

Open-Source Security Assessment

More organisations are using open source software, but what are the risks involved? **Yoav Aner** and **Carlos Cid** present a framework for security evaluation and assessment of open-source software based around threat modelling.

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE -
FRIEND OR FOE?](#)

[THREAT
MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)



1 INTRODUCTION

Nobody likes re-inventing the wheel. When you are faced with a problem, more likely than not, somebody else has stumbled upon a similar one before you, and they may have come up with a solution already.

Many problems can be solved with a simple online search: Google it! The same approach is used very frequently in an enterprise environment. Perhaps the solutions are not as simple as reading a how-to or watching a YouTube video helping you tie a Windsor knot, but there are solutions to many different business and technology requirements.

Customer Relationship Management (CRM), Helpdesk systems, virtualisation platforms, web frameworks, middleware — these are all tools that (at least claim to) help solving business needs. Streamlining customer experience, providing a better return on investment (ROI), enabling shorter time to market, reducing total cost of ownership (TCO) are just a few of the reasons to use technology.

The days of a single computing environment are long gone. In your organisation (unless you happen to be working for Dr. Evil) you will not see scientists in lab coats punching cards and crunching numbers in a big warehouse filled with refrigerator-like machines. The enterprise is filled with diverse computing platforms, different software and hardware, operating systems, complex networking infrastructure and interconnections. Companies use in-house processing, third-party partners and providers, and exchange data with customers in many different locations worldwide.

This diversity has many benefits. The enterprise can pick and choose the best tools for the job with little compromise. Very few companies are locked-in with a single vendor or a one-size-fits-all platform.

Open-source software is increasing in popularity and prevalence. Not only is it becoming mainstream for home users, many commercial and business environments are also becoming increasingly reliant on open-source software, at times, without even realising it. Vendors, third-party software providers, as well as off-the-shelf

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

software and hardware may include open-source components. Choosing open-source might make business sense too. Some open-source products are on par with competing commercial products, some may even outperform and provide better value for money. Some companies actively contribute to open-source projects in order to ensure continuous improvements and set the direction of open-source tools to fit their particular needs.

Despite all the perceived benefits and attractive features, the increasing use of open-source software may pose several security challenges to organisations.

What impact does open source have on the security posture of the organisation?

What are the risks that the use of open-source software can introduce and how can organisations address the specific issues that its use presents? How can open-source projects give better assurance to their customers?

In this article, we present a framework for security evaluation and assessment of open-source software based around threat modelling. Some of the benefits and limitations of the framework are discussed in the context of open-source software and its unique security challenges and advantages. It is hoped that such framework can be used by organisations opting

to use open-source tools, open-source development teams and security specialists wishing to analyse open-source software.

2 OPEN SOURCE — FRIEND OR FOE?

Introducing any new piece of software or a solution to an organisation involves changing the level of risk, not only to the new service itself, but also to existing systems. If, for example, a vulnerability in the new software is exploited, the server and neighbouring computing resources, internal or external, may be affected as well. Conversely, introducing a security tool may reduce the risk to the organisation. When faced with a decision between closed-source, proprietary, in-house or third-party based tools and open-source solutions, other than the functional and business requirements, what are the security considerations that should be taken? And for open-source projects, how can developers provide more confidence that their software is secure?

Whilst many open-source applications specifically deliver security functionality (e.g. OpenSSH), it is not true of them all.. With the source code readily available and with the freedom to improve and modify it, there is a great potential for security improvements and robust security, but such

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE -
FRIEND OR FOE?](#)

[THREAT
MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

potential is not guaranteed. Kerckhoff's principle ^[8] states that the security of a cryptographic system should not rely on secrecy of the algorithm, but rather on the key. Similar arguments ^[4, 10, 14] are made on the merits of open-source software and its potential for improved security. However, open-source software also provides opportunity for attackers, not only to detect and misuse software vulnerabilities, but also to directly affect the code, add insecure code, and more easily distribute tainted versions of the software.

Despite all the perceived benefits and attractive features, the increasing use of open-source software may pose several security challenges to organisations.

Most information security experts agree that hiding the source code of an application does not guarantee better security. Whitfield Diffie supported this position and stated that "A secret that cannot be readily changed should be

regarded as a vulnerability", adding further that "If you depend on a secret for your security, what do you do when the secret is discovered? If it is easy to change, like a cryptographic key, you do so. If it's hard to change, like a cryptographic system or an operating system, you're stuck. You will be vulnerable until you invest the time and money to design another system." ^[2]. However, it is also clear that the opportunity for security improvement arising from use of open-source applications does not necessarily translate or lead to better security in practice.

With that in mind, it seems clear that relying on developers, who usually focus on functionality or fixing bugs, to code securely or fix security flaws may not always be the best strategy. Open-source applications should be evaluated from a security perspective to ascertain the level of security robustness or potential exposure to threats. With a commercial product or a bespoke third-party development, some of this assessment can be "delegated", at least contractually.

As a client, you can require a certain level of certification, assurance or at least some indication as to the level of security and compliance the product can offer.

When it comes to open-source solutions, there is rarely an equivalent level of assurance (One

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

notable example to the contrary is OpenSSL, which can offer a FIPS 140-2 certified option. See <http://www.openssl.org/docs/fips/fipsnotes.html>)

The motivation for the proposal of the framework contained in this article is to give better tools for security-focused evaluation, and to allow finding (and ideally fixing) security design flaws and bugs.

One of the benefits of open-source however, is that it can give the organisation an increased level of confidence. When buying a closed-source product or service, very few companies will open their code for inspection or independent evaluation. With open-source tools however, the ability to inspect and evaluate the code is always available. Furthermore, given a wide-enough audience and user base, some of the evaluation may be performed by others and shared within the community.

Other than relying on open-source developers to spot security vulnerabilities and fix them, or worse, wait for attacks to take place and 'learn' from the experience, which options are available to open-source projects and organisations who want to use open-source tools? What is the most effective and efficient route to assess the level of security of a given product? Does it involve having teams of people trawl through the

entire code base in search for security vulnerabilities? Should the application run through automated or manual penetration testing? Should static code analysis tools be used? Section 3 investigates and attempts to answer some of these questions, and presents one potential approach based on Threat Modelling.

3 THREAT MODELLING METHODOLOGY

3.1 Why Threat Model?

Before describing the threat modelling process, it is important to consider the reasons for using threat modelling to analyse the security of open-source applications. With the aim of reducing the risks to confidentiality, integrity and availability, there might be other alternative approaches to solving the same 'problem', i.e. identifying threats, vulnerabilities and risks to an application, and finding the best ways to reduce those to an acceptable level.

When an adversary chooses to attack an application, they might opt to use one or more of these approaches, and it seems sensible to at least consider these techniques or methods when trying to identify vulnerabilities and protect against the most likely attacks. In fact, these techniques are not purely alternatives to threat

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

modelling, but rather complementary to the analysis process, and form part of the open-source security assessment framework presented in this article. It is important to remember the attacker has many advantages over those protecting the system: put simply, an attacker only needs to find one vulnerability to exploit, whilst those trying to protect the system must take care of all potential issues.

Some of the alternative / complementary security analysis and attack methods include:

- **Code auditing** - involves going through source code of the application, searching for security bugs and vulnerabilities in the code itself. With an open-source application, an attacker can easily perform this task. For example, in C code an attacker can search for known functions vulnerable to buffer overflow, then manually investigate which call can most easily be manipulated. Whilst Code Auditing can identify many bugs, it is unclear whether it is very cost effective. It requires high level of expertise and is very time consuming. Code auditing is also limited to uncovering programming flaws, and may miss out on perhaps more important design and architectural flaws.

- **Penetration testing** - utilises active attempts at 'breaking' (or hacking into) an application.

This type of testing simulates a real attack on the system, and therefore can be considered a prominent form of analysing an application's security - trying to emulate what the adversaries would attempt and learn how to protect the system better from it. Open-source applications are easier to penetration test (than e.g. a proprietary system installed in a location with limited access and strict monitoring), since a copy of the application can easily be used in a lab environment.

Despite all the perceived benefits and attractive features, the increasing use of open-source software may pose several security challenges to organisations.

Once the attack is 'perfected', it can be launched on a real system. This gives attackers another advantage of avoiding detection. However, penetration testing has some fundamental limitations on all but the most simple applications. Whilst penetration testing can prove the existence of

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

vulnerabilities, it fails to provide evidence of their absence. Even if the penetration test results in no successful attacks, the system may still be vulnerable to other attacks, which simply were not attempted.

- **Static analysis tools** - allow automatically inspecting source code for known security issues, or use heuristics to detect unsafe functions and libraries and therefore discover potential coding vulnerabilities. Static analysis tools are similar to code auditing to the extent of concentrating on the actual source code, but uses gained knowledge and known issues more effectively and focuses the attention on areas of the code more likely to be vulnerable. “Code reviews depend on the expertise of the human reviewers, whereas automated techniques can benefit from expert knowledge codified in tools” [3]. There are many open-source and commercial static analysis tools covering different programming languages and using different analysis methods. However, static analysis tools also have limitations, particularly in identifying design and architecture issues and vulnerabilities.

3.2 Threat Modelling Approach

ISO/IEC 27005:2008 states that a “threat has the potential to harm assets such as information,

processes and systems and therefore organizations” [7]. Threat modelling is defined as “a method of assessing and documenting the security risks associated with an application” [13].

Threat Modelling is primarily meant to be used in the design and development phases of applications, rather than to analyse existing applications [5]. However, the same methodology can be adapted and employed when trying to analyse an existing system. The threat modelling approach, with slight modifications, can aid in the identification of security vulnerabilities for applications or systems which are already developed. Application design and architectural threats and vulnerabilities can be identified using the threat modelling approach, as well as assisting in focusing the investigation of coding issues and implementation mistakes.

The threat modelling process used in this article is therefore likely to identify vulnerabilities rather than just threats. In addition, some threats can easily be omitted if they are already known to be mitigated in the design or implementation. For that reason, the terms threat and vulnerability may be used interchangeably in this instance.

One further addition and modification to the threat modelling process used in this article is the mixed approach used to model processes

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

and applications.

Section 3.1 mentions some techniques which may be used by both security analysts and attackers. Whilst some security assessment can be done using an implementation of a system (as may be provided by a test implementation of a closed-source application), there are further benefits that may be obtained from the use of open-source applications. Part of the investigation process of threats to an open-source system can therefore include some level of practical experimentation with the application, reviewing code segments, searching for keywords in code as well as focused and specific elements of penetration testing. Using threat modelling to aid code reviews and penetration testing is already documented in [6, 13]. However, instead of using a linear process, where the threat model feeds into code review and penetration testing, the approach presented here advocates the use of a more symbiotic (or hybrid) attitude. For example, if the analysis is trying to establish how authentication is performed by the application, in some instances it may be easier and more effective to look at the code or libraries used by the application. A balance should be made between higher-level, design and architectural (or even conceptual) threat modelling and the

more hands-on tasks of looking through code or running automated tools.

3.3 Threat Modelling Process

The threat modelling process used in this article is based predominately on the one pioneered by Microsoft since 1999 [11]. The process itself evolved over the years and is now incorporated into the Microsoft Secure Development Lifecycle at the core of the Risk Analysis process [6].

Threat modelling incorporates 4 key stages:

1. Application Analysis / Diagramming.
2. Threat Enumeration.
3. Threat Rating.
4. Mitigation Options.

3.4 Application Analysis / Diagramming

This stage of the threat modelling process consists of analysing the application from a flow of data perspective. All assets that make up the application are catalogued, and then the relationships between them are identified in terms of data exchange. Data Flow Diagrams (DFD) [9] help both visualise elements of the applications and the flow of data between them. The Microsoft threat modelling further enhanced the 'classic' DFD by adding a 'trust boundary' element [5]. Trust boundaries aid in analysing differ-

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE -
FRIEND OR FOE?](#)

[THREAT
MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

ent privilege and trust levels and better assess threats. Figure 1 shows a sample DFD diagram.

Other background information which is used in the analysis process includes:

- **Use Scenarios** - descriptions of typical as well as atypical use (including unauthorised use scenarios). This allows better understanding of the application and its components, and makes sure no data flow diagram elements are missing. Considering unauthorised use scenarios can help direct security threat assessment and security testing as well as ensure practical scenarios are considered.

- **External Dependencies** - any third-party components, libraries, tools and services which may affect the security of the application.

- **Security Assumptions** - assumptions relating to the security provided by any third-party or relied-upon components.

- **External Security Notes** - notes which are made available to users of the application about the security provided by default or which can or cannot be configured. Notes can also include limitations, recommendations and warnings.

3.5 Threat Enumeration

Each element on the Data Flow Diagram is analysed against a list of potential threats depending on the element type. Threats are categorised based on the STRIDE [13] taxonomy:

- **Spoofing** - Masquerading, stealing or disguising one identity with another. Spoofing does not solely apply to user identities and individuals. Server spoofing can also take place, e.g. in phishing attacks.

- **Tampering** - Altering, modifying, adding or retracting data. Tampering includes any unauthorised or unintended modification and compromise of data integrity. Tampering threats not only apply to data, but also to communication links and processes.

[HOME](#)

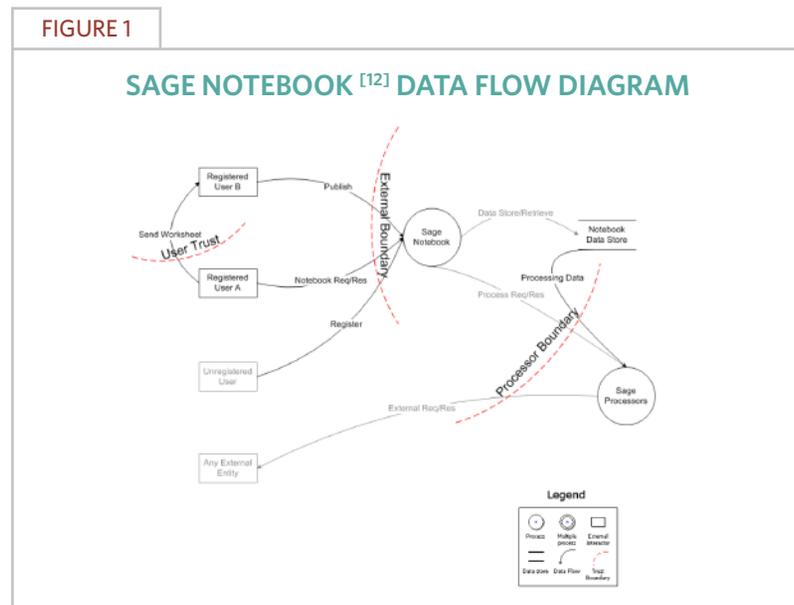
[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)



- **Repudiation** - Denying having performed an action, or covering tracks after a malicious act or misuse.

- **Information Disclosure** - An attack on information confidentiality. Obtaining unauthorised access to information.

- **Denial of Service** - Compromising system or application availability. Reducing or denying access to resources.

- **Elevation of Privilege** - Obtaining unauthorised elevated access to services or resources. Elevation of Privilege attacks primarily aim at obtaining the highest level of access (Administrator, or root), but it is not limited to it. Elevation from anonymous user to a registered one, e.g. from a registered user to 'power' user, are also of concern.

When threats are analysed, a consideration is given whether or not a control is in place to partially or fully mitigate the threat. When analysing an existing open-source system this process is slightly different from traditional threat modelling. Controls are either already implemented or they are missing.

It may be useful to enumerate all threats, but in order to keep the analysis brief and focused, enumeration usually includes only realistic threats

and actual vulnerabilities (i.e. lack of control) that need to be considered and potentially addressed.

3.6 Threat Rating

Rating threats is essential to determining the most cost effective approach for remediation and mitigation. It helps to ensure the necessary resources, time and attention are given to the more critical threats. The most effective threat rating is therefore a scaling based on risk. The higher the risk to the application caused by the threat, the higher the priority or rating of the threat. There are different risk rating or estimation methods, including various qualitative and quantitative techniques. Even though the threat modelling process is, as its name suggests, looking at threats, the process is in fact equivalent to risk rating

as described in ^[7]. When rating a threat, the following aspects have to be considered:

- **Application assets (e.g. components, data flows)** - described on the Data Flow Diagram, as covered in section 3.4.

- **Threats to assets (e.g. spoofing, tampering)** - threats are identified and enumerated, as described on section 3.5.

- **Identification of existing controls** - when analysing an existing application, as is the case

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

in this paper, some controls would already exist.

However, even with an application not yet implemented, some controls may have already been designed. Also covered in section 3.5

- **Identification of vulnerabilities** - Vulnerabilities may exist at the architecture level and are also considered as part of section 3.5.
- **Identification of consequences** - The impact of a threat 'materialising' by exploiting a vulnerability to an asset.

Therefore, when determining the risk levels for each of the enumerated threats and establishing a rating of threats, all those considerations are analysed together with the estimated likelihood/probability of the consequences taking place.

3.7 Mitigation Options

Threat mitigation is provided to reduce the risk associated with threats to an acceptable level. Mitigation is subject to cost/benefit analysis and consideration into the best ways to address particular threats or vulnerabilities. Threat mitigation of already developed applications is more constrained than with applications still in the design process, primarily because mitigation tends to be more complex and costly, and the

pressure to fix vulnerabilities might be higher.

There are several types of mitigation options available, which range between removing functionality, patching, adding other security controls to re-design. Some of the risks may be acceptable to the organisation or specific deployment and left unmitigated.

3.8 Caveats and Limitations

It is important to understand the limitations and caveats of the threat modelling process, and to avoid a false sense of security. In the context of threat modelling, a failure to identify a threat, vulnerability or a problem might mean the problem does not get addressed, whereas the threat or issue does in fact exist.

A typical threat model is also likely to be more focused, and therefore miss wider-scope issues, such as:

- **Future threats and patterns** - exploits and attacks evolve and change constantly. Attackers change their patterns or interest and techniques gain focus or 'go out of fashion'. For example, integer overflow vulnerabilities climbed to number 2 for operating system advisories in 2007, after barely being included in the top 10 list in the previous few years ^[1].
- **Human elements** - Some human elements are

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE -
FRIEND OR FOE?](#)

[THREAT
MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

covered by a typical threat model, but others are not. Social engineering attacks are not impossible to identify, but more difficult to capture using the threat model than more direct threats.

4 CONCLUSION

The framework presented in this article was applied to the Sage Notebook ^[12] open-source software as part of the first author's MSc thesis project. Using the framework, the analysis highlighted numerous threats and vulnerabilities both to the development process of the software and the software itself. Further to the paper, a number of changes to the software were planned and executed to reduce the level of risk in accordance with the project's priorities and goals.

Open-source software may not be more secure than closed-source alternatives. It does however bear the potential of becoming increasingly more secure, given the right attention and the necessary approach to security. The methodology and processes covered in the article hopes to give some direction to those who wish to assess the security of open-source software, either in an attempt to determine if this software can be usefully implemented in their organisation or as a prelude to improving that software. By utilising such a framework, open-source software can be

assessed in a more comprehensive and focused manner to gain confidence and assurance.

Whilst not revolutionary nor new, it is believed that the framework and ideas presented in this article can be applied to many different open-source projects and in many environments. Building upon well-established threat modelling methodologies as well as the inherent benefits of open source, this framework can be used by both open-source project teams wishing to give assurance to prospective clients, and organisations and companies wishing to use open-source. ■

ABOUT THE AUTHORS

Yoav Aner is an information security specialist with 16 years experience. His areas of expertise include security architecture, design and evaluation, with particular focus on application security and practical usage of cryptography. He has worked both domestically and internationally in the telecom, finance and IT industries.

Carlos Cid joined the Information Security Group at Royal Holloway in October 2003 as a postdoctoral research assistant to work on the EPSRC-funded project "Security Analysis of the Advanced Encryption Standard (AES)". He is currently a RCUK Academic Fellow. Carlos has a broad interest in the area of information security, in particular cryptography.

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE - FRIEND OR FOE?](#)

[THREAT MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)

REFERENCES

- [1] S. Christey and R.A. Martin. Vulnerability type distributions in CVE. Common Weakness Enumeration, version, 1.1, 2007. Available at <http://cwe.mitre.org/documents/vuln-trends.html>. Last accessed, 30 August 2009.
- [2] W. Diffie. Risky business: Keeping security a secret. Available at http://news.zdnet.com/2100-9595_22-127072.html, 2003. Last accessed, 14 July 2009.
- [3] D. Evans and D. Larochelle. Improving security using extensible lightweight static analysis. IEEE software, pages 42-51, 2002.
- [4] J.H. Hoepman and B. Jacobs. Increased security through open source. COMMUNICATIONS-ACM, 50:79-84, 2007.
- [5] M. Howard and D.E. Leblanc. Writing secure code. Microsoft Press, second edition, 2002.
- [6] M. Howard and S. Lipner. The Security Development Lifecycle. Microsoft Press, 2006.
- [7] ISO/IEC. ISO/IEC 27005:2008 Information technology - Security techniques - Information security risk management. First edition, International Organization for Standardization, Geneva, Switzerland., 2008.
- [8] Auguste Kerckhoffs. "la cryptographie militaire". Journal des sciences militaires, vol. IX, 1883. available at <http://www.petitcolas.net/fabien/kerckhoffs/>, Last accessed, 14 July 2009.
- [9] P.G. Larsen, N. Plat, and H. Toetenel. A formal semantics of data flow diagrams. Formal aspects of Computing, 6(6):586-606, 1994.
- [10] Bruce Schneier. Open source and security. Crypto-Gram. Counterpane Internet Security, Inc., September 15, 1999. Available at <http://www.counterpane.com/cryptogram-9909.html>. Last accessed, 25 July 2009.
- [11] A. Shostack. Experiences Threat Modeling at Microsoft. In Modeling Security Workshop. Dept. of Computing, Lancaster University, UK, 2008. Available at: <http://blogs.msdn.com/sdl/attachment/8991806.ashx>. Last accessed, 27 August 2009.
- [12] W. A. Stein et al. Sage Notebook Public Server. The Sage Development Team. Available at <http://www.sagenb.org/>. Last accessed, 28 July 2009.
- [13] F. Swiderski and W. Snyder. Threat Modeling. Microsoft Press, 2004.
- [14] B. Witten, C. Landwehr, and M. Caloyannides. Does open source improve system security? IEEE SOFTWARE, pages 57-61, 2001.

[HOME](#)

[INTRODUCTION](#)

[OPEN SOURCE -
FRIEND OR FOE?](#)

[THREAT
MODELLING](#)

[CONCLUSION](#)

[REFERENCES](#)