of scrutiny applied (see table above). For example, compared with EAL3, an EAL4 evaluation mandates an increased level of detail for the design documentation that is provided for evaluation requirement ADV_TDS.3, and introduces the requirement ADV_IMP.1 for inspection of source code samples to verify the correct implementation.

To achieve 100% security assurance in a software product is impossible. Consequently, the goal of the Common Criteria is to demonstrate, at a reasonable (and comparable) level of effort and detail, that measures have been undertaken to decrease the likelihood of malfunctions in security functionality and to diminish the existence of vulnerabilities that would allow the circumvention of this security functionality. In order to achieve this, evaluation efforts include the detailed analysis of:

- Architecture, design and interface specifications
- Source code (at EAL4 or higher)
- Product manuals that instruct consumers on secure operation
- Functional tests performed by the vendor and the lab
- Development life-cycle, physical and logical development security, and secure delivery of the product.

During all of these evaluation activities, the lab will be on the lookout for potential vulnerabilities in the product, and will perform a Vulnerability Assessment that seeks to verify the existence (or non-existence) of hypothesised vulnerabilities in the product. Evaluation work within the CCRA is based on a harmonised Common Evaluation Methodology (CEM) that provides detailed work instructions for the labs.

CC Evaluation Limitations

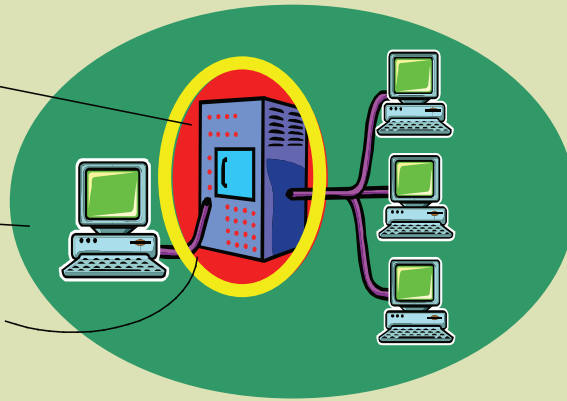
Undergoing a Common Criteria evaluation usually requires a significant commitment of resources, both monetary and human, especially for an initial evaluation. Design and other documentation may have to be created or augmented, tests may have to be designed and performed etc. A typical first-time evaluation of a product with medium complexity may take between six and twelve months of involvement, with reduced time spans and efforts for subsequent re-evaluations of new product releases. The fact that a Common Criteria certificate is only valid for a specific release and patch level is considered an annoyance by some, but therein lies the nature of the beast. If changes to the product implementation are made, the lab and the Certification Body can no longer vouch for the product behaving the same as originally tested, inspected and validated.

Another purported limitation of the Common Criteria is the usability of a product while in its evaluated configuration. With the increased complexity of a product and its exposed interfaces, it becomes more likely that an evaluation will be performed with a limited scope, and that stipulations will be made as to how the product may be used. For example, consumers may be instructed to disable a specific function or interface when deploying the product. This may have one of several justifications. The vendor may have chosen to exclude it from the evaluation because it is not a 'mainstream' function or interface, and therefore the effort required to extend the analysis to cover it might have been uneconomical. Alternatively, enabling this function or interface might allow circumvention of a product's security functionality. While schemes usually require that the typical security functionality for a specific type of product has to be evaluated, such restrictions are common for less-used or non-essential parts of the product. For example, a firewall product would hardly be accepted for evaluation if its filtering mechanism was excluded from evaluation. But instructing users to only use the command line interface and to disable the graphical user interface for managing the firewall's configuration might be acceptable.

Scope of lab evaluation
CC analysis (red)

Consumer's Information
Security Management
domain (green)

Integration (yellow) is still
consumer's responsibility!



Integrating Common Criteria certified products into a system

The role of Risk Management

Organisations using evaluated products have the option of addressing these limitations as part of their risk management programme. After all, security products, such as a firewall, or the security functionality in general-purpose products, such as discretionary access control mechanisms in an operating system or database, provide countermeasures to threats that have been identified in an organisation's operating environment. Common Criteria certificates for these products may be desirable because they provide a defined level of independent assurance that these countermeasures actually work when employed correctly.

However, this does not relieve an organisation from having to ensure that a product will be properly integrated into their larger systems.

Nor should it force the organisation into an 'all or nothing' approach, using only the evaluated version of a product in its evaluated configuration. The decision to apply a security patch to a product, thereby invalidating its Common Criteria certificate, should be risk-based. Organisations should ask: In our environment, is it more likely that the patch compromises security functionality that was determined to work correctly in the evaluated version, or that an

attacker will exploit the unpatched vulnerability in the certified version of the product? A similar approach should be considered for non-evaluated functionality or interfaces – that is, is the risk of using them (and by doing so, violating the stipulations of the product's evaluated configuration) outweighed by their benefit to the organisation?

Even if the answer to these questions is to use a product outside of its certified scope, a Common Criteria certification can still provide benefits in that the overall product architecture and a high percentage of the functionality will have been assessed and tested, and the product's development environment will have been scrutinised. Therefore, certification still provides an organisation with a jumpstart in terms of establishing trust in the provided security functionality. A Common Criteria evaluation provides an assurance baseline, and then only the delta not covered by the evaluation will need to be scrutinised by the organisation itself.

David Ochel (david@atsec.com) is a Principal Consultant and Evaluator with atsec information security.

From our own Experts

ENISA Position Papers: Social Networks, Reputation Systems, and Botnets

Giles Hogben



ENISA has recently published three Position Papers representing expert opinion on important emerging Network and Information Security risks. Each paper has been produced by independent expert groups from industry, academia and Member State governments who have been selected for their expertise in the relevant area. This article summarises the findings of these groups. The full papers can be downloaded from: www.enisa.europa.eu/pages/position_papers.htm.

Security Issues and Recommendations for Online Social Networks

Social Networking Sites (SNSs) are expanding at a dramatic rate. For example, as of June 2007, MySpace was the most visited website in the US with more than 114 million global visitors, representing a 72% increase on 2006. In this position paper, we start from the idea of Social Networking as a positive social phenomenon. We look at the security threats to Social Networking and make recommendations on how to address these to gain the full benefits offered by SNSs.

SNSs may be seen as informal but all-embracing identity management tools, defining access to user-created content via social relationships. They provide:

- Tools for posting personal data into a person's 'profile' and user-created content linked to a person's interests and personal life
- Tools for personalised, socially-focused interactions, based around the profile (e.g. recommendations, discussion, blogging, games, events organisation)
- Tools for defining social relationships which determine who has access to data



There is even evidence that SNSs contribute to increased self-esteem and satisfaction with life (what social scientists call 'Social Capital') and even that it can help to reduce crime.

Threats



The commercial success of the multi-billion Euro SNS industry depends heavily on the number of users it attracts. Combined with the strong human desire to connect, this encourages design and online behaviour where security and privacy are not always the first priority: users are often not aware of the size of the audience accessing their content. The sense of intimacy created by being among digital 'friends' often leads to inappropriate or damaging disclosures.

Social Networking may be seen as a 'digital cocktail party'. In general, the more contacts you have, the more popular you are, and the more influence you have. However, compared with a real-world cocktail party, SNS members broadcast information much more widely, either by choice or by mistake. For example, a brief survey of popular SNSs shows several people openly publishing answers to 'surveys' with questions such as:

- Have you ever stolen money from a friend?
- Have you ever been in a fist fight?
- Have you ever cheated on a boyfriend/girlfriend?

Some of the main threats identified by ENISA and the expert group are:

- **Digital Dossiers:** The network never forgets – what happens when the huge warehouses of seemingly transient personal information provided by SNSs fall into the hands of blackmailers and spammers?
- **Face Recognition:** Images enable unexpected linking to apparently anonymous personal data, especially

when combined with uncontrolled 'tagging'. Some SNSs even allow members to tag faces in images with another person's e-mail address.

- **Social Engineering Attacks on Enterprises using SNS:** Some information is necessary to enter an online community but often the privacy settings are neglected and therefore the threshold for gaining information which could be used in a social engineering attack is very low. Several professional SNSs publish information on lists of employees. For example, one SNS search results page lists employees currently or previously working at Barclays Bank, which could be very useful to someone collecting information for a social engineering attack on an enterprise.

Other SNS threats include Spear Phishing using SNSs, Reputation damage through ID theft, Stalking and Cyber-bullying. A complete list can be found in the Position Paper.

Recommendations

This paper makes 19 recommendations – some of the most important ones are:

- **Review and Reinterpret Regulatory Framework:** Social Networking did not exist when current legislation (especially data protection law) was created. Clarification or even modification is needed, in particular of the Directive 2002/58 on privacy and electronic communications.
- **Increase Transparency of Data Handling Practices:** Examples of areas requiring greater transparency for SNSs are:
 - What is done with data on profile visits?
 - What data is transmitted to widget providers?
 - What are the secondary purposes of the processing of profile data?
 - A clear description of the difference between the 'deactivation' and the 'closure' of an SNS profile.
- **Awareness-raising & education:** Recommendations include the 'real-time' education of users, campaigns for schools, security best practice training for software developers and security-conscious corporate policy for SNS usage.
- **Discourage the banning of SNSs in schools:** Instead favouring co-ordinated campaigns to educate children, teachers and parents in the safe usage of SNSs.
- **Promote Portable Networks:** Allow users to move, control and syndicate their own data and privacy preferences between SNSs.



The paper also identifies some important emerging trends which deserve further research:

- **Convergence with virtual worlds and 3D representation:** Given the similar aspirations of many SNSs and virtual world users and the extra functionality offered by virtual worlds (e.g. Second Life), a widespread convergence between the two seems only a matter of time. Such applications may introduce new security threats such as those related to virtual world economics.
- **Misuse by criminal groups:** While there are many benefits to a tool which allows like-minded individuals to discover and interact with each other, this feature can also have negative implications. We recommend research into the extent and nature of illegal activity on SNSs.
- **Online presence:** Increasing amounts of information and tools are available relating to online presence (whether someone is currently online and logged into a particular site), or even physical location, which is typically revealed more in mobile-based SNSs. More research is needed into the privacy and security implications of online presence.

Reputation-based Systems: a Security Analysis

Electronic reputation is becoming as valuable an asset as traditional offline reputation. As new applications embrace reputation-based systems, the value of online reputation will continue to increase – and online reputation will be the target of attacks. Reputation allows users to form an expectation of behaviour based on the judgements of others, bringing the significant economic and social benefits of



being able to trust people (or systems) not directly known to the user. As well as discouraging misbehaviour because of the penalties implied by a bad reputation, reputation can also encourage good behaviour, as users seek to establish a good reputation and to benefit from it.

We describe four use-cases for reputation: online markets (such as eBay), peer-to-peer networks (e.g., for bandwidth management), anti-spam techniques and public key authentication (web-of-trust). From these, we have derived the main threats and attacks against reputation systems. This has led to a set of core recommendations for best practice in the use of reputation systems. The most important threats described are:

- **Whitewash attack:** The attacker resets a poor reputation by rejoining the system with a new identity. Systems that allow for the easy change of identity are most vulnerable.
- **Sybil attack (i.e., pseudospoofing):** The attacker creates multiple identities

(sybils) and exploits them in order to manipulate a reputation score.

- **Impersonation and reputation theft:** One entity acquires the identity of another entity (masquerades) and consequently steals his reputation.
- **Denial-of-reputation:** Attacks designed to damage an entity's reputation and create an opportunity for blackmail in order to have the reputation cleaned.

- **Privacy threats for voters and for reputation owners:** Votes are more likely to be accurate if voters are anonymous. Lack of privacy for reputation-owners tends to cause whitewash attacks (resetting reputation) and inaccurate reputation scores.

- **Threats to ratings:** There is a whole range of threats which exploit the features of metrics used by the system to calculate reputation ratings from single scores.

Recommendations

- **Develop reputation systems which respect privacy requirements:** Using reputation brokers, it is possible to offer anonymity for reputation-owners. Privacy for reputation voters also improves the accuracy of scores.
- **Provide open descriptions of metrics:** A reputation metric is the algorithm used to calculate a reputation score from the individual votes collected. They vary in complexity, with some algorithms taking

into account so-called second-order factors such as the reputation of the voters themselves. As with most security algorithms, security is increased if reputation metrics are not kept secret but are subject to scrutiny. This also helps people to judge whether to trust the results.

- **Differentiation by attribute and individualisation as to how the reputation is presented:** A given reputation system should allow a user to customise the reputation according to different attributes (i.e. different aspects/assertions about the reputation subject), and to set a threshold for each of them.

- **Encourage research into:**
 - a. **Common solutions to threats against reputation-based systems:** Despite the variety of use-cases for reputation-based systems, they often show vulnerabilities to similar threats and attacks – common solutions to defeat these should be investigated.

- b. **The management of global reputation:** How can a user gain control and/or awareness of his overall electronic reputation when it is composed of fragments scattered across the Internet?

- c. **Use of weightings in reputation metrics:** Second-order reputation using score weightings can improve the resistance of the metric to attacks.

- **Research into and standardisation of portable reputation systems:** One possibility is to integrate reputation into authentication transport standards, e.g. OASIS Security Assertion Markup Language (SAML) Authentication Context.

- **The importance of automated reputation systems for e-Government:** Reputation is informally already an important component of several high-assurance systems such as document issuance and security clearance processes. Automatic ad hoc reputation systems provide scalability and flexibility which is not present in existing systems (such as Public Key Infrastructure (PKI) systems, for example). Policy-makers are therefore encouraged to investigate using state-of-the-art reputation-based systems in e-Government systems. Governments should also start investigating the impact of online reputation systems on existing legislation such as the EU privacy directives.

Botnets – the Silent Threat

Bots are lightweight programmes that are installed silently without any user intervention. A botnet is a network of computers on which a bot has been installed, and is usually managed remotely from a Command & Control (C&C) server. The main purpose of botnets is to use hijacked computers for fraudulent online activity; they are managed by a criminal, a group of criminals or an organised crime syndicate.

Typically botnets are used for identity theft, unsolicited commercial e-mail, scams, Distributed Denial of Service (DDoS) attacks and other fraud. It is estimated that more than 6 million infected computers worldwide are connected to a botnet, with China, the US, Germany, Spain and France the top five countries for the number of infected computers. Most owners of infected computers do not know that their machines have been compromised.

The criminal organisations behind the implementation of this new online threat are well organised. They employ software developers, they buy and sell infrastructure for their criminal activities and they recruit people (mules) for money laundering to hide their identities. They have the technical resources to continually improve their attacks – conditions that make online fraud more successful than offline fraud. Lack of user security awareness combined with the common habit of using old (sometimes pirated) and unpatched operating systems increase the success of criminal exploitation.

Botnets represent a steadily growing problem threatening governments, industries, companies and individual users with devastating consequences that must be avoided. Urgent preventive measures must be given the highest priority if this criminal activity is to be defeated. Otherwise the effect on the basic worldwide network infrastructures could be disastrous.



Recommendations

Non-technical recommendations

The solutions to the non-technical problems can be divided into three different initiatives, according to the actors: Government, law enforcement agencies and private companies, and the end-user.

- **Involving the Government:** Whether botnet activity is punishable depends on the precise activity and on the law that can be applied. Agreement is needed within the EU and beyond to prosecute cyber crime in a consistent and co-ordinated way (for example in line with the European Convention on Cybercrime which has still not been ratified by all signing countries). Too few decision-makers are sufficiently aware of the extent of the botnet problem and the consequences of inaction.
- **Better co-operation between Law Enforcement Agencies and private companies:** (ISPs, financial entities, security companies etc.), working for a better dialogue and helping each other to detect, prevent and react to botnet incidents.
- **User awareness:** Everyone who uses a computer connected to the Internet should know and understand the threats that could affect him/her. Proper education and awareness about security measures should be included in school curricula, in public service announcements on television and the Internet and other awareness-raising initiatives.

Technical recommendations

Technical solutions to the problem of botnets include:

- **Secure operating systems and software applications:** Vendors should make strenuous efforts to increase the security of their products and, for example, improve the update and patch management process. Investment should

be encouraged with public and private funding for secure software development.

- **ISP co-operation:** ISPs are key to the solution, since they can detect and block botnet communication. Of course they would need to inspect the user's traffic, which could lead to privacy issues. Guidance on this from a privacy authority would be welcome, similar to the Article 29 Working Party's opinion 118 on e-mail filtering.
- **Give law enforcement agencies the capability to clean botnets:** Almost any botnet can upload and force all its zombie computers to execute a specific programme. This programme could be a malicious code removal tool that uses the same technology for good purposes. However, given the privacy implications and potential side-effects, this should not be considered an option at this time.

The detection of Botnets:

- **At ISP level:** Some products analyse DNS queries to detect whether a computer has been infected by malicious code. Then, in order to detect botnet traffic, ISP administrators need to combine a signature-based method (e.g., based on DNS or HTTP) with a heuristic one, for instance a flow-based method (analysing where the user is connecting), which looks for anomalous connections.
- **At LAN level:** As many worms try to infect nearby computers, a local honeypot (a computer system set up as a trap for attackers) could help with the early detection of any malicious software that is trying to infect all the computers in an organisation. Local administrators play a key role since they can detect any infection and take appropriate action.
- **At computer level:** There are some hints as to whether malicious code is running on a computer, for example, strange process names and slow connection to the Internet, which might mean that the computer is sending spam or participating in a DDoS attack.

All the hints explained both here and in the paper are only valid if the computer has not installed a rootkit, because a rootkit would hide all such indicators to enable it to survive in the system without being detected. There are, however, special software tools (rootkit detectors) that help to uncover the existence of rootkits on infected machines.

Giles Hogben (giles.hogben@enisa.europa.eu) is an expert in the Network and Information Security Policy unit of ENISA.

Towards a European Information Sharing and Alerting System

Marco Thorbruegge



Several publications point out that, for various reasons, the computers of home-users and Small and Medium Enterprises (SMEs) are the most popular victims of targeted attacks. It is comparatively easy to incorporate these users' computers into botnets, use them as obfuscated paths for launching attacks by hackers, as proxies to send spam or to enrol them as repositories for spreading viruses and worms. At the same time, SMEs are important to Europe's economic growth. However, due to their size, SMEs rarely employ dedicated security personnel, so the protection of their information assets is often left to non-security experts.

In its Communication to the Council, Parliament, Economic and Social Committee and the Committee of the Regions (COM(2006) 251), the European Commission emphasises that public authorities in Member States and at EU-level have a key role to play in properly informing home-users so that they can contribute to their own safety and security. The Communication points out that *"individual users need to understand that their home systems are critical for the overall 'security chain'".* The Commission also recognised that the possibility of facilitating *"effective responses to existing and emerging threats to electronic networks"* should be explored. Acknowledgement of these needs prompted the Commission's request to ENISA to *"examine the feasibility of a European information sharing and alert system (EISAS)"*, highlighting the role of ENISA in fostering a culture of Network and Information Security (NIS) in Europe. ENISA accepted this request and embarked on this study.

EISAS at a glance

In the context of this study an information sharing system is a mechanism that shares information about good practice in NIS, and potentially also about more recent threats and attacks (alerts & warnings), with home-users and SMEs. It was already known that some EU Member States have such mechanisms in place and that other activities exist. However, it was also known that not all EU citizens are covered, and the main purpose of the feasibility study was to identify the gaps and to define the role the European Union might play in that field.



NIS information for citizens and SMEs is important

There is already a lot going on in the Member States, but ...



Motivation: where are the gaps in the coverage of European citizens with adequate NIS information?

Setting the scene

In order to provide thorough and responsible advice to the European Commission, ENISA first conducted an analysis of the current state of play in both the public and private sectors in all EU Member States, and identified possible sources of security information which could potentially contribute to a Europe-wide Information Sharing and Alert System. The findings of this analysis led to the development of a scenario to both address the lack of available NIS information in some Member States and provide a (yet-to-be determined) added value to existing information sharing systems in other Member States. Ideally such an EISAS would also build on these existing systems, firstly to avoid the duplication of effort and competition, and secondly, to benefit from the lessons

learned and the good practices that these (national) systems can provide. ENISA therefore produced two inventories. The first lists existing information sharing systems in Europe (with a special emphasis on home-users and SMEs as a target group); the second lists publicly available sources of NIS information. This latter inventory was prepared by the ENISA Ad hoc Working Group "CERT Services" in 2006.

In addition, existing material was reviewed that could potentially contribute to the study, including the final reports of the EWIS (European Warning & Information System) project that was carried out by the European Commission in 2001/2002, and ENISA's own study into "CERT co-operation and its further facilitation by relevant stakeholders" in 2006.

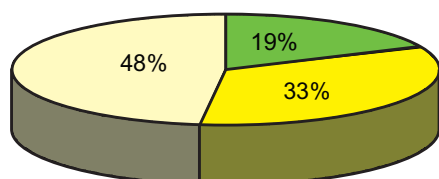
The three aspects of feasibility

As well as examining the technical feasibility of an EISAS, a broader approach was necessary to ensure the acceptance of such a system by EU Member States and, last but not least, the target audiences. Thus, the study examined the question of feasibility from three angles:

- **Technical/organisational aspect:** the technical/organisational feasibility of an EISAS, such as components and workflows etc.
- **Political aspect: the political feasibility of an EISAS. This can be reduced to the question:** will Member States accept and support the proposed solution?
- **Social/Cultural aspect:** the feasibility of achieving real impact by successfully, effectively and sustainably raising NIS awareness among home-users and SMEs. This angle is the most crucial, as the target groups have specific perceptions and needs (for example, language), not least the fact that they are, in most cases, non-security experts.

In order to analyse the current state of play, the two inventories produced were examined by asking eight questions, for example, "What activities exist in the Member States that explicitly target citizens and SMEs?", "What type of information is shared among citizens and SMEs?" and "How is the information distributed". The answers were used to derive the most feasible role the European Union could play in this field.

ENISA was supported by an Expert Group comprising members from national and other information sharing activities nominated by EU Member States and ENISA's Permanent Stakeholders' Group (PSG). The group's main tasks included the provision of information about the information sharing activities for which they are responsible (thereby making the vision of an EISAS more concrete), assessment of the two inventories prepared by ENISA, and the provision of advice both during the study and about the methodology. The group worked mainly via email; one face-to-face meeting was held in Brussels in April 2007.



■ MS w. dedicated ISAS ■ MS with non-dedicated ISAS
■ MS without ISAS

48% of the EU Member States do not have any information sharing activity for home-users and SMEs.

Problem: how to reach out to the end-user?

The analysis and discussions with the Expert Group offered interesting insight into the difficulties in adequately addressing home-users to achieve real impact. These problems have not yet been completely solved.

Some of the findings of the study are listed below:

- end-users and SMEs should be addressed in their **native language**.
- messages (warnings, good practice documents etc.) should be phrased semantically in an **understandable way** (addressing the non-expert).
- the method of information dissemination should be thoroughly planned, i.e., in addition to web pages and mailing-lists, other communication channels should be examined such as podcasts, RSS-feeds, traditional media etc. to make it as **convenient** as possible for the end-user/SME to obtain information.
- information **overflow** should be **avoided** and it should be thoroughly planned what and when to publish.
- information disseminated to end-users/SMEs must be **trusted** by the recipients if it is to be accepted (on average, national governments are already trusted by end-users/SMEs).
- information should be disseminated as **close** to end-users/SMEs as possible.
- the information sharing system should be **advertised**, as systems will only be used when people know they exist.

Conclusions

The findings of the study do not suggest that a centralised Europe-wide information sharing system is the most feasible scenario. Instead the European Union should build on

existing resources to foster the establishment of information sharing systems at the national level in Member States. The study concludes by making four recommendations for a potential role for the European Union:

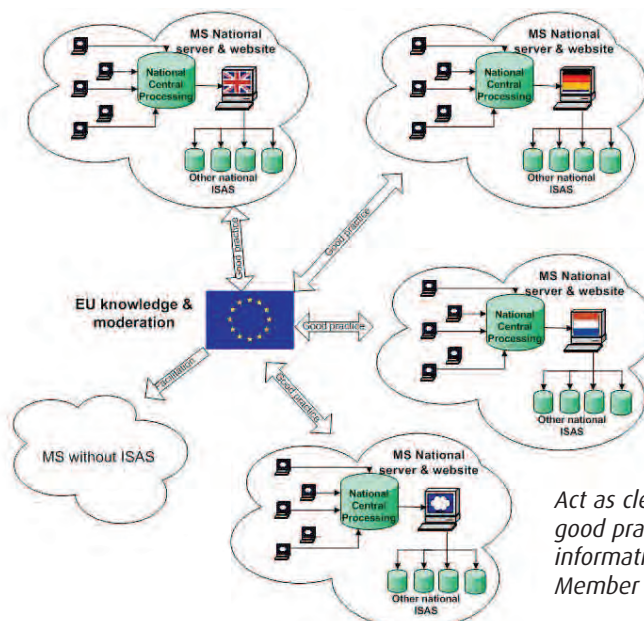
- Act as clearing house for good practice for national Information Sharing and Alert Systems (ISASs)
- Support new national ISASs
- Foster dialogue among existing national ISASs
- Analyse and review practice, components and processes to optimise information sharing for existing ISASs

Next steps

In order to demonstrate the feasibility of the proposed EISAS scenario, the study report encourages a 'proof of concept'. This means that a small group of experts, drawn from existing national information sharing activities, should be assembled and work in a number of meetings facilitated by the European Union to produce a set of deliverables, taking into account as much existing material and expertise as possible. The proof of concept should both prove that co-operation among existing activities produces comprehensive results and lay the base for future work in the area. Once assembled, the group should work independently and be responsible only to itself; however, to assist the process, the study report offers suggestions as to how it might proceed.

The full report about the feasibility study for a Europe-wide information sharing and alerting system can be found on the ENISA website.

Marco Thorbruegge (marco.thorbruegge@enisa.europa.eu) is a Senior Expert in ENISA's Computer Incident and Response Handling unit.



Act as clearing-house of good practice and support information sharing in the Member States.

Food for Thought Stop using the Traffic Analogy!

Pernilla Skantze and Nick Coleman



Why is it that everyone persists in using the analogy with car safety or traffic regulation when they want to make comparisons with information security? Cars and computers actually have very little in common.

Not only would it be difficult to re-start a car after a crash by closing all the windows and then rebooting it, but the numerous ways of using information technology (IT), as well as the speed of its development, make for big differences.

While roads are used fundamentally in the same way as they were when they were first constructed, today the way we use computers and electronic communications has very little in common with how we used the technology 20 years ago.

Just imagine how small and fast today's cars would be if automotive technology improved with the same speed as that of micro-processors! While we can expect roads and the cars travelling on them to work the same way in 15 years' time as they do now, we have very little idea about how we will be using IT and electronic communications in the 2020s.

The speed of development of computing also creates completely different problems when planning for information security. Where does the concept of peer-to-peer sharing fit in the car analogy? This is another

good reason to stop using the traffic analogy and to find more appropriate analogies for information security in today's world.

One idea would be to look at information security as a sort of health care system for IT – a system that addresses society at many different layers, is prepared to handle ageing infrastructures as well as new ones, and to treat all kinds of diseases and injuries, a health care system which requires public and private sectors to come together with citizens and which includes all of these stakeholders in planning for protection and cure.

To protect our health we are taught to wash our hands to avoid the spreading of germs (also known as peer-to-peer sharing), to eat a variety of food and to avoid smoking in order to keep well.

At the same time we have hospitals and professional help for those who do nevertheless fall ill, and there are incentives for industry and academia to develop new drugs and eradicate disease (which parallels antivirus companies).

All stakeholders are encouraged to fulfil their own responsibilities within the health care system while at the same time there are also control functions to test new products and ensure that professional care is up to standard.

Certainly health care is not the only possible new analogy for IT security but it might stimulate a more relevant discussion than thinking of driving licences, road works and airbags!

We hereby invite all contributors and readers of EQ to challenge us with new and thought-provoking analogies in the future!

Pernilla Skantze (pernilla.skantze@enterprise.ministry.se) is a lawyer specialising in IT-related issues, working for the Swedish Ministry of Enterprise, Energy and Communications, and a member of the ENISA Management Board.

Nick Coleman (nick@n-coleman.com) is the former Head of Security Services for IBM across Europe, the Middle East and Africa. He is currently the Independent Reviewer of information assurance and security to the UK Government. He is also a member of ENISA's Permanent Stakeholders' Group.

Enhancing ENISA's Impact

Hotel Sofitel Athens,
Athens International
Airport, Greece
22-23 January 2008

ENISA invites all stakeholders to a meeting to discuss how to optimise the Agency's impact in the Member States. Under contract to ENISA, GNKS Consult has conducted a "Survey to assess the practical usability of ENISA's deliverables" over the last few weeks, by questioning various stakeholders. GNKS will present the first findings of their survey in Athens, and the results will be discussed.

This meeting will be an opportunity for our stakeholders to offer ENISA immediate and direct input, suggestions and recommendations, based on the results of the survey. The Agency is looking forward to learning from the findings as well as from stakeholders' feedback, and will use the information acquired to help create a more secure e-environment for Europe.

In order to increase networking opportunities, ENISA will organise a social programme for all participants on the Tuesday, followed by a dinner at the conference venue.

Funds are available to reimburse a limited number of participants.

For more information, visit:
www.enisa.europa.eu/pages/04_01_ws_athens_20080122.htm

ENISA Short News – Fourth Quarter 2007

Panagiotis Trimintzios

Barriers and Incentives for Network and Information Security (NIS) in the Internal Market for e-Communication

ENISA held a half-day workshop on 10 December 2007 in Brussels on "Barriers and Incentives for Network and Information Security (NIS) in the Internal Market for e-Communication", launching a discussion among relevant stakeholders ensuring their input to a study which ENISA is commissioning. The event brought together representatives from all relevant stakeholder groups, i.e., EU and national decision-makers, industry and consumer and user representatives, as well as academia. The presentations are available online.

(www.enisa.europa.eu/pages/04_01_ws_brussels_20071210.htm)

3rd European Network and Information Security Conference, Vilnius, 20-22 November 2007

ENISA supported the Communications Regulatory Authority of the Republic of Lithuania in organising the third European NIS Conference, in co-operation with the Ministry of the Interior of the Republic of Lithuania.

(www.securityconference.rtt.lt)

Workshop on creating a partnership for collecting data on security incidents and consumer confidence, Berlin, 13-14 November 2007

ENISA has examined the feasibility of a data collection framework for security incidents and consumer confidence, and organised a workshop to discuss the results and to decide on the type of partnership for data collection which might be established.

(www.enisa.europa.eu/pages/04_01_dcws_wrsp.htm)

eChallenges 2007: The Hague, 24-26 October 2007

ENISA was heavily involved in the 17th eChallenges Conference and Exhibition, and organised a number of special tracks and sessions.

(www.echallenges.org/e2007/default.asp?page=home)

RSA Conference Europe 2007: Excel London, 22-24 October 2007

ENISA organised a special track, "A dialogue with ENISA", during the RSA Europe Conference at ExCel London, United Kingdom.

(www.rsaconference.com/2007/europe/about/)

High Level Dialogue, Porto, 11 October 2007

The summary of the high level dialogue that took place on 11 October in Portugal, gives a brief overview of the discussions on Synergies between ENISA and Member States, and ENISA and other institutional stakeholders, in particular the European Commission and the private sector.

(www.enisa.europa.eu/doc/pdf/news/minutes_hld_10112007.pdf)



ENISA and INTECO co-organised an event on Risk Management, 8-9 November 2007

The event took place in Barcelona and was broadcast online through video streaming on the event's web portal. This unique event shed light on how to make SMEs safer and less liable to technological incidents.

(<http://enisa.inteco.es/>)

11th ENISA Management Board Meeting – Porto, Portugal, 10 October 2007

The main focus of the agenda was ENISA's Work Programme 2008. (www.enisa.europa.eu/pages/02_03_news_2007_10_04_mgmt_board.html)

ENISA Work Programme for 2008 adopted: ENISA driving for impact

The ENISA Work Programme 2008, "Build on Synergies – Achieve Impact", focuses on increasing the Agency's impact in Network and Information Security (NIS) based on co-operation with relevant stakeholders.

(www.enisa.europa.eu/pages/02_01_press_2007_11_21_wp_2008.html)

Social Networking – How to avoid a digital hangover?

ENISA has launched its first Position Paper: 15 key threats and 19 recommendations for safer Social Networking.

(www.enisa.europa.eu/pages/02_01_press_2007_10_25_social_netw.html)

Botnets – The Silent Threat on the Internet

ENISA has published a Position Paper on Botnets, calling for stronger prosecution of cyber criminals to combat the threat caused by 6 million computers, which have been silently hijacked for online fraud.

(www.enisa.europa.eu/pages/02_01_press_2007_11_27_botnets.html)

ENISA wishes to thank all the contributors to the publication. Please remember that all contributions reflect the views of their authors only, and are not in any way endorsed by the European Network and Information Security Agency. ENISA assumes no responsibility for any damages that may result from use of the publication contents or from errors therein.

The ENISA Quarterly is published once each quarter. You can find information about ENISA Quarterly, including back issues and subscription information, on the EQ pages on the ENISA website: www.enisa.europa.eu/enisa-quarterly/

Editor-in-Chief, Panagiotis Trimintzios: eq-editor@enisa.europa.eu

More about ENISA For the latest information about ENISA, check out our website at www.enisa.europa.eu

European Communities, 2008 Reproduction is authorised provided the source is acknowledged.