Securing the Sage Notebook

Yoav Aner

Technical Report
RHUL–MA–2010–04
31st March 2010

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England

http://www.rhul.ac.uk/mathematics/techreports

Royal Holloway, University of London
MSc Information Security

Securing the Sage Notebook

Yoav Aner

Student Number: 100628630
Supervisor: Dr Carlos Cid
Co-Supervisor: Martin Albrecht

Submitted as part of the requirements for the award of the MSc in Information Security at Royal Holloway, University of London.

I declare that this assignment is all my own work and that I have acknowledged all quotations from the published or unpublished works of other people. I declare that I have also read the statements on plagiarism in Section 1 of the Regulations Governing Examination and Assessment Offences and in accordance with it I submit this project report as my own work.

Signature:

Date:

September 1, 2009

Acknowledgements

Thanks firstly to my parents, who keep encouraging me to further my education and convinced me to sign up to this MSc.

Claire, who had to sacrifice most of her social life in the last few months. Your patience means much to me.

Dr Carlos Cid for helping structuring the analysis methodology and support throughout the project. Your advice, knowledge and guidance made the project work so much easier and more enjoyable.

Martin Albrecht for proposing the project and supporting it every step of the way, explaining the software architecture, helping with the threat analysis and introducing me to key people on the Sage project.

William Stein and so many others on the Sage community, who have been friendly, helpful and inspiring in their dedication and knowledge.

Raquel Caparrós and the organisers of Sage Days 16 in Barcelona for their hospitality.

Contents

Executive Summary **1**

1 Introduction **2**
1.1 Sage Notebook 4

2 Threat Modelling Methodology **6**
2.1 Why Threat Model? 6
2.2 Threat Modelling Approach 8
2.3 Threat Modelling Process 9
2.3.1 Application Analysis / Diagraming 10
2.3.2 Threat enumeration 10
2.3.3 Threat rating 13
2.3.4 Mitigation options 14
2.4 Threat Model Analysis 16
2.5 Caveats and Limitations 16

3 Development Process Threat Model **19**
3.1 Threat Model Information 19
3.2 Data Flow Diagrams 20
3.2.1 Sage Open Source Development Process 20
3.3 Threats and Mitigations 21
3.3.1 Elements 21
3.3.2 Add/Change Code 21
3.3.3 Load 22
3.3.4 Retrieve Code for Installation 23
3.3.5 Update 23
3.3.6 Use 24
3.3.7 Code Base 24
3.3.8 Contributors 25

3.3.9 System Owners . . . 25
3.3.10 Users . . . 25
3.3.11 Sage . . . 25
3.3.12 Development Process . . . 27
3.4 External Dependencies . . . 28
3.5 Implementation Assumptions . . . 28
3.6 External Security Notes . . . 29

4 Sage Notebook Threat Model 30
4.1 Threat Model Information . . . 30
4.2 Data Flow Diagrams . . . 32
4.2.1 Sage Notebook . . . 32
4.3 Threats and Mitigations . . . 33
4.3.1 Elements . . . 33
4.3.2 Data Store/Retrieve . . . 33
4.3.3 External Req/Res . . . 34
4.3.4 Notebook Req/Res . . . 34
4.3.5 Process Req/Res . . . 35
4.3.6 Processing Data . . . 35
4.3.7 Publish . . . 37
4.3.8 Register . . . 37
4.3.9 Send Worksheet . . . 39
4.3.10 Notebook Data Store . . . 39
4.3.11 Any External Entity . . . 41
4.3.12 Registered User A . . . 42
4.3.13 Registered User B . . . 43
4.3.14 Unregistered User . . . 44
4.3.15 Sage Notebook . . . 44
4.3.16 Sage Processors . . . 49
4.4 External Dependencies . . . 53
4.5 Implementation Assumptions . . . 53
4.6 External Security Notes . . . 54

5 Sage Threat Model Analyses 55
5.1 Overview . . . 55
5.2 Development Process Threat Model Analysis . . . 55
5.3 Sage Notebook Threat Model Analysis . . . 56
5.4 Untrusted Code . . . 57

5.4.1 Operating System Protection 58
5.4.2 Security Add-ons 59
5.4.3 Network Isolation 60
5.4.4 Virtualisation 61

6 Conclusion **63**

Executive Summary

This paper looks at some of the information security challenges of Web based Open Source applications through a case study of the Sage Notebook application.

Considering the core underlying issues of open source and web based applications, predominately the fact that the source code of the application is exposed to any potential attacker, the paper investigates methodologies to examine and improve upon the security of such applications.

The Sage Notebook application provides some unique information security challenges, both in terms of analysis and mitigation. The paper uses a structured threat modelling process based on industry methodologies to identify threats and vulnerabilities to both the Sage open source development process and the application itself. It rates the discovered threats and suggests several mitigation options to consider.

The paper analyses the findings, focusing on several architectural and design mitigation options, and investigates some of the technologies and tools to address the discovered threats and vulnerabilities most effectively. It covers generic open source and web based security challenges as well as issues affecting cloud computing, software as a service, virtualisation, process isolation and containments and others.

Chapter 1

Introduction

"Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in" [43]
The Free Software Foundation (FSF) further defines *Free* Software as: "... a matter of the users' freedom to run, copy, distribute, study, change and improve the software.", and specifically in relation to Open Source Software "The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this." [25]

Whilst there are a number of open source applications specifically delivering security functionality (e.g. GPG [27], Truecrypt [102], IPTables [67], OpenSSH [69]; The author of [40] dedicated an entire book covering over 40 of the leading open source security tools in detail), many open source applications are not necessarily focused on security. 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 potential is not guaranteed. Kerckoff's principle [50] states that the security of a cryptographic system should not rely on secrecy of the algorithm, but rather on the key. Similar arguments [83] [110] [37] 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.

The opportunity for security improvement does not necessarily translate or lead to better security in practice. What seems to be agreed by most Information Security leading experts however, is that hiding the source code of an application does not guarantee better security. Whitfield Diffie summarised it as "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." [17].

Web based applications appear to increase in popularity and gain more acceptance and support every day. The relative simplicity and uniformity of web browsers and the ability to reach users from across the globe using the Internet, a single, standardised and robust infrastructure, means that web applications provide a cost effective and efficient way of providing service. Users can nowadays access a wide range of applications and services online. From email and social networking applications like Hotmail, Facebook and Twitter, through web based banking, to purchasing groceries, travel tickets, sharing photos online and much more. Many software services are increasingly provided over the Internet, and are not simply browser based applications, but available as an online service. Further enhancements such as AJAX (Asynchronous Javascript + XML) technologies provide much improved user experience and usability approaching those of more traditional desktop applications [31] [74]. Such a delivery model provides many benefits, but also opens up opportunities for misuse. Bounderies and physical locations no longer play a part, notions of trust change, definitions of inside or outside lose their meanings. Web applications and browsers are an increased target for attacks and there is a continuous growth in Web application vulnerabilities [42].

Open Source and Web based applications, when combined, present some interesting yet dawnting Information Security challenges. Opening up the source code means risking exposing potentially harmful information about the inner working of the application. Such information, in the wrong hands, can allow attackers to abuse the system. Simultaneously, the aim of such applications is to provide an online, always available service to (usually) as large a community as possible. Balancing these two seemingly conflicting aims is difficult, even if the underlying assumption that the application is developed with good security practices and that no major vulnerabilities exist. Such conflicting aims can collide even further if, for example, vulnerabilities are discovered, or when the confidence in the security of the application is undermined. Particularly with smaller open source projects, and those who lack dedicated or qualified resources focusing on improving Information Security. John Viega, the original author of Mailman [58] and an author of several Information Security books, states that "...the benefits open source provides in terms of security are vastly overrated, because there isn't as much high-quality auditing as people believe, and because many security problems are much more difficult to find than people realize. Open source programs which appeal to a limited audience are particularly at risk, because of the smaller number of eyeballs looking at the code.". [105]

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? If an open source project does decide to improve security, what is the most effective and efficient route? Does it involve having teams of people trawl through the entire code base in search for security vulnerabilities? Should the web application run through automated or manual penetration testing? Should static code analysis tools be used? Chapter 2 investigates and attempts to answer some of these questions and presents one potential approach adopted and used on this paper.

1.1 Sage Notebook

Sage is an open source tool for studying and experimenting with mathematics, including algebra, calculus, number theory, cryptography, numerical computation, commutative algebra, group theory, combinatorics, graph theory and more [94]. Sage Notebook is a web based front-end to Sage, allowing users to use Sage online, over the network and on-demand without necessarily installing and running Sage locally. Users or groups can set up their own Sage installations and share work, collaborate or provide a service to others. Services range from providing local installation for a small group of users, through campus networks, and up to providing global free and anonymous service [93].

The free open source nature of Sage means anybody can use it, modify it and provide services based on Sage to others. Sage is licensed under the GNU General Public License (GNU GPL) [26], which does not prevent companies or individuals selling Sage, nor does it stop companies or individuals from providing a paid-for service based wholly or partly on it [28].

Sage uses and relies upon numerous other open source projects and tools as much as possible [92]. This approach allows great flexibility in terms of modularity, reusability, extensibility and keeps the Sage code base smaller and more efficient. However, it comes at a cost of reliance on third parties, a dependence that can become risky from security perspective some times. If a third party component contains a vulnerability or a flaw, Sage will also be vulnerable or suffer from the same flaw. Even if a vulnerability is discovered by the Sage open source community, it may be hard to fix. The Sage developers may be able to apply a patch for a specific easy-to-fix issue. However, if the problem is harder and require more detailed knowledge of the component, they have to rely on the third party to provide a fix. Third parties may have different priorities when it comes to applying fixes.

On first impression, the Sage Notebook can be viewed as not much different from many other web applications or from a simple web front-end to a legacy application. In essence, it operates using a very basic request-response model. It receives a request for calculation (e.g. formula), processes it by passing it to a backend 'engine', and produces a response (e.g. a result, graph, equations etc). However, the unique nature of the data being processed and the flexible architecture creates security challenges. Sage is a mathematics application, and as such, processes mathematical formulas or in some cases, application source code or embedded system calls. The execution can be carried out on different back-end systems, such as Singular [35] and GAP [29], as well as using direct access to system calls via the Unix shell [7]. Chapter 4 and Chapter 5 explore the security challenges created by using the variety of back-end systems in such a way. Such a processing model is considered quite unique and dissimilar to most other web based applications. Furthermore, unlike other web applications, where the user-supplied data is relatively predictable and constrained, Sage mathematic formulas and user code cannot easily conform to certain rules or formats and it is therefore extremely difficult to distinguish between valid and malicious user-supplied data. Input validation and sanitization is one of the main security issues faced by web applications [82] [95] [103], and one of the methods of dealing with untrusted input is using validation constraints and

transformation rules [84]. Due to the complex nature of user input in Sage, such approach may not be viable, or at least not trivial to implement.

Another element of risk, much less unique to Sage and which applies to many other open source applications, is the ability of adversaries to affect the product itself. By actively contributing tainted or deliberately insecure code to the code base, they not only rely on existing vulnerabilities in the applications, but can also *introduce* new vulnerabilities into particular areas. If such contributions are accepted into the code base unnoticed, they are then included in all subsequent versions (binary or code) versions of the application (at least until they are detected, by which time the attackers may have already carried out their plan). Chapter 3 uses the same methodology to explore such threats and suggest mitigation options.

Chapter 2

Threat Modelling Methodology

2.1 Why Threat Model?

Before describing the threat modelling process, it's important to consider the reason(s) for using threat modelling as a methodology of analysing the security of an open source web application. Assuming the aim of the process is to reduce the risks to an application's 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 way(s) to reducing those to an acceptable level. Some possible directions were mentioned briefly on Chapter 1 and are to be considered.

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. It's important to remember, the attacker has many advantages over those protecting the system. The authors of [38] describe 4 principles showing the difficulties faced by attempting to protect an application against attacks and the advantages adversaries are assumed to have. Similar attitude is taken when considering the ability of an adversary against a cryptographic system [66].

Some of the alternative 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 the very same task. The auditing process can be manual, automated or using a mixed approach. For example, in C code an attacker can grep (search) for known functions vulnerable to buffer overflow (e.g. strcpy) [38], then manually investigate which call can most easily be manipulated.
  Whilst Code Auditing can identify many bugs, it's unclear whether it is very cost effective. Code auditing is also limited to uncovering pro-

gramming flaws, and miss out on perhaps more important design and architectural flaws. As described by John Viega, Code Auditing "is not a very cost-effective way to find bugs, even when you're paying for tools. Not only is it expensive, but also the bugs you do find with code auditing often won't be the same ones the bad guys will find. As a result, even if you're finding lots of bugs, it's tough to show the value of an audit because you might not have gotten the stuff that the bad guys will find most easily" [106]

- **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. 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 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 weren't attempted. In addition, penetration testing may achieve most if it is driven from an architecture review or a threat modelling [38], but it may not be sufficient on its own or as an *alternative* to threat modelling. "Penetration testing is also useful, especially if an architectural risk analysis is specifically driving the tests. The advantage of penetration testing is that it gives a good understanding of fielded software architecture in its real environment. However, any black-box penetration testing that doesn't take the software architecture into account probably won't uncover anything deeply interesting about software risk. Software that falls prey to canned black-box testing which simplistic application security testing tools on the market today practice is truly bad. This means that passing a cursory penetration test reveals very little about your real security posture, but failing an easy canned penetration test tells you that you're in very deep trouble indeed." [62]

- **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. "Techniques to detect and correct software flaws include human code reviews, testing, and static analysis. Human code reviews are time-consuming and expensive but can find conceptual problems that are impossible to find automatically. However, even extraordinarily thorough people are likely to overlook more mundane problems. Code reviews depend on the expertise of the human reviewers, whereas automated techniques can benefit from expert knowledge codified in tools" [18].

There are many open source and commercial Static Analysis Tools, e.g. RATS [86], Coverity [16], Yasca [85], Klocwork [52] covering different programming languages and using different analysis methods. The authors of [41] demonstrate an approach for static code analysis specifically for web applications. However, Static Analysis Tools also have limitations, particularly in identifying design and architecture issues and vulnerabilities.

"Static analysis can't solve all your security problems. For starters, static analysis tools look for a fixed set of patterns, or rules, in the code. Although more advanced tools allow new rules to be added over time, if a rule hasn't been written yet to find a particular problem, the tool will never find that problem. When it comes to security, what you don't know is likely to hurt you, so beware of any tool that says something like, "zero defects found, your program is now secure." The appropriate output is, "sorry, couldn't find any more bugs." ... Knowledgeable people still need to get a program's design right to avoid any flaws although static analysis tools can find bugs in the nitty-gritty details, they can't critique design. Don't expect any tool to tell you, "I see you're implementing a funds transfer application. You should tighten up the user password requirements."" [13]

2.2 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." [44].

Threat Modelling is defined as "a method of assessing and documenting the security risks associated with an application. This methodology can help development teams identify both the security strengths and weaknesses of the system and can serve as a basis for investigating potential threats and testing and investigating vulnerabilities." [96].

Threat Modelling is primarily meant to be used in the design and development phases of applications, rather than to analyse existing applications [38]. However, as this paper attempts to demonstrate, the same methodology can be 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 differences between threat modelling new or existing applications as well as the modifications to the 'traditional' threat modelling methodology are relatively minor. However, they are important to highlight clearly. When designing a new application, it's important to consider as many threats as possible, and try to counter those threats using various means, based on the perceived risk and the cost/benefit to mitigate. During the design and development phase, it may be sensible to drop some functionality to mitigate a threat. For example, if the functionality is not a 'must-have' yet it exposes the application

to serious threats, it might be easier to drop it from the design. Cost will be substantially lower. However, when the functionality is already developed and used, it's much harder to take it away, even if psychologically. Time and effort were already put into implementing it, and it's difficult to 'throw' all this effort away. On the other hand, even if the application was developed with no formal threat modelling process, some threats may have already been mitigated. For example, numerous applications implement authentication and authorisation (even if trivial), or using cryptographic tools to protect communication links (e.g. using SSL). One of the bigger advantages of identifying threats early on in the development process is the ability to prevent threats from becoming vulnerabilities [39] [96]. Threats identified early on can be fixed before they are implemented. Threats that are identified later on, once the application is already implemented, are harder and more costly to mitigate (The authors of [6] claim that the cost of fixing a software issue after implementation is often 100 times more expensive than during the design phase; Similar stance is taken by [98], albeit with an estimated cost factor of 30). In addition, threats which were not considered and mitigated (either formally or informally) as part of the design and development process, are not only threats, they are most likely vulnerabilities in the application.

The threat modelling process used on this paper is therefore likely to also identify vulnerabilities rather than just threats. In addition, some threats may have been omitted if they were already known to be mitigated in the design or implementation. For that reason, the terms threat and vulnerability may be used interchangeably on some instances.

One further addition and modification to the threat modelling process used on this paper is the mixed approach used to model the process and application. Section 2.1 mentions some techniques which may be used by attackers. From security analysis perspective (both an attacker's and on this paper), having an existing system provides some benefits to a completely 'hands-off' threat modelling approach. Some of these benefits are directly linked to the open source nature of the application, allowing easy access to both the application and its code. Part of the investigation process of threats therefore included some level of practical experimentation with the application, reviewing code segments, searching for keywords in code and to a limited extent, focused and specific elements of penetration testing. Leveraging threat modelling for code reviews and penetration testing is already documented in [39], [96]. However, instead of using a linear process, where the threat model feeds into code review and penetration testing, this paper used a more 'symbiotic'/'hybrid' approach. It is important to note however that due to limited timescales, and in order to keep the attention focused, the process largely involved 'traditional' threat modelling and only a small contribution was made using those additional techniques. Nevertheless, such contribution was noted in a number of threats and vulnerabilities which might have not been easily identified otherwise.

2.3 Threat Modelling Process

The threat modelling process used on this paper is based predominately on the one pioneered by Microsoft since 1999 [88]. 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 [39].

Threat modelling incorporates 4 key stages:

1. Application Analysis / Diagraming
2. Threat enumeration
3. Threat rating
4. Mitigation options

The following sections describe each stage in more detail.

2.3.1 Application Analysis / Diagraming

Analysing the application from a flow of data perspective. All assets which make up the application are catalogued, and then the relationships between them are identified in terms of data exchange. Data Flow Diagrams (DFD) [56] [55] 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 [38]. Trust boundaries aid in analysing different privilege and trust levels and better assess threats. Table 2.1 lists the various types of elements on the diagram, their description/use and gives some examples of typical elements of each type.

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

- **Use Scenarios** - identify 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 give a direction and flag potential threats even before going through more elaborate analysis.
- **External dependencies** - any 3rd party components, libraries, tools and services which may affect the security of the application.
- **Security Assumptions** - assumptions relating to the security provided by any 3rd 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.

2.3.2 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 STRIDE taxonomy:

DFD Element Type	**Description**	**Example**	**Shape**
Complex Process (also called multiprocess)	An entity which performs many complex operations, which can be broken down further to other processes	An application component, a web server/service	Double circle
Process	An entity which performs a distinct task or a function. The distinction between complex and simple process is arbitrary, and depends on the level of detail which needs to go into the threat model and analysis	A web server/service, an executable, library object	Circle
External Entity (also called Interactor)	An entity which uses or activates elements within the application, outside the control of the application or process	Typically users, but also asynchronous events, external processes or third parties	Rectangle
Data Store	Persistent data storage element	Typically files or databases	Parallel Lines
Data Flow	A flow of data between diagram elements	Networking links, function calls, remote procedure calls	Arrowed line
Trust Boundary	Mark where data flow traverses between different trust levels	Typically crosses between external entities and the application, but can also be used within the application, e.g. functionality that resides in system or user space	Red dotted line

Table 2.1: DFD Element Types (based on [39])

- **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 [46].
- **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 to data, but also to communication links and processes.
- **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, from a registered user to 'power' user etc are also of concern.

STRIDE can be considered the flip side of the core information security goals: Confidentiality, Integrity and Availability (CIA), Authentication, Authorisation and Non-Repudiation. STRIDE enables focusing on the attackers perspective and investigate which techniques can be used to breach the security of an application or system. Some threat types are only applicable to certain objects. For example, an external entity may be subject to spoofing threats, whereas the notion of a data flow (communication link) being 'spoofed' doesn't make logical sense. Table 2.2 shows each DFD element and its corresponding potential threats. Each element is listed against STRIDE threats, and a checkmark is used to indicate whether or not the threat applies to the data element. On the intersection of Data Store and Repudiation there's a dagger symbol (†). It is there to illustrate that repudiation threats apply to a data store if it is used to store audit records / log trail. Repudiation can take place if the data is linked to actions by individuals, and (unauthorised) modification of the data can cover the tracks for misuse or allow the individual to deny having performed certain actions.

When threats are analysed, a consideration is given whether or not a control is in place to partially or fully mitigate the threat. When the analysis is carried out on a new application, controls may or may not be part of the design. At this stage, the threat modelling process should avoid trying to 'fix' the design when weaknesses are identified or controls are highlighted as missing [101]. Any such fixes may apply later on, once the process is complete. When analysing an existing system this process is slightly different. Controls are either already implemented or they are missing. It may be useful to enumerate all threats, including those which are addressed/mitigated by existing controls. However, in some cases, fully mitigated threats are not listed in order to keep the analysis brief and flag only realistic threats and actual vulnerabilities (i.e. lack of control) that need to be considered and potentially addressed.

DFD Element Type	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privileges
External Entity	✓		✓			
Data Flow		✓		✓	✓	
Data Store		✓	†	✓	✓	
Process	✓	✓	✓	✓	✓	✓

Table 2.2: Mapping of STRIDE to DFD Elements (Source: [39])

2.3.3 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 [44]. 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 2.3.1.
- **Threats to assets** (e.g. spoofing, tampering) - threats are identified and enumerated, as described on section 2.3.2.
- **Identification of existing controls** - when analysing an existing application, as is the case on 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 2.3.2
- **Identification of vulnerabilities** - Vulnerabilities may exist at the architecture level even before the application is implemented. Existing implementation vulnerabilities are also considered as part of section 2.3.2. See section 2.2 for more information about identification of vulnerabilities.
- **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.

One of the risk rating approaches recommended and used historically in the Threat modelling process is DREAD [96] [88]:

- **Damage Potential** - The maximum potential impact to the asset.
- **Reproducibility** - How often an attack attempt is likely to succeed.
- **Exploitability** - How easy or hard it is to exploit the vulnerability, including necessary pre-conditions.

- **Affected Users** - The percentage of application users affected by an exploit.
- **Discoverability** - How easy or difficult it is to discover the vulnerability.

However, the authors of [39] maintain that this process is highly subjective and explain the difficulty in keeping consistent rating.

In the context of this paper, the DREAD approach was chosen to only be used implicitly. When considering the DREAD elements in the context of threat modelling an open source project, the following observations can be made:

1. **Discoverability** is high for all threats. Sage is an open source system and as such it is safer to assume threats can easily be discovered. In addition, the results of this paper are published and shared with the Sage development community, and such sharing is in fact encouraged.
2. **Reproducibility** and **Exploitability** are closely linked and difficult to assess independently. They represent the likelihood/probability of the threat materialising.
3. **Damage Potential** and **Affected Users** are also linked. They both represent the impact to assets.

Therefore, whilst the Threat rating analysis would consider DREAD elements, it does not attempt to map or rate each element separately, but rather amalgamate the elements into an assessment of risk rating as a product of impact and probability.

The rating chosen for this paper is a simple categorisation into High, Medium and Low. Rating is not arbitrary and is subject to discussion. The threats as well as their ratings are put forward to the Sage open source development community to comment on. All feedback is considered as part of the process, and as a result, the 'many eyeballs' benefit of Open Source projects can come into play when performing threat modelling. It's also important to highlight that the rating is not absolute, but is in many cases environment and deployment dependent. For example, in one deployment, Information Disclosure threats may be rated much higher than in another, where more concern may be placed on Denial of Service. The rating on this paper attempts to look at the sagenb.org deployment as a reference, but it is by no means limited to this particular instance.

2.3.4 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 (as already discussed in section 2.2), primarily because mitigation tends to be more complex and costly and the pressure to fix vulnerabilities might be higher. In such circumstances, a quick-fix or a patch may be preferable to resolve the problem in the most economical way.

Such solution may be appropriate or acceptable, but it may not be the most robust and long term option (which may have been implemented if the threat was identified earlier).

There are several types of mitigation options available:

- **Removing functionality** - Such approach may be relatively simple to implement, but it means delivering less functionality to users. This option may only be available if the functionality is seldom used and there is no dependence on it (either by software components, third parties or users).
- **Patching** - patching involves adding or modifying existing code to close a gap or fix a problem. Patches are used by software vendors to resolve security issues identified in their products after release. Vendors attempt to release patches as soon as possible after a vulnerability is discovered and try to reduce the window of vulnerability [48]. Patches are a fairly effective fix for problems, bugs and security vulnerabilities that can be pinpointed to a limited area of code or functionality. However, design flaws and vulnerabilities are much harder to fix using a patch. For example, if no authorisation system was designed, it often can't simply be patched. A whole subsystem may be required.
- **Adding controls** - additional or compensating controls can be added to the application in some cases. For example, placing an external Firewall or an Intrusion Prevention System in front of a server may provide sufficient mitigation for some network related threats. Such approach may have the advantage of little or limited modification to the application code itself, and can therefore be relatively easier to implement. However, adding controls might not be sufficient to address certain threats. Threats which are inherent to the application design and architecture may be impossible to fix with such an approach, as those threats are at the core of the application. For example, authentication issues may not be fixed by a completely independent additional control.
- **Re-Designing** - Re-Designing an application should provide the most comprehensive mitigation alternative. Re-design can range from being limited to a subset of functionality, to a complete overhaul and re-design of the entire application. Re-design enjoys the benefits of applying fixes early and having the benefits of early threat modelling. Obviously, this is the most 'extreme' option, and as such is usually cost/time prohibitive. Re-design is usually considered 'the last resort'. However, in some cases, re-designing a subset of the application can in the long run be more beneficial than solving it using patches and external controls. Re-factoring (and to a more limited extent throwing away and starting from scratch) is necessary and considered 'healthy' within the software community [22]. Similar arguments can be made for the benefits of re-factoring and re-designing for threat mitigation.
- **Accepting** - Some threats may be acceptable to the application, or in certain deployments. If the cost/benefit ratio of applying a fix is above a certain threshold, a decision can be made to leave the threat unmitigated or accepted. In such cases, the external security notes (see section 2.3.1)

should be updated to make sure users are aware of such threats when using the application.

On this paper, an attempt was made to suggest at least one, but preferably several mitigation options for the threats and vulnerabilities identified as part of the modelling process. The alternative suggestions tried to consider the amount of effort required to implement versus the potential benefit from implementing the mitigation. However, the final decision as to the most appropriate mitigation option, if any, is left to the Sage project to consider. Such decision is based on a cost/benefit analysis that depends on many circumstances mostly outside the scope of this paper. For example, one environment which already has certain infrastructure in place may consider one option relatively cheap and easy to implement, whilst in a different environment such infrastructure would be costly to acquire. Similar arguments were mentioned on section 2.3.3.

Furthermore, some mitigation options may have other security, performance or other indirect impact. For example, implementing SSL may be highly desirable, but might potentially introduce some performance bottlenecks and increase the chances of Denial of Service. Attackers can abuse the SSL handshake mechanism, which is relatively 'costly' in terms of processing on the server [9]. Other examples include a typical trade-off when designing lock-out periods for authentication failures. With longer lock-out period and fewer failed attempts acceptable, the chances of a successful brute-force or dictionary attacks are reduced, but at the cost of potential denial of service, by locking out multiple accounts on the system. The implications and other potential indirect impacts need to be further considered by the Sage project before implementing a mitigation.

2.4 Threat Model Analysis

The process described on section 2.3 is highly structured and therefore fairly rigid. It goes through each and every element and lists potential threats and mitigations. As such, the output from the threat modelling process makes it difficult to 'see the wood for the trees'. The threat models in chapters 3 and 4 highlight a number of threats and vulnerabilities, and suggest numerous mitigation options. However, going through each element, and a list of threats for the elements can become difficult to digest. Chapter 5 attempts to provide further analysis and interpretation of the information on the threat models in a more 'digestable' form. The analysis tries to distill the key findings and provide more detailed information and further insight into the most crucial threats, vulnerabilities and mitigation options. It also attempts to identify themes and threads going through more than one threat and applying to several elements. Such themes are harder to identify when viewing each threat independently.

2.5 Caveats and Limitations

As robust, thorough, accurate or methodical as any process can be, there are always limitations, issues and caveats to consider and be aware of. It is important to understand the limitations and caveats of the threat modelling process

in order to get the right value out of it, and particularly so to avoid a false sense of security. Any process that attempts to map or identify weaknesses, deficiencies or vulnerabilities is particularly prone to such conditions. A failure to identify an issue of such nature may lead to the conclusion that the issue does not exist, or does not need addressing. However, the lack of *identification* of a problem, does not make the problem go away. 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. In an ideal situation, the output of the threat modelling process is acted on fully, all threats are mitigated to an acceptable level and no threats remain unaccounted for. However, what if the threat model itself failed to identify a deficiency or a threat? The threat can then go unmitigated. If (or once) the unidentified threat materialises, the system can be compromised. This concern over unidentified or unaccounted for threats is much greater considering the changes in technology, attacker's abilities and motivation. Systems are rarely static, and the environment keeps changing. As new threats emerge and attacks get better, the threat model may become obsolete in a relatively short period of time. Considering and realising there may be unknown threats, and understanding that the threat model is neither permanent, nor gap-free is therefore highly important.

A typical threat model may have difficulty covering or giving assurances in the following areas:

- **Secure coding practices** - a threat model is unlikely to discover specific insecure coding issues which a security code review, audit or static analysis tools may help identify.
- **Vulnerabilities** - Threats identified may be exploited if a vulnerability exists, but the threat model does not prove existence of vulnerabilities nor their exploitability. A penetration test may give better indication of such conditions.
- **Dependencies** - A part of the threat modelling process identifies external dependencies (see section 2.3.1). Other dependencies and assumptions may not be explicitly covered, but are still made. In such circumstances, threats or vulnerabilities to any of those external factors might not be fully considered as part of the modelling process.
- **Future threats and patterns** - exploits and attacks evolve and change constantly. Attackers change their patterns or interest and new 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 included in the top 10 list in the previous few years [14]. As such, not only do new threats and exploit techniques emerge, existing threats may change their profile and rating over time.
- **Human elements** - Some human elements are covered by a typical threat model. For example, spoofing of an external entity or a process takes into account potential to tricking humans to place trust where they should not. However, some human elements are not fully addressed by the threat

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

- **Broader risks** - Other aspects, such as those pertaining to organisational risks, operational risks, change management, documentation and knowledge sharing, environmental risks, disaster recovery and business continuity are also not easy to identify and address using the threat modelling methodology. Threat modelling tends to be application / system / process centric and it is therefore not particularly suitable to address wider areas of information security risks.

Chapter 3

Development Process Threat Model

3.1 Threat Model Information

This threat model tries to capture threats to the Sage open source *development and contribution* processes. The threat model is focused on the development process, code changes / contributions, and distribution of code and packages. The sage development process is not radically different from many other open source applications or systems, or even not purely open source ones. However, the model captures at a high level the data flows and some of the technology and manual processes that apply to Sage.

The Sage development process is fairly robust, and a great degree of care and consideration has already been put into it. Every bit of code must get reviewed and 'earn' its way into the codebase. The primary concern is to avoid buggy, incorrect or even not fully tested and documented code from entering into the product. The standards set and followed by the community seem to be fairly high, and the review process in most cases is thorough.

However, the threat model shows there are still ways of bypassing at least some aspects of the review process (e.g. faking positive reviews, modifying code or Trac records after review etc). How feasible or likely these threats to become attacks depends on the level of sophistication, effort and determination of a potential attacker, as well as the existence or absence of vulnerabilities in some of the underlying processes and / or technology. The base assumption is that there is still fairly low motivation to sabotage Sage in such a way. As such, most of these threats are considered low risk.

The threat model does not cover an element which perhaps could be considered as an 'anti-threat', i.e. the fact that Sage is an open source, means that contributors can identify security vulnerabilities in the code, and contribute changes or new code to reduce or remove those vulnerabilities.

3.2 Data Flow Diagrams

3.2.1 Sage Open Source Development Process

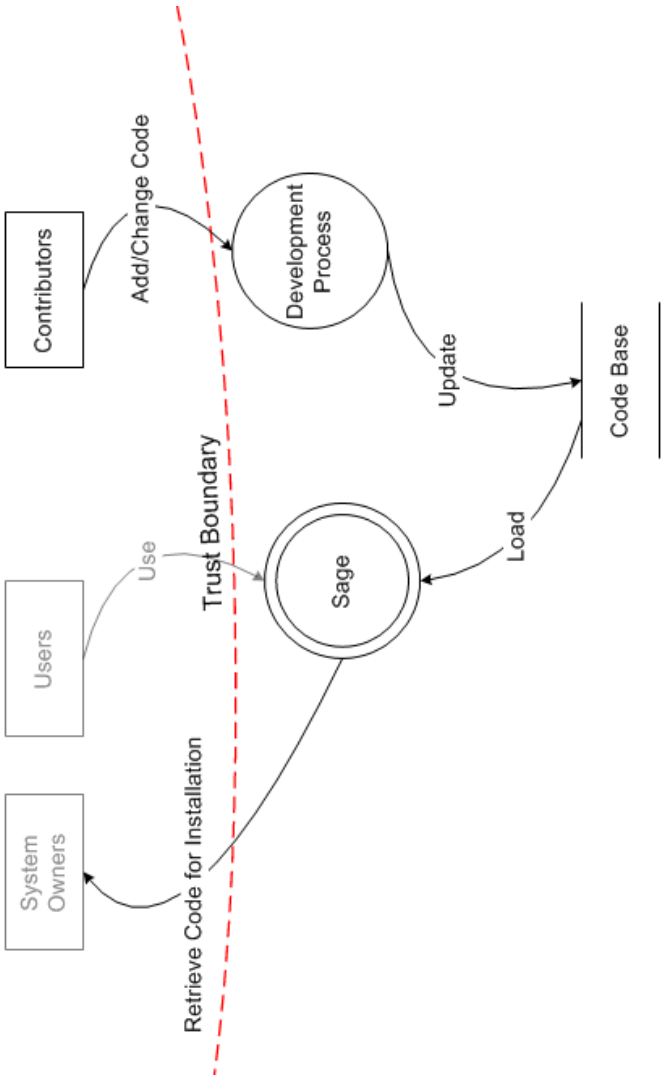

Figure 3.1: Open Source Development Process Data Flow Diagram

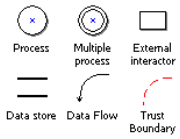

Figure 3.2: Legend

3.3 Threats and Mitigations

3.3.1 Elements

ID	Type	Name
38	DataFlow	Add/Change Code
40	DataFlow	Load
42	DataFlow	Retrieve Code for Installation
39	DataFlow	Update
36	DataFlow (Out of Scope)	Use
37	DataStore	Code Base
33.3	Interactor	Contributors
33.14	Interactor (Out of Scope)	System Owners
33	Interactor (Out of Scope)	Users
32	MultiProcess	Sage
34	Process	Development Process
35	TrustBoundary	Trust Boundary

3.3.2 Add/Change Code [DataFlow 38: Contributors → Development Process]

Threat	
ID	48
Type	Tampering
Risk Rating	Low
Description	Code submitted might contain deliberate or accidental security vulnerabilities.
Mitigation	Code is being reviewed and validated by at least one Sage contributor. The code review does not necessarily focus on security, but mostly on good coding practices and correctness. Obvious security 'backdoors' are unlikely to filter through, but less obvious security bugs and vulnerabilities may still get included in the code base unnoticed.

Threat	
ID	110
Type	Tampering
Risk Rating	Medium
Description	External packages are not code reviewed. This presents an easier route to inject malicious code into Sage, by tampering with the external package. Note the external dependency on external code being secure (the Sage project cannot realistically fix security issues in all of its components). However, this threat is about tampering with the packaged external code, i.e. introducing vulnerabilities that may not exist in the external component in the first place. (All third party packages in Sage are also Open Source and must have a compatible license).
Mitigation	Ensure packages of third party code are generated securely from trustworthy code retrieved and updated from the third party. Spot checks on packages.

DenialOfService: No Threat Certification	
Description	Extremely unlikely that any DoS would affect the code contribution process as a whole. Process is long and does not require high availability.

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so code is always public.

3.3.3 Load [DataFlow 40: Code Base → Sage]

Threat	
ID	54
Type	Tampering
Risk Rating	Low
Description	Code from mercurial version control system might be tampered with when being loaded / packaged for release.
Mitigation	Code changes are visible on inclusion as patch files, and the integration is performed by a trustworthy release manager. It is assumed unlikely (but not completely infeasible) that the release manager will tamper with code, or that the communication link or storage of the code base will be intercepted and / or tampered with

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so code is always public.

DenialOfService: No Threat Certification	
Description	Very little impact of DoS for loading new code to the Sage public website.

3.3.4 Retrieve Code for Installation [DataFlow 42: Sage → System Owners]

Threat	
ID	106
Type	Tampering
Risk Rating	Low
Description	Tampering with downloaded code can allow injection of malicious components. Sage is provided in many available formats, including code, virtual machine image and packages for various OS.
Mitigation	MD5 signatures are published together with the downloadable tar files, but if data tampering is achieved, those signatures could easily be forged.

Threat	
ID	108
Type	DenialOfService
Risk Rating	Low
Description	DoS on the public Sage website may prevent potential users from obtaining code. However, code can be replicated across many servers online.
Mitigation	Threat is minimal. Code/distribution packages are available on many locations online (mirror sites)

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so code is always public.

3.3.5 Update [DataFlow 39: Development Process → Code Base]

Threat	
ID	51
Type	Tampering
Risk Rating	Low
Description	Threat of tampering / intercepting communication between the release manager and the server where the final packaged version is stored before release.
Mitigation	Assuming SCP or another secure method of sending the file(s), this seems like a negligible risk

Threat	
ID	53
Type	DenialOfService
Risk Rating	Low
Description	Extremely unlikely that any DoS would affect the code base, which can also be replicated easily.
Mitigation	Threat is likely to be acceptable.

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so code is always public.

3.3.6 Use [DataFlow 36: Users → Sage]

Out of Scope	
Reason	Usage Threats are covered in chapter 4

3.3.7 Code Base [DataStore 37]

Threat	
ID	44
Type	Tampering
Risk Rating	Low
Description	All packaged releases (including older releases) are stored on sage.math. Tampering with those copies may lead to tainted versions of Sage being released, or even carried over on future releases.
Mitigation	Access to the packages is restricted to a few trustworthy and named individuals, and therefore the threat is considered low risk.

Threat	
ID	45
Type	Repudiation
Risk Rating	Low
Description	The code repository does have an element of logging (recording of who committed which changes and additions to the code base). If tampered with can also lead to repudiation threats. See Tampering threat ID 44.
Mitigation	See threat ID 44.

Threat	
ID	47
Type	DenialOfService
Risk Rating	Low
Description	Extremely unlikely that any DoS would affect the code base, which can also be replicated easily.
Mitigation	Threat is likely to be acceptable.

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so code is always public.

3.3.8 Contributors [Interactor 33.3]

Threat	
ID	33
Type	Spoofing
Risk Rating	Low
Description	Authentication processes aren't very robust (users are generated by request / email / IRC), but there isn't much to gain by spoofing an identity. All users have the same level of access and access is easily obtained.
Mitigation	This is a negligible threat. However, if deemed necessary, more robust online or offline authentication processes can be employed to reduce authentication threats.

Repudiation: No Threat Certification	
Description	No obvious threat or gain by repudiating actions. Contributors are primarily volunteers, any 'mistake' should be spotted by fairly rigorous code review process to prevent unstable/wrong/erroneous code from being accepted.

3.3.9 System Owners [Interactor 33.14]

Out of Scope	
Reason	Anonymous web users who download the Sage code and use it

3.3.10 Users [Interactor 33]

Out of Scope	
Reason	Usage Threats are covered in chapter 4

3.3.11 Sage [MultiProcess 32]

Threat	
ID	25
Type	Spoofing
Risk Rating	Low
Description	DNS spoofing or redirection can lead to a fake website. This can be used to distribute tampered code / packages and / or damage the project reputation. Low motivation and relatively high degree of effort is involved.
Mitigation	Can use SSL trusted certificates, e.g. by Verisign.

Threat	
ID	26
Type	Tampering
Description	Tampering threats on Sage Notebook are covered in chapter 4.
Mitigation	None

Threat	
ID	27
Type	Repudiation
Risk Rating	Low
Description	The public facing website is managed by a limited number of trusted individuals. Any repudiation threats to website changes/actions is therefore limited.
Mitigation	Other than placing trust in individuals, logging mechanisms and change control processes can be used to increase auditability of actions and reduce repudiation threats.

Threat	
ID	109
Type	Tampering
Risk Rating	Low
Description	Public facing website holds static data and is unlikely to be a subject for attack (however see spoofing threat as well as tampering threats on 'Retrieve Code for Installation' data flow).
Mitigation	'Standard' public website security best practice should apply (patching, monitoring etc) - see dependencies

InformationDisclosure: No Threat Certification	
Description	This is an open source system, so no information is withheld or secret.

DenialOfService: No Threat Certification	
Description	Very little impact achieved by preventing access to the Sage public server, apart from reputation and unavailability for system owners / users.

ElevationOfPrivilege: No Threat Certification	
Description	Public website does not have different levels of privileges. Read-Only website. Sage Notebook threats are covered on a separate threat model.

3.3.12 Development Process [Process 34]

Threat	
ID	35
Type	Spoofing
Risk Rating	Low
Description	Adding users to Trac (wiki and ticketing system) is fairly informal and involves emailing or IRC chats. All channels are currently unencrypted / unauthenticated. Passwords are known to more than one person (the user and the administrator creating the account), do not seem to follow a particular policy and cannot be changed by the user without assistance from the Trac administrator. Gaining access to Trac however does not provide much other than ability to interact with open tickets and submit code patches. Spoofing an identity would not achieve much from an attacker's perspective.
Mitigation	Current process may be considered acceptable.

Threat	
ID	36
Type	Tampering
Risk Rating	Medium
Description	Altering data in Trac tickets may be relatively easy to accomplish. 'Faking' positive code reviews and/or changing patches/code already reviewed may be easy. Such tampering can lead to insecure or malicious code included in the code base.
Mitigation	Trac does have internal revision control system, which should record any changes to tickets, who initiated the change and when. Circumventing the Trac internal controls may still be possible however. The main focus should be on including code and ensuring the reviews are genuine and that code was not tampered with after review. A manual process / spot checking is already performed by the release manager, but can potentially be further enhanced, particularly to better focus on security (e.g. include a second reviewer, whose task is to spot any security issues which may affect the release).

Threat	
ID	37
Type	Repudiation
Risk Rating	Low
Description	Whilst Trac / Mercurial both associate any change to individual user, it is not infeasible to circumvent its mechanisms or abuse privileges on the system and attribute them to another user.
Mitigation	Relatively high level of involvement of individuals on the project mean that each developer / user may spot any changes made on their behalf.

DenialOfService: No Threat Certification	
Description	Development process is fairly disconnected and not reliant on a centrally available system. Other than the most extreme DoS scenario (which is highly unlikely), DoS threats are deemed acceptable.

ElevationOfPrivilege: No Threat Certification	
Description	There are no different levels of privileges. Any 'privileges' are defined outside automated processes, and assigned to highly trusted individuals.

InformationDisclosure: No Threat Certification	
Description	Sage is an open source system and all information is readily available.

3.4 External Dependencies

ID	Name	Origin
1	Trac	External
2	Mercurial	External
3	Apache (with mod_proxy)	External
4	Ubuntu Linux	External
5	Third Party packages (bundled with Sage)	External

3.5 Implementation Assumptions

ID	Element Impacted	Assumption
1	All	Server OS running Sage servers is patched and configured securely
2	All	DNS is hosted externally and secured for unauthorised updates/modification

3.6 External Security Notes

There are no external security notes.

Chapter 4

Sage Notebook Threat Model

4.1 Threat Model Information

This threat model tries to capture threats to the Sage Notebook system. Sage and the Sage Notebook can be installed on many supported platforms and in different configurations. In order to keep the analysis focused and more effective, one deployment / setup was chosen to represent a 'typical' Sage Notebook installation. The typical configuration is based on the Sage default 'out-of-the-box' settings (also called 'Vanilla'), but also referring to the Sage Notebook public server [93], which is available for users to experiment and use, and is maintained by the Sage project. One of the main differences between the 'out-of-the-box' settings and the Sage Notebook public server, is that the latter is running on a virtual platform [100]. Running the server in a virtual environment provides some of the benefits covered in more detail on the analysis in Chapter 5, section 5.4.4. Amongst the benefits, the ability to limit resource usage, monitoring, and the ability to start from 'scratch' by restoring from a snapshot, in case of an issue or corruption. The Sage project does not give much guarantees or warranty as to the availability, confidentiality or integrity of data on the sagenb.org server, and it is provided on an 'as-is' basis.

The components and processes represented in the model remain at a fairy high level. Whilst a more detailed analysis and further breakdown of components may have been possible, the threat model is still effective, as it uncovers numerous threats and vulnerabilities. Some of the threats are fundamental at the architecture and design level, whilst others are more specific to implementation methods and environment configuration. It is therefore felt that the level of analysis and the modelling process is sufficient at this stage and further breakdown will not be cost-effective.

Unlike the Sage Open Source development process, which is relatively similar to other open or closed source projects (covered in Chapter 3), the Sage Notebook service provides unique functionality not likely to be found on other web-based applications. The flexibility given to essentially anonymous users,

in terms of allowed functionality, and specifically: running code directly on the server, presents interesting challenges from a security perspective. This flexibility 'violates' basic security design tenets, such as preventing access to the layer below [34] and the least privilege principle [81]. The threat model identifies several areas which can benefit from architectural and implementation improvements. It also raises some ideas on improving security which aren't necessarily technical measures - simply by reducing available functionality to users (or at least not all users). Nevertheless, the threat model recognises the desire of the Sage community to remain open, allow flexibility and provide more, rather than less functionality and power to its users. The decision remains with the Sage community and each of its users and installation owners (Universities, organisations or individuals choosing to deploy and use Sage). However, it is hoped that a more informed decision can be taken based on this threat analysis.

Some of the threats and vulnerabilities require substantial effort and re-design to fix, and therefore may not necessarily get implemented. However, many of the threats can be mitigated by relatively simple means. For example, ensuring SSL is used between users and the Sage Notebook would completely eliminate a number of threats. The ability to run the Sage Notebook over HTTPS is available to users and does not require any modification of the code base.

4.2 Data Flow Diagrams

4.2.1 Sage Notebook

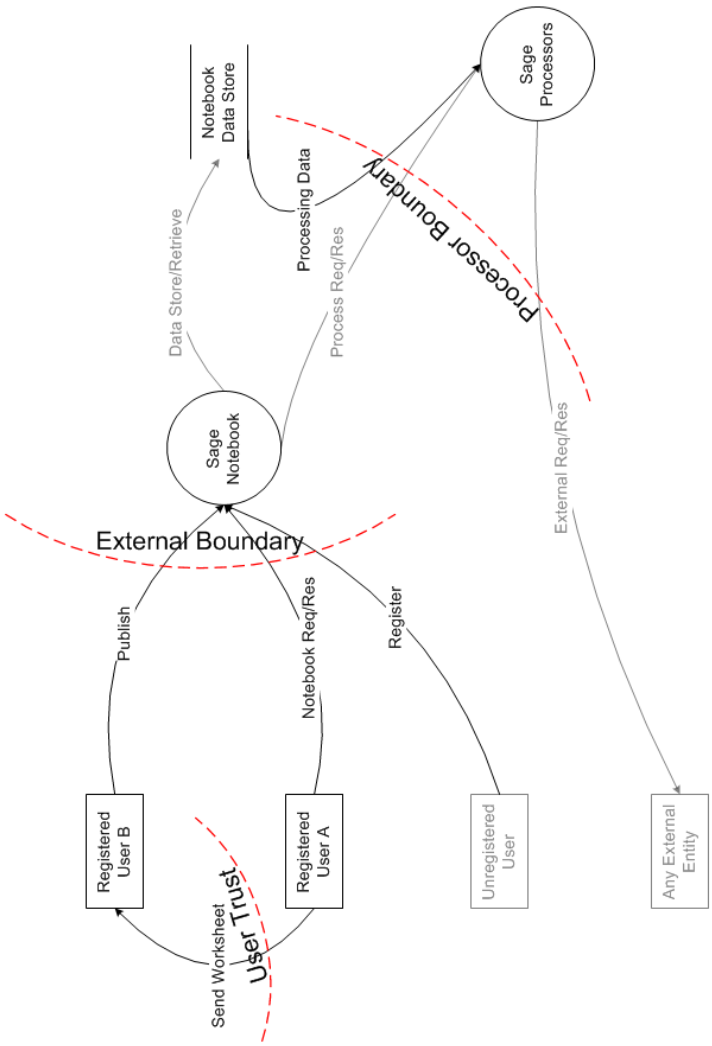

Figure 4.1: Sage Notebook Data Flow Diagram

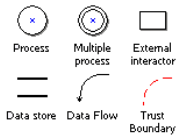

Figure 4.2: Legend

4.3 Threats and Mitigations

4.3.1 Elements

ID	Type	Name
21	DataFlow (Out of Scope)	Data Store/Retrieve
20	DataFlow (Out of Scope)	External Req/Res
16	DataFlow	Notebook Req/Res
18	DataFlow (Out of Scope)	Process Req/Res
25	DataFlow	Processing Data
14	DataFlow	Publish
17	DataFlow	Register
15	DataFlow	Send Worksheet
13	DataStore	Notebook Data Store
19	Interactor (Out of Scope)	Any External Entity
9	Interactor	Registered User A
8	Interactor	Registered User B
10	Interactor (Out of Scope)	Unregistered User
11	Process	Sage Notebook
12	Process	Sage Processors
23	TrustBoundary	External Boundary
24	TrustBoundary	Processor Boundary
22	TrustBoundary	User Trust

4.3.2 Data Store/Retrieve [DataFlow 21: Sage Notebook → Notebook Data Store]

Out of Scope	
Reason	Data storage and retrieval takes place on the same host and within the same trust boundary. Any threats to the internal storage channel are either attributed to a lower-level dependency (e.g. Operating System security) or is already accounted for on the Sage Notebook / Notebook Data objects.

4.3.3 External Req/Res [DataFlow 20: Sage Processors → Any External Entity]

Out of Scope	
Reason	External requests/responses data flow represents unauthorised use case, and are there to highlight potential for attacks on the Sage Processors of affecting other external entities. Such threat is an indirect threat, as it does not affect any of the Sage entities. However, it is a threat not to ignore as it might have severe impact on the project, its reputation, and the various installation hosts/sites. There are no direct threats to the data flow itself and therefore it is out of scope. Threats and mitigation are already covered on the Sage Processors entity (see threat IDs 38-42)

4.3.4 Notebook Req/Res [DataFlow 16: Registered User A → Sage Notebook]

Threat	
ID	53
Type	Tampering
Risk Rating	Medium
Description	Communication between users and the Sage Notbook is performed over HTTP, but can be SSL protected. However, by default, and on the sagenb.org public web server, no SSL is currently being used. User requests can be relatively easily tampered with.
Mitigation	It is not clear how much can be gained by tampering with user requests, particularly considering the fact that other attacks (such as stealing the user credentials, see threat ID 54) may be more effective and easier to accomplish. Tampering threats on communication links could easily be mitigated by using SSL with a trusted certificate.

Threat	
ID	54
Type	InformationDisclosure
Risk Rating	High
Description	All web server communication is over HTTP by default - sniffing the network can allow obtaining authentication credentials. In some use cases, such as in some research environments, the computation data itself may be sensitive or confidential. Protection of data from eavesdropping may be highly desirable in such situations. Sniffing the network is relatively easily achievable within a small physical environment, such as departmental network. The Sage public Notebook server (sagenb.org) does not currently use HTTPS. Other Notebook installation can run with SSL if required.
Mitigation	Use SSL with a trusted public CA certificate.

DenialOfService: No Threat Certification	
Description	Denial of service attack on the link to the Sage Notebook public server (sagenb.org) is possible, but unlikely. Other forms of denial of service are available, which are easier to accomplish. This threat is considered low and acceptable.

4.3.5 Process Req/Res [DataFlow 18: Sage Notebook → Sage Processors]

Out of Scope	
Reason	Sage Notebook and the processors are considered to be on a separate trust boundaries. However, the link between the two is usually secure using Secure Shell (SSH) [113] with keys and contained within a small physical location, some times on the very same host. Therefore, there are virtually no threats to the link itself. Other threats are covered on the Sage Notebook and Sage Processors entities.

4.3.6 Processing Data [DataFlow 25: Notebook Data Store → Sage Processors]

Threat	
ID	70
Type	Tampering
Risk Rating	High

Description	Using the default set-up of Sage Notebook, Processors (e.g. Sage, Python, r, Shell) run on the same host as the Notebook server. However, code running on the processors is user-controlled and therefore untrusted. In this situation the trust boundary crosses the same host and puts the Notebook and other processes at much higher risk. Malicious user code can easily access the filesystem or run system calls. It can therefore tamper with the Notebook Data Store and other processes (including the Notebook process). Even if the processors are running on a separate host or in tighter isolation (e.g. chroot'd jailed environment on the same host or virtualised), as long as the processors have access to the same file system, many of the aforementioned threats apply.
Mitigation	Separation of processors from other components is highly desirable. Restrictions and constraints on the running processors in addition to limits imposed on users should also be considered carefully. File system access should not be shared between the Notebook data and processor data (effectively removing this data flow completely). Limit filesystem quota per process, allow only temporary storage space, separated via filesystem permissions between processors. Also see threats to Sage Processors.

Threat	
ID	71
Type	InformationDisclosure
Risk Rating	High
Description	See tampering threat ID 70. Information Disclosure may also result from similar conditions.
Mitigation	See tampering threat ID 70 mitigation.

Threat	
ID	72
Type	DenialOfService
Risk Rating	Medium
Description	Sage Processors can relatively easily be used to launch a denial of service attack, either on external entities (see Sage Processors threats), or on shared storage area. Attacker-supplied code can very easily delete files on the filesystem, create masses of data and store it on the filesystem to create a Denial of Service attack.
Mitigation	Limit filesystem quota per process, allow only temporary storage space, separated between processors. Prevent sharing of filesystem space between processors and Notebook data.

4.3.7 Publish [DataFlow 14: Registered User B → Sage Notebook]

Threat	
ID	47
Type	Tampering
Risk Rating	High
Description	Tampering with published worksheets are assumed to be a 'higher value' target, particularly if tampering results in injected code to be executed on the client side - e.g. javascript. The more trustworthy or popular the publisher, the higher the motivation to tamper with their published worksheets, since those worksheets are more likely to be opened and copied by other Sage users. Injecting a javascript worm onto such worksheet can potentially allow the attacker to take over many accounts on the system. The communication is over HTTP by default and the Sage public Notebook server does not currently use HTTPS. Note that the high impact of this threat is not purely achieved by tampering the data flow, but rather relies on injection / Cross Site Scripting (XSS) [10] vulnerabilities on the application as well.
Mitigation	Notebook supports SSL/HTTPS. Use SSL with a trusted public CA certificate to provide channel integrity and data origin authentication.

InformationDisclosure: No Threat Certification	
Description	published worksheets are meant to be public, therefore no information disclosure threats are applicable.

DenialOfService: No Threat Certification	
Description	Any Denial of Service threats are not specific to the publish channel and considered fairly minimal.

4.3.8 Register [DataFlow 17: Unregistered User → Sage Notebook]

Threat	
ID	57
Type	InformationDisclosure
Risk Rating	Medium
Description	Communication links are unencrypted and unauthenticated. Usernames and passwords used through registration process can be relatively easily obtained through network sniffing, arp / dns spoofing, phishing / fake registration screens etc.
Mitigation	Implement SSL with a certificate issued by a trusted CA (e.g. Verisign)

Threat	
ID	58
Type	DenialOfService
Risk Rating	Low
Description	The registration process does not use email verification, captchas or any anti-scripting mechanisms. It should be very easy to 'bombard' the system with spurious registrations, contaminating the user storage and potentially leading to denial of service of the registration process for legitimate users.
Mitigation	Use captchas to prevent scripting. Use a two-step registration process: First step: provide email address A confirmation email will be sent out to the email with a random string. Second step: confirm email and complete registration. Note: The email generation itself might become a target for another type of DoS attack (e.g. by trying to register with someone else's email address, causing them to receive unwanted emails). Alternatively, when using a 3rd party user directory for authentication, no registration / enrolment is necessary, and registration can be disabled on the Sage Notebook (an option already available). Similarly in smaller user groups, registration can remain disabled and only re-enabled temporarily for an 'enrolment time window'.

Tampering: No Threat Certification	
Description	Tampering a registration request / response would not achieve much other than perhaps DoS (see DoS threats). Any user can generate registration requests.

4.3.9 Send Worksheet [DataFlow 15: Registered User A → Registered User B]

Threat	
ID	50
Type	Tampering
Risk Rating	Medium
Description	Worksheets are not integrity protected or authenticated, so can easily be tampered with when being sent from one user to another. Tampering can include malicious code which would then be executed using the recipient's credentials if uploaded to Sage. If the recipient runs Sage locally, the malicious code will execute on their local machine.
Mitigation	Add some form of integrity and data origin authentication to worksheets (e.g. HMAC with a shared key, or public key digital signatures) Users should be made aware of the risk of accepting worksheets, especially from untrusted sources, or when there is higher likelihood of worksheets being tampered with. (see external security note 1) When trust exists between two (or more) users, third party and non Sage-specific security tools can be used to prevent tampering - e.g. PGP, S/MIME.

Threat	
ID	51
Type	InformationDisclosure
Risk Rating	Medium
Description	Worksheets are not encrypted, so can easily be captured over the network (e.g. by sniffing) when being sent from one user to another. In some situations, confidentiality of worksheet data is a concern.
Mitigation	Third party and non Sage-specific security tools can be used to prevent information disclosure - e.g. PGP, SCP, SFTP. Information Disclosure affecting user-to-user communication is outside the direct scope of the Sage Notebook server. However, perhaps Sage can enforce encryption of worksheets when exported.

DenialOfService: No Threat Certification	
Description	There are many possible channels to send, receive and share worksheets between users.

4.3.10 Notebook Data Store [DataStore 13]

Threat	
ID	43

Type	Tampering
Risk Rating	High
Description	Notebook Data is stored on the filesystem of the same host and within the same trust boundary. However, notebook data (stored in .sobj files, which are essentially python 'pickled' objects) may contain untrusted code injected into the file. When notebook data is loaded from file (Python 'unpickling' process), it can execute arbitrary code running on the Sage Notebook process. .sobj files are normally created by the Sage Notebook itself, and hence would not normally contain malicious code. However, notebook data can be imported into Sage. Python programs must never unpickle data from untrusted or unauthenticated source [76] [59]
Mitigation	Consider an alternative mechanism for pickling to store notebook / worksheet data. Implement robust validation and santisation of sobj files and constrain the unpickling process to validated, trusted data. Avoid uploaded data unpickling and ensure sobj files are generated with validated data so that no code can be injected and then executed when unpickled. Beware the false sense of security provided by __safe_for_unpickling__ set to 1, or by being registered in a global registry, copy_reg.safe_constructors [104]

Threat	
ID	44
Type	Repudiation
Risk Rating	Low
Description	Only minimal logging of security or system events is performed, and the logging is not integrity protected or secured. An attacker may be able to easily manipulate any log records stored on the file system.
Mitigation	This is considered a relatively low priority issue at this stage, however more robust logging should be considered, perhaps also incorporating logging into external sources (e.g. SYSLOG)

Threat	
ID	45
Type	InformationDisclosure
Risk Rating	Medium
Description	Notebook Data is stored on the filesystem of the same host and within the same trust boundary. Information disclosure can however be achieved if data files are accessed either directly or indirectly (e.g. utilising a flaw in the Notebook process). File permissions and ownership is currently set so that any Notebook process can access all notebook files of all users. Also see threat ID 43 covering potential pickling issues, which can also lead to information disclosure. In addition, if filesystem is shared between the Sage Notebook and Processors, without privilege separation on the filesystem, an attacker can access worksheet data as well as Notebook username and passwords by using the processors (see threat ID 40)
Mitigation	Utilise separation of privileges on the filesystem or use a database. Tie each Sage Processor process and the notebook process to different system user ID for each session, e.g. using sudo -u to assume a user's identity for a process/operation. Resolve any known vulnerabilities or functionality which may lead to direct file access.

Threat	
ID	46
Type	DenialOfService
Risk Rating	Low
Description	There are no storage limits or quotas set on the filesystem. Users may potentially fill the storage area so there's no space left, leading to a Denial of Service.
Mitigation	Consider applying user-based quotas on the filesystem and / or on the Notebook process to prevent maxing out of disk space. Use monitoring to alert of low disk space and allow quick mitigation before space is fully exhausted.

4.3.11 Any External Entity [Interactor 19]

Out of Scope	
Reason	This entity is a placeholder to represent any external entity / system on the Internet, which may be subject to attack by processes running on Sage (see threats related to Sage Processors and External Req/Res Data Flow)

4.3.12 Registered User A [Interactor 9]

Threat	
ID	28
Type	Repudiation
Risk Rating	Low
Description	The ability to trace back to individuals and track actions is very limited and virtually non-existent. However, before a more robust authentication and authorisation mechanisms are put in place, it's hard to imagine very robust non-repudiation controls being implemented.
Mitigation	Any solution which logs user actions should be aware of user privacy concerns and enforce the necessary means to balance between repudiation threats and threats to user privacy.

Threat	
ID	73
Type	Spoofing
Risk Rating	High
Description	Bruteforce / Dictionary attack on users / passwords - No lockout mechanism in place. Users are free to choose trivial passwords.
Mitigation	Improve authentication mechanism to include stronger password controls, including: • password complexity requirements • failed authentication lockout or (preferably) a time-out mechanism to drastically reduce the effectiveness of automated attacks, whilst still allowing legitimate access • minimum password length • logging / alerting mechanisms to monitor attacks Usage of captchas or other anti-scripting techniques can increase the complexity of scripting attacks. Alternatively, or in addition, use a 3rd party back-end authentication (e.g. LDAP [54] [36])

Threat	
ID	74
Type	Spoofing
Risk Rating	High
Description	Cross Site Scripting - javascript code can easily get embedded in worksheets, which can be published and shared, allowing injecting malicious code and obtaining session cookies.
Mitigation	Consider disallowing javascript code and using input validation and anti-scripting mechanisms, e.g. libraries, input filtering, white/blacklisting, stripping of invalid characters. Note: Character filtering may seriously hinder the proper operation, particularly when mathematical forumlas are used. In addition, the ability to use javascript is currently enabled deliberately and almost encouraged, to create more interactive worksheets, and give the user higher degree of flexibility. Any filtering needs to take into account users ability to generate javascript 'on-the-fly' from python or other code allowed to run on the backend processors. A decision needs to be made to balance security and flexibility. Perhaps a compromise could involve limiting input based on user privilege level(s). i.e. Certain users will be permitted greater flexibility than others (those users will therefore need to be highly trusted)

Threat	
ID	78
Type	Spoofing
Risk Rating	Low
Description	Standard random functions are used for HTTP sessions. Such pseudo-random sources are considered insecure. Predicting session values can lead to session hijacking and spoofing.
Mitigation	Use more cryptographically robust random generators / sources.

4.3.13 Registered User B [Interactor 8]

Threat	
ID	25
Type	Spoofing
Risk Rating	Low
Description	Spoofing of identities and trust issues between Users (i.e. User A and User B) are to some extent out of scope of the Sage Notebook. However, sharing or distribution of worksheets is not authenticated or digitally signed.
Mitigation	Consider implementing a mechanism for verification of integrity and data origin authentication of worksheets

Threat	
ID	26
Type	Repudiation
Description	see threat ID 25 (Spoofing Registered User B)
Mitigation	see threat ID 25 (Spoofing Registered User B)

Threat	
ID	75
Type	Spoofing
Description	see spoofing threats for Registered User A (threat IDs 73, 74, 78)
Mitigation	see spoofing threats for Registered User A (threat IDs 73, 74, 78)

Threat	
ID	76
Type	Repudiation
Description	see repudiation threats for Registered User A (28)
Mitigation	see repudiation threats for Registered User A (28)

4.3.14 Unregistered User [Interactor 10]

Out of Scope	
Reason	Unregistered user cannot be spoofed or repudiated. There are no threats *to* the external user

4.3.15 Sage Notebook [Process 11]

Threat	
ID	31
Type	Spoofing
Risk Rating	Medium
Description	Creating a clone of the Sage Notebook server is relatively easy, since the code is readily available and anybody can install it to create an exact copy of the spoofed server. Spoofing can be achieved by various means, e.g. DNS spoofing, Man-in-the-Middle, Redirecting etc. The notebook server can be configured with SSL support. However, the public key certificates generated are self-signed, hence can also created by an attacker. Spoofing by itself would not achieve much, but can be used to assist in obtaining authentication credentials, tampering and other attacks.
Mitigation	Use SSL with trusted certificates. Educate users to pay attention to certificate details, URL address bar etc.

Threat	
ID	32
Type	Tampering
Risk Rating	High
Description	Very limited input validation is performed on the Notebook server for worksheet data, and Cross Site Scripting (XSS) is currently easily achievable using javascript inside HTML embedded in worksheets (either uploaded, entered directly, or published and shared with all or some other notebook users). Cross site scripting is one of the dominant and most powerful attacks against web applications [14], as they allow session hijacking, unauthorised actions and distributed denial of service attacks. XSS attacks usually involve clever crafting of input to bypass any filtering and input/output encoding. However, Sage notebook does allow javascript code to be included by design, substantially reducing the complexity of such potential attack.
Mitigation	Consider removing functionality which allows javascript or other browser executable code. Introduce input and output filtering and anti-XSS libraries. Scripting functionality may be enabled only on certain set-ups, or to specific trusted users as a compromise, as long as the risk is understood and communicated to notebook users. See external security note 2.

Threat	
ID	33
Type	Repudiation
Risk Rating	Low
Description	System logging on the Notebook is very minimal if not non-existent. Tracing back to user actions is therefore extremely difficult or impossible.
Mitigation	Implement logging mechanisms tied to identities of users. Consider sending log items to a separate host/entity (e.g. using SYSLOG), to avoid tampering with log records on the filesystem. Any solution which logs user actions should be aware of user privacy concerns and enforce the necessary means to balance between repudiation threats and threats to user privacy.

Threat	
ID	34
Type	InformationDisclosure
Risk Rating	Medium
Description	Notebook user passwords are stored using Unix crypt mechanism [19] (based on a modified DES algorithm), and with static hard-coded salt value. Password hashes, if obtained, can easily be dictionary attacked or brute-forced.
Mitigation	Implement a more robust one-way hashing with variable salts to store user passwords. Consider using 3rd party user repositories for authentication and authorisation, so to remove password storage inside the Notebook completely (e.g. LDAP, Unix authentication, Kerberos [53])

Threat	
ID	35
Type	DenialOfService
Risk Rating	Low
Description	The Notebook system does not enforce any processing, storage space, bandwidth or other usage limits. Users can potentially abuse the system and create many worksheets, upload very large files and launch other attacks which may result in DoS. In addition, processors may generate load if placed on the same host (for other processor related threats, please see Sage Processors).
Mitigation	Consider implementing some level of throttling or limits to prevent system resources being over-utilised to result in DoS.

Threat	
ID	36
Type	ElevationOfPrivilege
Risk Rating	Low

Description	Privilege levels on the system are basic and minimal. There are three types of users: • admin - Administrative user, can set Notebook settings and access all resources • user - a registered user account, may access their own worksheets or those explicitly shared with them (as collaborators) • guest - an unregistered user, may only view public worksheets and cannot make any changes It is unclear whether there are any vulnerabilities which could expose the Notebook server to elevation of privilege. However, the entire authentication and authorisation functionality was implemented inside the notebook (instead of using a 3rd party component).
Mitigation	Using a third party authentication and authorisation modules can greatly increase the confidence level in its robustness (as long as the third party component is well known and was subject to more scrutiny). A number of third party components to consider, include but are not limited to: • LDAP, e.g. OpenLDAP [70], Active Directory [63] • Kerberos • Linux/Unix users and groups • built-in user management frameworks for web applications, e.g. AuthKit [30], repoze.who [60] and repoze.what [64], django's built-in and extensible authentication and authorisation [23] Using a third party component can also reduce the size and complexity of the Sage notebook code itself.

Threat	
ID	77
Type	Tampering
Risk Rating	Medium
Description	No anti Cross Site Request Forgery (CSRF) [108] techniques are in use currently. CSRF attacks allow submitting requests on behalf of another user, e.g. publishing worksheets, deleting worksheets, submitting processing requests etc.
Mitigation	Use anti CSRF techniques or libraries, normally involve including a random nonce on each form, and checking for its value on the submitted request. Project is currently in transition between twisted web to pylons. There are 3rd party CSRF middleware for pylons which can be used. Note that any CSRF protection techniques are irrelevant if XSS attacks are available. If the attacker is able to launch an XSS attack, they can relatively easily obtain any randomly generated form values (apart perhaps from Captchas)

Threat	
ID	79
Type	InformationDisclosure
Risk Rating	Low
Description	The administrative user account is called 'admin' and therefore can become a primary target to brute force / dictionary attacks. The admin account can be renamed.
Mitigation	Allow admin user to be renamed

Threat	
ID	80
Type	InformationDisclosure
Risk Rating	Low
Description	When authenticating to the Notebook server, an attacker may be able to identify and enumerate existing accounts on the system. When an existing user account is used to login using an invalid password, the system returns 'Error: Wrong password' response. Whereas, if the user does not exist, the error response is 'Error: Username is not in the system'.
Mitigation	Provide an error response that does not give information which may be abused by an attacker. User returned error messages (not just for the login page), should just provide the minimal necessary information to understand there was a problem. More detailed error information can be logged on the server, and can aid in troubleshooting and investigation. Please note that once a user is registered and attempts to share a worksheet with another user, similar enumeration is possible as well.

Threat	
ID	81
Type	InformationDisclosure
Risk Rating	Low
Description	In certain environments sharing (or publishing) of worksheets may not be a desired feature. For example, some academic environments may use Sage in an examination conditions, where each user must work independently of others. Any sharing of worksheets will compromise the integrity of the examination.
Mitigation	Change all worksheet publishing and sharing functionality as a configurable parameter or option.

4.3.16 Sage Processors [Process 12]

Threat	
ID	37
Type	Spoofing
Risk Rating	Low
Description	Spoofing a processor may be possible, but fairly hard to accomplish (e.g. may involve dns poisoning and or redirecting at the notebook end), and would not achieve much. The cost/benefit ratio of spoofing a processor is very high, making any such threat minimal.
Mitigation	SSH can already be in use to connect the Notebook server with the processors. Using host key authentication should be easy to implement and would mitigate processor spoofing threats.

Threat	
ID	38
Type	Tampering
Risk Rating	High

Description	Each processor is capable (by design) of running arbitrary user-supplied code and therefore tampering is extremely easy to accomplish and impact is likely to be high. Whilst processors are considered to reside on a separate trust boundary, they are in many cases running on the same host as the Sage Notebook server. Any threats to processors in this scenario becomes a threat to the Notebook server as well. Tampering and running malicious code can: • tamper with other processors, modify results or interrupt processing • Denial of service to other processors by exhausting system resources - see threat ID 41 • Launch attacks against the Sage Notebook server (which may reside on the same host) or the processing host itself - see threat ID 42 • Launch attacks on external targets - e.g. spam, distributed Denial of Service, install and run a bot, relay communication, peer to peer traffic etc • eavesdrop on communication to/from the processor host - see threat ID 40

Mitigation	There are several mitigation options to consider. Ideally, several of these options, or even elements of each, should be used to reduce the threat. Mitigation options include: • Consider disallowing some users from running arbitrary code, e.g. by restricting processing capability to as narrow list as possible. Whitelist filtering is preferable to blacklist, but neither is guaranteed to completely mitigate running arbitrary and potentially malicious code. The more restrictive the filtering is, the harder it is to inject code. • Sage processors are considered 'untrusted' and should run on a separate host if possible. Techniques such as virtualisation can be used to reduce cost and simplify running of Sage as a single platform, but physical separation of the processors from the Sage Notebook and other external entities is highly recommended if possible. • Run each processor in its own separate space. Each system process will run with a different userid to another, limiting interaction between processes. Use techniques like chroot, jail shell, sudo -u etc. • Limit the available resources to each processor. Resource constraints should include: cpu time, memory, elapsed running time, disk space usage, number of threads, number of sockets etc. • Use a firewall to restrict network access to/from the processors and other hosts - e.g. Sage Notebook or external entities • Remove or tightly restrict access to filesystem shared between the processors and the Sage Notebook - see threat IDs 70,72

Threat	
ID	39
Type	Repudiation
Risk Rating	Low
Description	System logging on each processor is very minimal if not non-existent. Tracing back to user actions is therefore extremely difficult or impossible.
Mitigation	Implement logging mechanisms tied to identities of users and / or callees of the processors. Consider sending log items to a separate host/entity (e.g. using SYSLOG), to avoid tampering with log records on the filesystem. Any solution which logs user actions should be aware of user privacy concerns and enforce the necessary means to balance between repudiation threats and threats to user privacy.

Threat	
ID	40
Type	InformationDisclosure
Risk Rating	High
Description	Information disclosure can take place if one processor can eavesdrop on communication of other processors, and also if filesystem access is available and depending on filesystem permissions. If filesystem is shared between the processors and the Sage Notebook, then it increases the likelihood of threats to notebook usernames and passwords as well as worksheet data (see threat IDs 34, 45 respectively)
Mitigation	see mitigation for threat ID 38.

Threat	
ID	41
Type	DenialOfService
Risk Rating	High
Description	Denial of service can take place by exhausting system resources, and 'suffocating' other running processors. Denial of service is also more likely to result from simple user error, experimentation or lack of knowledge. If a user is unaware of the processing time for a request, or includes a mistake - it can lead to infinite loops, recursions, large memory allocation, disk space over-use, high processing time etc.
Mitigation	see mitigation for threat ID 38.

Threat	
ID	42
Type	ElevationOfPrivilege
Risk Rating	Low
Description	All processors run using the same privileges. However, a processor can elevate its system privileges (e.g. using an operating system exploit), to gain less restricted access to system resources, as well as higher privileged access to filesystem, memory, processes etc. If root access is obtained, it can lead to a compromise of the entire host running the processors, and if this host is shared with the Sage Notebook it means taking over the entire system.
Mitigation	see mitigation for threat ID 38.

4.4 External Dependencies

ID	Name	Origin
1	Third Party packages (bundled with Sage)	External
2	Linux / Unix / Mac OS	External

4.5 Implementation Assumptions

ID	Element Impacted	Assumption
1	All	Server OS running Sage services is patched and configured securely

4.6 External Security Notes

ID	Note
1	Users should be aware that worksheets received by untrusted third parties, or sent via insecure communication channels may be tampered with and potentially contain malicious code. Worksheets should be inspected before uploaded and used.
2	Sage notebook currently allows worksheets to contain scripts inside HTML. This provides a very easy way to launch Cross Site Scripting (XSS)attacks, undermining any authentication and authorisation mechanisms in place.
3	Unless the Sage processors are isolated / separated from the Sage Notebook, any threats and vulnerabilities applicable to the Sage Processors will in all likelihood also apply to the Sage Notebook and its data store. Therefore, security of worksheet data, user account information or any information residing on the same platform as the processors is easily accessible to attackers.
4	When deploying Sage, deployers/users should be aware that threats to Sage are not confined only to Sage itself, but can also affect other external or internal systems. Attacks on the Sage platform may allow launching further attacks on other system across the internal and external networks. Until such threats are mitigated or contained within Sage, deployers should use 3rd party controls when possible. Controls such as firewalls, intrusion detection or prevention systems and host/network monitoring tools are recommended.

Chapter 5

Sage Threat Model Analyses

5.1 Overview

The threat model on chapter 3, covering the Sage Open Source Development process, documented 17 threats, 15 of which are rated Low and only 2 rated Medium. In contrast, the Sage Notebook threat model on chapter 4 raised 35 threats: 11 rated High, 9 Medium and 15 Low. It is difficult to compare one threat model with another, particularly when attempting to count or rate threats. Any such quantitative comparison runs the danger of giving a (potentially misleading) qualitative rating or measurement. The threat models were deliberately separate from each other to avoid such situation. Even though both threat models cover Sage, they cover completely independent aspects of the application. The methodology, process and rating system for threats is the same across both threat models, but a Low threat on one threat model is not equal to a Low threat on the other. In addition, threats may be 'linked', if they relate to the same component, vulnerability and most importantly, mitigation option(s). If one control mitigates more than one threat, the *number* of threats is almost irrelevant from a practical perspective. With that considered, it is still apparent that the Sage Notebook threat model highlights more significant threats and vulnerabilities than those to the development process. The two threat models are further explored on the following sections.

5.2 Development Process Threat Model Analysis

As reflected on the threat model, the Sage development process is robust and thorough. The two Medium rated threats identified are both tampering threats. One (Threat ID 110), raises a concern over third party software packages, which may be tampered with to inject malicious code or vulnerabilities. The other (Threat ID 36), identifies potential threats to the ticketing system, potentially

allowing bypass of some of the peer review processes. The latter is partially mitigated, and arguably can be 'downgraded' to Low, and the former can be relatively easily mitigated using more robust processes and ensuring the integrity of 3rd party packages from the source. It is interesting to note that the author of [33] raised a very similar concern after (and in all likelihood without knowledge of) threat ID 110 was first identified and a draft threat model shared with the Sage project [2]. Most other Low rated threats are to a large extent at least partially mitigated already, are at an acceptable level of risk, or can be resolved with comparatively easy means (most of which are process rather than technology driven).

This paper's view is that there is little concern over the development process, even though some enhancements are recommended. It is felt that other open source or commercial projects could 'learn' from the Sage development process. By and large, the Sage project takes advantage of the 'many eyeballs' principle [78] in their development code review process. In addition, as mentioned previously, the Sage development process is not much different from other processes used within the open source community or business environments. It is therefore hoped that other projects can learn from the Sage experience, use the threat model presented on this paper, adapt it for their own processes or needs, or create a completely new threat model. Such approach could potentially further enhance the security of other development processes.

Using a threat model in this case demonstrated the ability to identify both process and technology related security issues. The threat model assisted answering questions such as "What assurances or confidence can we get about the protection from a malicious party injecting insecure code or backdoor into our product?", "How can we be sure insecure code gets inspected and rejected before it goes into the codebase?", "What might be the 'weakest link' in the process?", "Are these issues technology related or process driven?". Furthermore, the threat modelling process not only provided answers, but also some level of assurance. As long as the model itself is accurate and comprehensive, it ensures a methodical walkthrough of each and every entity with respect to potential security threats. Such 'coverage' may not be possible by simply 'brainstorming' ideas or trying to answer these questions directly.

5.3 Sage Notebook Threat Model Analysis

The Sage Notebook provides an interesting combination of security related problems. On one hand, it is not much different from other web based applications, and as such, some of the threats are to a certain degree similar to those expected of many other web based applications. Threats to data integrity and confidentiality are not unique to Sage, and for that reason, mitigation options are widely available and relatively easy to install and use. A fairly substantial number of Medium and High rated threats (Threat IDs 53, 54, 47, 57, 31, 50, 51 and others) can be very easily mitigated with the usage of 'standard' information security tools. Using SSL, which is already enabled on the Sage Notebook - but not set by default, can fully mitigate the majority of these aforementioned threats. Similarly, albeit in this case at a more substantial mitigation effort, using 3rd party authentication and authorisation packages can support more

robust separation of privileges and protect against a number of different threats (Threat IDs 73, 34. 36, 45, 40). On the other hand, Sage provides unique functionality not commonly available in other applications, and much less so web based ones. This unique functionality and user flexibility comes at a rather substantial 'price tag' from a security perspective. The Sage Notebook threat model identified a number of threats that boil down to the core architecture as well as the range of possibilities available to users. Those type of threats are much more difficult to mitigate and address, but there are still a number of possibilities to explore. The following sections will explore those unique features from a security standpoint, and cover a number of directions for mitigation in more depth.

From a practical point of view, it is advised that the Sage project starts addressing the more generic, and easy to mitigate threats first. These 'low hanging fruit' present very cost effective way to reduce the threats to the Sage Notebook with relatively minimal effort and cost. In fact, some of these efforts are already underway. This project is actively engaged trying to assist the Sage project in getting and installing a signed SSL wildcard certificate for all sagenb.org domains. Obtaining such certificate would allow the Sage Notebook public server to run over HTTPS without users getting unnecessary browser alerts (for 'untrusted' certificates). All communication between users and the Sage Notebook servers will be encrypted and integrity protected, the Sage Notebook server will be authenticated by users, and a number of threats fully mitigated. At the point of writing, it is not clear whether the certificate will be generated and implemented before the submission due date for this project paper. This is a result of limited availability of key people within the Sage project, whose involvement is necessary for implementation.

5.4 Untrusted Code

One of the key challenges and perhaps the most substantial threat to Sage stems from the nature and architecture of the application. Sage allows users to not only run calculations and formulas, but to directly execute code on the server, using various programming languages and back-end processing engines. Such powerful feature comes at a great cost from security perspective. Allowing users to run any code on the server means that users are also capable of running malicious code. Malicious code can not only place the Sage users, information and platform at risk, but can also threaten external systems and resources. Sage deployments can therefore become a 'launching ground' for attackers, botnet operators and other malicious parties. Such scenario puts the Sage project and server operators at an additional, indirect risk. They may be held responsible (and perhaps even legally liable [49]) for unauthorised or illegal actions launched from their server(s).

Therefore, one of the principal approaches of mitigating such risk is trying to contain and isolate processes running untrusted user-supplied code from other processes, systems and resources. Additional mitigation options which can be used in unison with any containment and isolation techniques include filtering for untrusted instructions, limiting the availability of certain 'insecure' functions, as well as using the least privilege concepts and only allowing a subset of

users such access.

As unique and as flexible as the Sage architecture is, and as complex it may be to solve its security challenges, it appears to share some of the security problems that are faced by service providers of one of the increasingly popular areas in Information Technology, namely: Cloud Computing and Software as a Service. Those problems however, are far from new, and are at the core design principles for secure systems: e.g. separation of privilege, least privilege [81], preventing access to the layer below [34] and compartmentalization [61].

"Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS). The datacenter hardware and software is what we will call a Cloud." [3]

Software as a Service (SaaS), Service Oriented Computing (SOC), Service Oriented Architecture (SOA) [4] [73] and Cloud/Grid/Utility Computing usually run different user applications or services on shared data centre resources in order to ensure economies of scale and lower cost of ownership, utilising virtualisation technologies [107] [8].

This project paper does not attempt to go into much detail covering the different types of risks associated with cloud computing and service oriented architectures. However, one area of commonality with Sage which is going to be explored further, is the usage of shared resources by different users. In cloud computing, if a user introduces malicious code or a misbehaving application - it can potentially affect other applications and user data on the shared resource. This scenario is similar to the Sage architecture, where users run computations and code directly on the Sage server.

The following sections attempt to give a brief overview of some of the available process isolation and containment methods available today, as they apply to shared resource environments. Such environments range from modern operating systems to virtualised platforms and cloud computing resources. Each method will be assessed based on the Sage requirements, perceived benefits or drawbacks and ease or difficulty of implementation.

5.4.1 Operating System Protection

Most modern operating systems provide at least a basic level of security for processes. User and System level processes are executed using different privilege levels, and each process memory space, registers and process control block data is protected by the operating system based on its protection domain [97]. However, user processes are still able to perform a variety of operations on a typical Unix/Linux platform, including binding to sockets, spawning threads and child-processes, consuming CPU, memory and disk space, accessing the filesystem etc. Such access can be used to launch attacks on other processes or systems or used locally in order to elevate privileges (e.g. via exploiting of a buffer overflow attack on a vulnerable library or local process). Therefore, these operations can and should be limited by using standard operating system facilities where possible. Sage should utilise as many of the following process

constraining, isolation and protection techniques:

- Each Sage process should run using a different Unix UID/GID to utilise the operating system built-in protection domains.
- Ensure file system permissions are set to the minimal required to allow a process to run. Typically read/write access will only be enabled to a per-process storage area with the sticky bit set. Setting the sticky bit may also help protecting against race conditions [5].
- Run processes in a chroot environment [75]. However note chroot environments consume more disk space, require configuration, and can still be 'escaped' from by some attacks, particularly if not configured securely [15].
- Enforce resource limits per user / process - using mechanisms such as ulimit. (The authors of [20] suggest ulimit, as well as additional operating system security enhancements for securing untrusted mobile code; Some of their suggestions are reflected on other items on this list).
- Use logging and monitoring mechanisms to track misbehaving processes and suspicious error messages.

5.4.2 Security Add-ons

The controls and options mentioned on the previous section demonstrate a number of options to reinforce the built-in (Unix based) operating system security. However, some areas are not natively supported by the operating system, and can be augmented by adding enhanced security mechanisms to further restrict processes running on the system. This section touches briefly on some of the leading tools in this area.

- **SELinux** - Security-Enhanced Linux [65], provides a flexible policy to allow an extensive set of security controls, including (but not limited to): confining the potential damage caused through an exploit, protecting privileged process from executing malicious code and preventing user processes from interfering with system processes [89]. SELinux provides a powerful set of security mechanisms. However, it is considered difficult to configure and fine-tune even for security experts [57].
- **AppArmor** [68] - is similar to SELinux, but is considered less comprehensive yet much simpler and easier to maintain and manage [77]. AppArmor seems relatively suitable to an application like Sage, since it is application and process centric. However, a system-wide protection is preferable from a security perspective. AppArmor is an open source project, and is still bundled with a number of Linux distributions, but support from Novell has been dropped in 2007, which puts the future of the project in question [87].
- **LIDS** - Linux Intrusion Detection System [112] enhance the basic Linux security by introducing mandatory access control and system-level policies. LIDS support a concept referred to as Trusted Domain Enforcement

(TDE). TDE aids in application sandboxing, a more powerful version of chroot [109]. LIDS uses an Access Control List (ACL) system similar to IPTables [67]. It allows restricting access in several ways: file system protection, devices, important system processes, network restrictions (including sniffing protection, socket binding restrictions and port scanning detection) and others [111].

- **RSBAC** - Rule Set Based Access Control [72] is a similar framework to SELinux (both in terms of capabilities and complexity of implementation and management). RSBAC is based on the Generalized Framework for Access Control (GFAC) [1]. Out of the box, RSBAC does not come with a pre-defined policy and may therefore be difficult to set up initially [47]. In terms of flexibility, stability and extensibility, it appears to provide a good alternative to SELinux and LIDS to consider.
- **grsecurity** [91] offers configuration-free operation and protection of address space attacks such as buffer overflow and fork bombs. grsecurity also uses an ACL system which includes IP and TCP based restrictions as well as robust auditing and logging system. It is considered simpler to install and use than SELinux [24] and also provides chroot hardening.

If used appropriately, any one of these add-ons should increase the Sage process compartmentalization and isolation and could prevent attacks on other processes, hosts and networks. It is difficult to recommend any particular add-on from the above list as the most suitable one for Sage. The decision should be made on the balance of ease of installation, use, maintenance and simplicity of inclusion as a Sage package or as an add-on option.

5.4.3 Network Isolation

Network isolation is an important element to protect Sage against attack propagation and reduce exposure of internal and external systems. If a malicious code is running on Sage, it may attempt to attack external systems, establish a covert channel to its operator, or send and retrieve information. Blocking external network access and isolating the process from other network resources can mitigate such threats. Any network filtering, firewall or intrusion detection and prevention techniques can be used. IPTables [67] is probably the simplest and easiest tool to implement as a first line of defence. Network isolation can also assist in creation of a barrier to enforce the trust boundary between the Sage Notebook and the Sage Processors. More elaborate network restrictions can be implemented by using security add-on techniques mentioned in section 5.4.2. It is important to understand that network isolation by itself does not provide much protection for localised attacks and attacks which utilise existing resources (e.g. filesystem, memory) as well as attacks over existing authorised communication channels. Network isolation, as its name suggests, assists in containment and isolation of processes and therefore aid in *limiting* attack propagation.

5.4.4 Virtualisation

The use of virtualisation in the context of grid and cloud computing was mentioned briefly on Section 5.4. Virtualisation is not a new technology, and was first used in the 1960s. It however offers a number of benefits and seems to become more and more popular, with numerous commercial and research projects available on commodity hardware. Amongst the many benefits, the ability to provide separation of services can improve security and portability [12]. Virtualisation can take several forms, from virtualisation of the filesystem (as the case in chroot, mentioned in section 5.4.1, or a more elaborate copy-on-write filesystem called Solitude [45]), via operating system virtualisation using containers [90], through to hardware abstraction and emulation techniques using hypervisors [80] and hardware virtualisation support, such as Intel VT and AMD-V [21]. Whilst virtualisation vendors and research papers appear to mention the potential for increased security [51] [21], security and isolation in a virtualised environment isn't guaranteed. As several research papers suggest, security of the virtualisation platforms can be compromised and improved further [71] [32] [79].

Nevertheless, in the context of Sage, even if virtualisation does not guarantee the 'ultimate' solution from a security perspective, it can serve a number of purposes by improving process isolation and for running untrusted code:

- De-coupling of the Sage Notebook and processors can be easily achieved using virtualisation. One of the core architectural issues covered in the threat model, was the lack of separation between the Notebook and the processors, which are placed on separate trust areas. Separation of the two is therefore key to improving security. The Sage project is likely to prefer to 'bundle' both the notebook and the backend processors into one distributable package. Bundling a virtual machine, or packaging Sage into virtual machine components can assist in creating better separation between the two components.

- Virtualisation could provide potentially simpler to implement isolation than some of the techniques mentioned on previous sections. Simplicity and ease of implementation can translate to better security in practice if the choice is between (theoretically insecure) virtualisation, and not implementing any isolation techniques at all. Security of virtualisation software is also likely to increase and become a concern of many. Sage is not the only software to benefit from improved security in virtualisation platforms.

- Performance can also be better controlled through virtualisation. Controlling performance of misbehaving processes can assist in prevention of denial of service, and provide more stability. In addition, multiple virtual machines can be used on multiple hardware platforms for load balancing and added redundancy.

- File system isolation may be easier to implement using virtualisation. chroot techniques are effective, but perhaps not as easy to maintain and not necessarily more secure.

- Processor virtual machines can be 'disposable'. Since no persistent data is normally stored on the processing engines, virtual machines can be restarted to a 'clean' state. Such an approach can also help against rootkits and malware which may have penetrated. The more frequent the disposal process, the smaller the window of attack.
- Portability can improve across different hardware platform. This is not a pure security advantage. However, having Sage optimised for one platform ensures the code is less complex, which aids in auditing and proving security.

As mentioned on section 4.1, the sagenb.org server is already running on a virtual server and therefore enjoys some of the benefits virtualisation can offer. It's important to remember that the sagenb.org server runs both the notebook server and processing engines on the same virtual machine. Virtualisation technologies are therefore recommended to be used further with Sage and particularly to decouple the notebook from the processor engine(s). Usage of virtualisation should augment the techniques mentioned on previous sections, not replace them.

Chapter 6

Conclusion

The Sage open source project presents several interesting information security challenges. Some of the identified security threats and vulnerabilities can be relatively easily mitigated, whilst others may require more substantial investment in time and resources.

When considering Sage, it is difficult to come to a clear conclusion whether the open source nature of the project contributed to its security, or detracted from it - by the fact that the source code and other information is readily available.

It appears that very few of the threats were discovered purely by looking at the source code of the application. The majority of the threats would still be relatively easy to identify simply by using the application, even with no access to code. In that respect, publishing the source code did not appear to put the application at much additional risk. However, leaving this project paper aside[1], the open source nature of Sage, and the 'many eyeballs' principle did not seem to provide much insight and contribution from an information security perspective. The Sage project is comprised of many highly dedicated, multi-faceted and very talented individuals. However, it appears that most of those members of the community aren't particularly concerned with security. The open and friendly nature of the project is to a large extent opposed to putting barriers and mounting protection. It promotes collaboration and trust instead.

It is not anticipated that the Sage project would implement all of the recommendations on this paper, perhaps only a small part of them. However, it is hoped that the paper can serve as a basis for further discussion and future work. Some improvements have already been initiated, and others are in discussion or planned for the future. A recent online discussion [11] has begun, suggesting a rewrite of the notebook and de-coupling it from the processing engine. This discussion was generated by developers from a 'spin-off' project, similar to Sage Notebook, called codenode [99], providing users with web access to Python and Sage.

[1] although some can claim this paper does in fact prove the opposite, and it cannot therefore be discounted.

The work presented on this paper would hopefully assist not only Sage but also codenode. Other open and closed source projects can perhaps compare their development process to the Sage one, and use similar threat modelling methodologies to analyse threats to their code review and approval process as well as their application itself. Some of the coverage of process isolation and protection techniques might give ideas to other application developers and system owners, particularly when dealing with untrusted code.

Finally, contribution to open source projects can take many forms. Humbled by the talent and dedication of so many volunteers and collaborators and their achievements, it is hoped that this paper provided a welcome contribution to at least one such project.

Bibliography

[1] M.D. Abrams, KW Eggers, L. LaPadula, and I. Olson. A generalized framework for access control: An informal description. In *Proceedings of the 13th National Computer Security Conference*, pages 135–143, 1990.

[2] Y. Aner. Sage Development Process Threat Model. Available at `http://groups.google.com/group/sage-devel/msg/1f851e27f5500712`. Last accessed, 18 August 2009.

[3] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A Berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009. Available at `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf`. Last accessed, 28 August 2009.

[4] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-based software: the future for flexible software. In *Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific*, pages 214–221, 2000.

[5] M. Bishop and M. Dilger. Checking for race conditions in file accesses. *Computing systems*, 2(2):131–152, 1996.

[6] B. Boehm and V.R. Basili. Software defect reduction top 10 list. *IEEE Computer*, 34(1):135–137, 2001.

[7] S R Bourne. Unix time-sharing system: The unix shell. *Bell System Technical Journal*, (57):1971–1990, 1978.

[8] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[9] C. Castelluccia, E. Mykletun, and G. Tsudik. Improving secure server performance by re-balancing SSL/TLS handshakes. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 26–34. ACM New York, NY, USA, 2006.

[10] CERT Coordination Center, Software Engineering Institute,Carnegie Mellon University. malicious HTML tags embedded in client web requests.

Technical report, CA-2000-02, 2000. Available at http://www.cert.org/advisories/CA-2000-02.html. Last accessed, 28 August 2009.

[11] O. Certik. notebook rewrite. Available at http://groups.google.com/group/sage-devel/browse_thread/thread/65ca1e0489a0a980/c3abd60f1e13a3a3. Last accessed, 24 August 2009.

[12] P.M. Chen and B.D. Noble. When virtual is better than real. In *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS)*, pages 133–138, 2001.

[13] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, pages 76–79, 2004.

[14] 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.

[15] A. Chuvakin. Using Chroot Securely. Available at http://www.linuxsecurity.com/content/view/117632/49/, 2007. Last accessed, 21 August 2009.

[16] Coverity. Coverity Prevent - Static Analysis. Available at http://www.coverity.com/products/coverity-prevent.html. Last accessed, 27 July 2009.

[17] 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.

[18] D. Evans and D. Larochelle. Improving security using extensible lightweight static analysis. *IEEE software*, pages 42–51, 2002.

[19] D.C. Feldmeier and P.R. Karn. UNIX Password Security-Ten Years Later. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 44–63. Springer-Verlag London, UK, 1989.

[20] V. Felmetsger and G. Vigna. Exploiting OS-level mechanisms to implement mobile code security. In *10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings*, pages 234–243, 2005.

[21] J. Fisher-Ogden. Hardware support for efficient virtualization. Available at http://cseweb.ucsd.edu/~jfisherogden/hardwareVirt.pdf, 2006. Last accessed, 25 August 2009.

[22] B. Foote and J. Yoder. Big ball of mud. In *Pattern Languages of Program Design*, pages 653–692. Addison-Wesley, 1999.

[23] Django Software Foundation. User authentication in Django. Available at http://docs.djangoproject.com/en/dev/topics/auth/. Last accessed, 25 July 2009.

[24] M. Fox, J. Giordano, L. Stotler, and A. Thomas. Selinux and grsecurity: A case study comparing linux security kernel enhancements. Available at `http://www.cs.virginia.edu/~jcg8f/GrsecuritySELinuxCaseStudy.pdf`, 2003. Last accessed, 27 August 2009.

[25] Free Software Foundation. The free software definition. Available at `http://www.fsf.org/licensing/essays/free-sw.html`. Last accessed, 14 July 2009.

[26] Free Software Foundation. GNU General Public License. Available at `http://www.gnu.org/licenses/gpl.html`. Last accessed, 28 July 2009.

[27] Free Software Foundation. The GNU Privacy Guard. Available at `http://www.gnupg.org/`. Last accessed, 28 July 2009.

[28] Free Software Foundation. Selling Free Software. Available at `http://www.gnu.org/philosophy/selling.html`. Last accessed, 28 July 2009.

[29] The GAP Group. GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra. Available at `http://www.gap-system.org/`. Last accessed, 28 July 2009.

[30] J. Gardner. Authkit - WSGI Authentication and Authorization Tools. Available at `http://authkit.org/`. Last accessed, 25 July 2009.

[31] J.J. Garrett et al. Ajax: A new approach to web applications. *Adaptive path*, February 18, 2005. Available at `http://www.adaptivepath.com/publications/essays/archives/000385.php`. Last accessed, 27 August 2009.

[32] C. Gebhardt and A. Tomlinson. Security consideration for virtualization. *Technical Report RHUL–MA–2008–16, Department of Mathematics, Royal Holloway, University of London*, 2008. Available at `http://www.ma.rhul.ac.uk/static/techrep/2008/RHUL-MA-2008-16.pdf`. Last accessed, 28 August 2009.

[33] A. Ghitza. checklist for reviewing an spkg? Available at `http://groups.google.com/group/sage-devel/browse_thread/thread/030ff5c32e632936#`. Last accessed, 18 August 2009.

[34] D. Gollmann. *Computer security*. John Wiley & Sons, second edition, 2005.

[35] G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR A computer algebra system for polynomial computations. Available at `http://www.singular.uni-kl.de`. Last accessed, 28 July 2009.

[36] R. Harrison. Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms. RFC 4513, Internet Engineering Task Force, June 2006. Available at `http://tools.ietf.org/html/rfc4513`. Last accessed, 28 August 2009.

[37] J.H. Hoepman and B. Jacobs. Increased security through open source. *COMMUNICATIONS–ACM*, 50:79–84, 2007.

[38] M. Howard and D.E. Leblanc. *Writing secure code*. Microsoft Press, second edition, 2002.

[39] M. Howard and S. Lipner. *The Security Development Lifecycle*. Microsoft Press, 2006.

[40] T. Howlett. *Open Source Security Tools: Pratical Guide to Security Applications, A*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2004.

[41] Y.W. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, and S.Y. Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*, pages 40–52. ACM New York, NY, USA, 2004.

[42] IBM Internet Security Systems. X-Force®2008 Midyear Trend Statistics. Available at `http://www-935.ibm.com/services/us/iss/xforce/midyearreport/xforce-midyear-report-2008.pdf`. Last accessed, 17 August 2009.

[43] Open Source Initiative. Available at `http://www.opensource.org`. Last accessed, 14 July 2009.

[44] ISO/IEC. ISO/IEC 27005:2008 Information technology - Security techniques - Information security risk management. First edition, International Organization for Standardization, Geneva, Switzerland., 2008.

[45] S. Jain, F. Shafique, V. Djeric, and A. Goel. Application-level isolation and recovery with solitude. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 95–107. ACM New York, NY, USA, 2008.

[46] M. Jakobsson. Modeling and Preventing Phishing Attacks. In *Financial cryptography and data security: 9th international conference, FC 2005, Roseau, The Commonwealth of Dominica, February 28-March 3, 2005: revised papers*, page 89. Springer Verlag, 2005.

[47] M. Jawurek. RSBAC–a framework for enhanced Linux system security. In *Dependable Distributed Systems*, Laboratory of dependable distributed systems, RWTH Aachen University, 2006. Available at `http://rsbac.org/doc/media/rsbac-marek2006.pdf`. Last accessed, 28 August 2009.

[48] A. Joshi, S.T. King, G.W. Dunlap, and P.M. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 91–104. ACM New York, NY, USA, 2005.

[49] E. Kenneally. Stepping on the digital scale–Duty and Liability for Negligent Internet Security. *;login: The Magazine of USENIX & SAGE*, 26(8):62–77, 2001.

[50] 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.

[51] N. Kiyanclar. A survey of virtualization techniques focusing on secure on-demand cluster computing. *Arxiv preprint cs/0511010*, 2005. Available at `http://arxiv.org/pdf/cs/0511010`. Last Accessed, 30 August 2009.

[52] Klocwork. Klockwork Insight. Available at `http://www.klocwork.com/products/insight.asp`. Last accessed, 27 July 2009.

[53] J. Kohl and C. Neuman. The Kerberos network authentication service (v5). RFC 1510, Internet Engineering Task Force, 1993. Available at `http://tools.ietf.org/html/rfc1510`. Last accessed, 28 August 2009.

[54] Vassiliki Koutsonikola and Athena Vakali. LDAP: Framework, Practices, and Trends. *IEEE Internet Computing*, 8(5):66–72, 2004.

[55] K.A. Kozar. Representing systems with data flow diagrams. Available at `http://spot.colorado.edu/~kozar/DFD.html`, 1997. Last accessed, 27 August 2009.

[56] P.G. Larsen, N. Plat, and H. Toetenel. A formal semantics of data flow diagrams. *Formal aspects of Computing*, 6(6):586–606, 1994.

[57] N. Li, Z. Mao, and H. Chen. Usable mandatory integrity protection for operating systems. In *IEEE Symposium on Security and Privacy, 2007. SP'07*, pages 164–178, 2007.

[58] Mailman, the GNU Mailing List Manager. Available at `http://www.gnu.org/software/mailman/index.html`. Last accessed, 18 July 2009.

[59] A. Martelli and D. Ascher. *Python cookbook*. O'Reilly Media, Inc., 2005.

[60] C. McDonough. `repoze.who` - wsgi authentication middleware. Available at `http://docs.repoze.org/who/`. Last accessed, 25 July 2009.

[61] G. McGraw and Viega J. Software security principles: Part 3. Available at `http://www.ibm.com/developerworks/library/s-priv.html`, 2000. Last accessed, 21 August 2009.

[62] Gary McGraw. Software security. *IEEE Security & Privacy*, March/April 2004.

[63] Microsoft. Windows Server 2008 Active Directory. Available at `http://www.microsoft.com/windowsserver2008/en/us/active-directory.aspx`. Last accessed, 5 August 2009.

[64] G. Narea. `repoze.what` - authorization for wsgi applications. Available at `http://what.repoze.org/docs/1.x/`. Last accessed, 25 July 2009.

[65] National Security Agency. Security-Enhanced Linux. Available at `http://www.nsa.gov/research/selinux/index.shtml`. Last accessed, 21 August 2009.

[66] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *COMMUNICATIONS–ACM*, 21(12):993–999, 1978.

[67] Netfilter.org. The netfilter.org iptables project. Available at `http://www.netfilter.org/projects/iptables/index.html`. Last accessed, 28 July 2009.

[68] Novell. Project AppArmor. Available at `http://forge.novell.com/modules/xfmod/project/?apparmor`. Last accessed, 21 August 2009.

[69] OpenBSD. OpenSSH. Available at `http://www.openssh.com/`. Last accessed, 28 July 2009.

[70] OpenLDAP Foundation. OpenLDAP - community developed LDAP Software. Available at `http://www.openldap.org/`. Last accessed, 5 August 2009.

[71] T. Ormandy. An empirical study into the security exposure to host of hostile virtualized environments. In *CanSecWest 2007: Applied Security Conference*, 2007.

[72] A. Ott. RSBAC – Rule Set Based Access Control. Available at `http://www.rsbac.org/`. Last accessed, 22 August 2009.

[73] M.P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12. NW Washington: IEEE Computer Society, 2003.

[74] LD Paulson. Building rich web applications with Ajax. *Computer*, 38(10):14–17, 2005.

[75] V. Prevelakis and D. Spinellis. Sandboxing applications. In *Proceedings of the USENIX Technical Annual Conference, Freenix Track*, pages 119–126, 2001.

[76] Python Software Foundation. pickle - Python Object Serialization. Available at `http://docs.python.org/library/pickle.html`. Last accessed, 28 July 2009.

[77] N.A. Quynh, R. Ando, and Y. Takefuji. Centralized security policy support for virtual machine. In *LISA '06: Proceedings of the 20th conference on Large Installation System Administration*, pages 79–87, Berkeley, CA, USA, 2006. USENIX Association.

[78] E. Raymond. The cathedral and the bazaar. *Knowledge, Technology, and Policy*, 12(3):23–49, 1999.

[79] J.S. Reuben. A Survey on Virtual Machine Security. *Helsinki University of Technology*, 2007. Available at `http://www.tml.tkk.fi/Publications/C/25/papers/Reuben_final.pdf`. Last accessed, 30 August 2009.

[80] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. Van Doorn, J.L. Griffin, and S. Berger. sHype: Secure hypervisor approach to trusted virtualized systems. *IBM Research Report RC23511*, 2005. Available at `http://domino.watson.ibm.com/library/cyberdig.nsf/papers/265c8e3a6f95ca8d85256fa1005cbf0f/$file/rc23511.pdf`. Last accessed, 30 August 2009.

[81] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[82] J. Scambray, M. Shema, and C. Sima. *Hacking Exposed Web Applications*. McGraw-Hill Osborne Media, second edition, 2006.

[83] Bruce Schneier. Open source and security. *Crypto-Gram. Counterpane Internet Security, Inc.*, September 15, 1999. Available at `http://www.counterpane.com/crypto-gram-9909.html`. Last accessed, 25 July 2009.

[84] D. Scott and R. Sharp. Specifying and enforcing application-level web security policies. *IEEE Transactions on Knowledge and data Engineering*, pages 771–783, 2003.

[85] Michael Scovetta. YASCA - Yet Another Source Code Analyzer. Available at `http://sourceforge.net/projects/yasca/`. Last accessed, 27 July 2009.

[86] Secure Software Inc. RATS - Rough Auditing Tool for Security. Available at `http://www.fortify.com/security-resources/rats.jsp`. Last accessed, 27 July 2009.

[87] S. Shankland. Novell lays off AppArmor programmers. Available at `http://news.cnet.com/8301-13580_3-9796140-39.html?part=rss&subj=news&tag=2547-1_3-0-5`, 2007. Last accessed, 21 August 2009.

[88] 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.

[89] S. Smalley and T. Fraser. A Security Policy Configuration for the Security-Enhanced Linux. 2001. Available at `http://www.artware.qc.ca/~fil/banned/selinux/policy-200012181053.pdf`. Last accessed, 28 August 2009.

[90] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 275–287. ACM New York, NY, USA, 2007.

[91] B. Spengler. grsecurity. Available at `http://www.grsecurity.net/index.php`. Last accessed, 22 August 2009.

[92] W. A. Stein et al. *Sage Components*. The Sage Development Team. Available at `http://www.sagemath.org/links-components.html`. Last accessed, 28 July 2009.

[93] W. A. Stein et al. *Sage Notebook Public Server*. The Sage Development Team. Available at `http://www.sagenb.org/`. Last accessed, 28 July 2009.

[94] W. A. Stein et al. *Sage Mathematics Software (Version 4.1.1)*. The Sage Development Team, 2009. `http://www.sagemath.org`. Last accessed, 25 August 2009.

[95] Z. Su and G. Wassermann. The essence of command injection attacks in web applications. In *Annual Symposium on Principles of Programming Languages*, pages 372–382. ACM New York, NY, USA, 2006.

[96] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press, 2004.

[97] A.S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, third edition, 2007.

[98] G. Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology RTI Project*, 2002. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.3316&rep=rep1&type=pdf`. Last accessed, 30 August 2009.

[99] The codenode group. codenode. Available at `http://codenode.org/`. Last accessed, 24 August 2009.

[100] G. Tornaria. Question about notebook server setup in a VM. Available at `http://groups.google.com/group/sage-devel/browse_thread/thread/3927795c8f1c8a8f/2f21594bd6486d6`. Last accessed, 29 August 2009.

[101] P. Torr. Demystifying the threat-modeling process. *IEEE Security & Privacy*, pages 66–70, 2005.

[102] TrueCrypt Foundation. Truecrypt - Free open-source disk encryption software for Windows Vista/XP, Mac OS X, and Linux. Available at `http://www.truecrypt.org/`. Last accessed, 28 July 2009.

[103] K. Tsipenyuk, B. Chess, and G. McGraw. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security & Privacy*, 3(6):81–84, 2005.

[104] G. van Rossum and T. Peters. PEP-307 Extensions to the pickle protocol. Available at `http://www.python.org/dev/peps/pep-0307/`. Last accessed, 28 July 2009.

[105] J. Viega. The myth of open source security. Available at `http://www.developer.com/tech/article.php/10923_626641_1`, 2000. Last accessed, 18 July 2009.

[106] J. Viega. *The Myths of Security: What the Computer Security Industry Doesn't Want You to Know*. O'Reilly Media, Inc, 2009.

[107] M.A. Vouk. Cloud computing–Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40, 2008.

[108] P. Watkins. Cross Site Request Forgeries (CSRF). *BugTraq posting*, 2001. Available at `http://www.tux.org/~peterw/csrf.txt`. Last accessed, 27 August 2009.

[109] Y. Wilajati Purna. LIDS Trusted Domain Enforcement (TDE): An Introduction. Available at `http://www.lids.org/document/LIDS-TDE-feature.txt`, 2004. Last accessed, 21 August 2009.

[110] B. Witten, C. Landwehr, and M. Caloyannides. Does open source improve system security? *IEEE SOFTWARE*, pages 57–61, 2001.

[111] H. XIE. LIDS Hacking HOWTO. Available at `http://www.lids.org/lids-howto/lids-hacking-howto.html`, 2000. Last accessed, 21 August 2009.

[112] H. XIE, P. Biondi, Y. Wilajati Purna, S. Klein, and K. Omo. LIDS – Linux Intrusion Detection System. Available at `http://www.lids.org/`. Last accessed, 21 August 2009.

[113] T. Ylonen. The Secure Shell (SSH) Authentication Protocol. RFC 4252, Internet Engineering Task Force, January 2006. Available at `http://tools.ietf.org/html/rfc4252`. Last accessed, 28 August 2009.