Recx

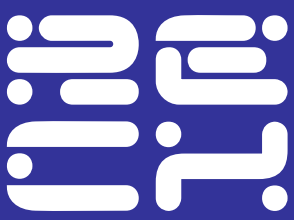
Whitepaper

Software Security Austerity - Software security debt in modern software development

RECX/2012/RELEASE/01

5th March 2012

Release 1.0



COPYRIGHT 2009-2012 RECX LTD.

1 Abstract

The concept of technical debt is not a new one. Technical debt has historically referred to the trade-off between getting a solution or system to market versus a perfectively designed and a bug free product. In the process of trading perfection for an economically viable development model the software incurs a degree of debt.

The debt analogy is starting to be applied to systems and software security. Recx have previously discussed some of the trade-offs made with regard to software security and time-to-market in our article entitled: *Breaking the Inevitable Niche/Vertical Technology Security Vulnerability Lifecycle* [1].

It is important to recognise that a Secure Development Life Cycle (SDLC) does not stop the presence or accumulation of security debt. A maturing SDLC allows an organisation to identify weaknesses and thus convert a larger volume of previously unknown debt to known security debt. The security debt once known, then needs robust processes to both service and repay it over a period of time. Typically SDLCs only set the criteria for the issues that must be fixed over a certain impact level. As organisations get better and more efficient at identifying security issues they typically start to accrue a substantial number of lower rated issues in the process. While these issues may on their own have less of an impact, when several lower issues are combined they can be as impactful as higher rated issues. As a result, the accumulation of a large number of lower impact issues without any strategy on how to resolve them can be equally as risky to security.

In this white-paper, Recx first introduces the reader to the concept and risk of software security debt. A review is then performed of the types and sources of debt before discussing how it can build up when using a risk assessment based approach to prioritisation. A number of debt management strategies are then presented along with associated events, such as servicing, repayment, overhang and expiry. Finally a number of conclusions are drawn around software security debt and why it needs to be considered as part of mature secure software development and risk management processes.

1.1 Intended Audience

This white-paper is intended for software and security professionals who are unfamiliar with the concept of security debt or are interested in strategies for the identification and management, and how it relates to risk.

[1] http://recxltd.blogspot.com/2011/12/breaking-inevitable-nichevertical.html

2 Introduction

The concept of technical debt within software was introduced in 1992 by Ward Cunningham[1]; summarised[2] as:

> "*Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.*"

In February 2011 Chris Wysopal applied the technical debt metaphor to software security[3] whilst attempting to develop a financial model[4] within which to quantify the cost of a breach. These initial discussions were interesting but not fully developed, as acknowledged by Wysopal, due in part to a primary focus on latent security issues. Subsequent to these posts, others, such as Alfonso Barreiro (February 2012), also started talking about security debt in software[5].

Although broadly similar to technical debt, security debt is typically more subtle. Due to its nature an organisation can have a significant risk exposure to security debt without any immediate effects. Although both types of debt affect the users of the software or system, it's more likely that technical debt will be identified and repaid first. Users will report faults, or the software will perform poorly and these bugs will need to be debt managed to guarantee user satisfaction. Although more subtle, security debt can be far more severe for an organisation. Not least, in its complexity to repay and the broad negative factors associated with its disclosure or active exploitation.

This paper introduces software (or application) security debt while also outlining a number of considerations and practical strategies for managing the problem. It is important to recognise that strict adherence to the metaphor of financial debt is not necessarily possible in all instances and thus divergence occurs. Recx also does not attempt to assign a monetary value to the cost of security debt; due to the number of variables and unknowns unique within every organisation.

Whilst the focus of this paper is upon software security debt, security debt as a general concept has many other potentially applicable areas. These areas include for example: infrastructure security, end-user awareness/training, technology selection and operating procedures and processes. All of these areas can accrue security debt over their lifetime if not maintained appropriately. There is also some considerable interplay between the various security debt sources which should not be overlooked.

[1] http://c2.com/cgi/wiki?WardExplainsDebtMetaphor
[2] http://en.wikipedia.org/wiki/Technical_debt
[3] http://www.veracode.com/blog/2011/02/application-security-debt-and-application-interest-rates/
[4] http://www.veracode.com/blog/2011/03/a-financial-model-for-application-security-debt/
[5] https://www.techrepublic.com/blog/security/be-careful-not-to-incur-security-debt/7417

3 Software Security Debt

All software no matter how simple, is likely to carry a degree of security debt. As software complexity increases the likelihood of incurring security debt also increases. This relationship between development and security debt is analogous to development and bugs. The reason the two are analogous is due to the simple fact that some security defects are a type of software bug.

Debt should not necessarily be considered negatively, as an artefact of an economic development processes, the accrual of debt, security or otherwise is normal and good business. It is not likely that software will ever reach a debt free utopia. The knowledge of security debt is an important input into the risk profile and overall understanding of a product's or organisation's exposure.

The objective of having a security debt management process should be a business and development goal for software organisations. If the development and testing teams of an organisation say there isn't any debt; yet cannot demonstrate the steps taken to control it, then it's likely that a significant amount exists. If proactive steps are not taken, then the accrued security debt can in time become toxic. Debt that has become toxic can then require far greater level of investment to resolve, under externally dictated time-lines and typically under forced repayment conditions.

3.1 The Value in Measuring Security Debt

No organisation can avoid the presence of security debt. On this basis, no responsible security and risk aware organisation can reasonably ignore its presence.

As security debt cannot be avoided, it should be seen as another input to the risk management processes. Any attempt to avoid understanding or measuring the level of debt, is akin to not wishing to understand the risk exposure. This lack of visibility and understanding will result in a lack of an effective approach to mitigate or remediate the identified risks.

The value in measuring the level of software security debt is in the broader understanding of risk exposure it provides. This broader risk picture can then facilitate the understanding of:

- The business exposure to the risk from a security incident (forced repayment event).
- The speed and rate of payback of issues of differing severity.

Once the level of debt is understood it also facilitates strategic planning and metrics. For example:

- The potential for future expenditure to address known defects over the short to medium term.
- Debt trends and the gauging of the return-on-investment from the SDLC process.
- Understanding the percentage of the debt that is compromised of the OWASP top 10 or similar, facilitating additional prioritisation.
- Identification those security issues that can be linked together to achieve a higher level of impact.

3.2 Secure Development Life Cycles Identify Debt

It's important to understand the relationship between an SDLC and security debt. The precursor to an SDLC is security mindfulness[1]. Security mindfulness is where a formal SDLC may not be deployed throughout the organisation, but assurance processes or security related activities do occur at the different phases of development or testing. These activities will then likely mature into a full SDLC.

When adopting an SDLC the benefits of identifying vulnerabilities earlier in the life-cycle will be seen for new development. However, when SDLC or security mindfulness activities are applied to both new and old development there will prolonged periods of implementation debt discovery.

As these activities increase, the likelihood is that the volume of issues found in software will quickly start to out pace the resources available to resolve them on a per release or per product basis. The reason for the acceleration in the discovery of security issues can be numerous, however, likely drivers include:

- Increased manual code coverage.
- Increased use of static code analysis.
- Increased use of automated security testing (fuzzing).
- Development and testing team knowledge and awareness of security issues enabling identification.
- Root cause analysis and variation identification based on publicly disclosed flaws.

[1] A concept introduced in our January 2012 blog posting: http://recxltd.blogspot.com/2012/01/cost-of-following-sdl.html

As a result of this increase in the volume of issues and the associated resource constraints, organisations tend to focus only on the most severe issues. Over time, a mountain of security debt starts to grow fuelled by the volume of lower impact issues. However, while individual issues may be rated at a certain severity level, the same is not true for combinations of issues. That is to say, a number of distinct lower impact issues when combined or chained together, can carry equal impact to a single higher rated issue. While the complexity related to discovery and exploitation is greater, the ultimate impact can be the same. SDLCs today do not adequately deal with this scenario of aggregating lower severity issues to understand impact.

3.3 The Rise of Security Debt

Whilst it's tempting to think that the risk of security debt is not significantly different from that of technical debt there are important differences to consider. These differences stem from the fact that the impact on both vendor and users if this debt is discovered and exercised compared to technical debt is typically greater.

As the challenges of software security have become more widely understood, methodologies to identify and address these challenges have been developed[2]. The processes and procedures to improve software security typically manifest themselves as an SDLC in one guise or another. While an SDLC is a useful set of methodologies and processes for identifying, resolving or mitigating security exposures within software development, they are not without small print.

The reality is that SDLCs are variable in their application, coverage and cost, coupled with the challenge of actually addressing the issues once identified. At every stage of an SDLC when an issue is discovered there is a risk, cost, time and benefit analysis[3] for that version of the software product. The generally accepted wisdom is that identifying, mitigating or resolving a security weakness earlier in the life cycle is cheaper, is in Recx's opinion valid. However attempting to do so is not without any associated cost. This fact is sometimes lost in the SDLC rhetoric and needs to be kept in mind.

The tables below lists the different SDLC phases and provides examples of the security issues that can be identified and their example associated remediation activities.

3.3.1 Requirements and Story Phases

The table that follows shows that during a requirements (waterfall) or story (agile) security assessment there are two key types of finding. While security issues

[2] Similar methodologies have also been developed to address software quality.
[3] http://recxltd.blogspot.com/2011/12/business-v-security-bugs-risk.html

identified during this phase should in theory be the cheapest and easiest to resolve, it is not always the case. The reason for this fact is the divergence from accepted development practices. For example, requirements being written retrospectively or development having already started.

While these are not good development practices they are also a reality at times. As a result it can be just as challenging to resolve issues identified in this phase as it is in the later phases depending on the organisation in question.

Security Issue Identified	Remediation Activities
Existing requirement which carries with it security risk.	Documentation of risk and considerations. Revisions to requirement and impact analysis. New architecture and/or design considerations. New development training and effort to implement or meet requirements.
Missing security requirement.	New requirement documentation and impact analysis. New requirement change impact approval. New architecture and/or design. New development training and effort to implement or meet requirements.

3.3.2 Design and Architecture Phases

The design and architecture phases are some of the most critical when it comes to ensuring a robust security theme. Due to their critical nature to the success of a project it is also important that the output of a security assessment is risk assessed, where appropriate actioned and always tracked.

Security Issue Identified	Remediation Activities
Design or architectural security issue.	New requirements activities. Re-design and associated activities. Re-review of design or architecture. Additional development effort to meet the design.

3.3.3 Implementation Phase

The implementation phase is typically the most expensive within which to identify new security exposures. However, in Recx's experience it is also common for the most issues by volume to be identified during this phase. Recx are currently of the opinion that it's highly unlikely that no security issues will ever be found during an implementation assessment even with a mature SDLC. However, organisations should aim to at least minimise the risk of identifying fundamental requirement, design or architecture related security issues during this phase by performing assessments during the earlier phases.

Security Issue Identified	Remediation Activities
Missing security requirement or original requirement not implemented.	New requirements and associated activities (see previous table).
Design or architectural security issue.	Design or architecture changes and associated activities (see previous table).
Implementation issue.	Effort to analyse issue and perform root cause analysis. Effort to identify variations of the root cause across products. Effort to resolve vulnerability. Effort to update unit and regression test cases. Effort to understand affected existing products. Quality assurance effort to validate fixes and functionality. Security team effort to verify quality of remediation activities.

As the three previous tables show, it is possible to accrue security debt during any development phase. However, it's important to recognise that with an SDLC, whilst latent security debt is likely reduced (unknown debt) it is likely that in its place new known debt will accumulate instead throughout the different stages of the life cycle.

It is also important to recognise that in the last table a significant amount of effort will likely be spent analysing and performing root cause analysis and variation discovery compared to the other activities. The reason for the disproportionate amount of effort spent doing this analysis is typically due to a number of factors including:

- Quality of available information about the vulnerability or active exploitation.

- Complexity of software.
- Experience of staff in doing root cause analysis of security issues.
- Experience of staff in doing variation discovery of security issues.

It is also important to never underestimate the level of testing required after security fixes have been developed. Unit, functional and compatibility testing not only typically takes considerable time and resources, it also carries with it the risk of identifying reasons that a patch cannot ship. While this should not preclude security debt management it is important small print to the process.

3.4 Types and Sources of Security Debt

Before discussing software security debt management strategies it's important to establish a common vocabulary. The previous sections introduced briefly the concept of known and unknown debt and where in the development life cycle it can be accrued. The following list defines these two key types of software security debt in more detail:

- **Known security debt**: Security issues that have been identified (through assessment of requirements, design or implementation of software) but have yet to be addressed.
- **Unknown security debt**: Debt that occurs as a result of latent security issues in the requirements, design or implementation of software.

These two coarsely defined types of debt can then be applied equally to a number of distinct sources of debt. The influence a software vendor can bring to bear on these sources varies significantly.

- **Self Created Debt**: Security debt incurred due the software development practices and activities directly of the vendor.
- **Supplier Chain Debt**: Debt incurred as a result of the software development practices and activities of a vendor from whom the product producer licenses components (this includes both proprietary and open source code). In this scenario it is not a given that there is access to source code by the first party.
- **Dependency Chain Debt**: Debt incurred as a result of a dependency on an organisation where the vendor has no contractual control and access to or ability to change code. An example of this type of debt may include protocols a vendor needs to implement in order to interoperate, platforms and components they are dependent on or software their suppliers license.

When using these definitions it becomes clear that several sources of debt are outside of the direct control of the software vendor and thus can complicate and frustrate the debt management process.

3.5 Accruing Security Debt Based On Risk

In order to knowingly accept security debt for a product it is reasonable to expect the existence of a risk analysis process. However, it is important to realise that this risk analysis process is not purely based on technical security risk such as CVSS[4] or a *bug-bar*[5] but on many factors, some of which were covered in our article titled: *The Business v Security Bugs, Risk Management of Software Security Vulnerabilities by ISVs*[6]. Examples of the factors to consider as part of the risk analysis process include:

- Financial cost of resolution versus:
  - Revenue from product sales and licensing.
  - Cost of a response to the external discovery of a security exposure.
  - Brand (company and product) impact of external discovery.
  - Liability of security event in service orientated architectures or delivery models.
- Time cost of resolution versus:
  - Available resources and the onward impact to other development activities.
  - Time to market pressures.
  - Financial costs.
- Impact of issue versus:
  - Likelihood of discovery by a third party.
  - Presence of appropriate mitigations.
  - Complexity of exploitation and any prerequisite conditions.
  - Access requirements for exploitation in a typical deployment.
  - Customer and market expectation of security.

[4] http://en.wikipedia.org/wiki/CVSS
[5] Microsoft SDLC terminology
[6] http://recxltd.blogspot.com/2011/12/business-v-security-bugs-risk.html

However, in order to be able to apply a risk analysis process the security debt must be known or its presence reasonably suspected based on some indicator. If a supplier is identified as the source then appropriate agreements, processes and information needs to be available to accurately answer some of the points in the previous list. This creates a potential situation where a percentage of security debt is either known but not payable or worst-case unknown and latent debt. While the first of these may be manageable, the second is not in the majority of circumstances[7].

Although it's accepted by the authors that software can never be entirely security debt free; with management, it can reach a point where the debt is controlled and accepted by the business while in possession of the knowledge of level of exposure (debt / risk) that is being carried. However, this acceptance should be point-in-time as degradation can occur to a position. The risk profile of software isn't static as the inputs into the original risk analysis process can change. For example, debt managed through robust use of exploitation defences may attract increased risk should the effectiveness of those mitigations be reduced.

This relationship between security risk and debt highlights an important difference as well as an important interplay. Security debt management can be thought of as identifying, understanding and mitigating technical risk with an eye for repayment or expiry over a period of time. Whereas risk management is the application of a process to minimise, monitor, and control the probability and/or impact of security issues given certain constraints and thus allowing the organisation to go about its day to day business.

3.6 Importance of Latent Security Debt Resilience

Latent security debt is one of the most complex aspects of software security. The value of reducing the amount of latent debt has been demonstrated however, the reality is that there will always be a proportion. In the software development business the biggest challenge is what to do when latent debt is discovered and yet the facilities or resources don't exist to address it.

This highlights an important point in vendor supply chain management. Whilst processes may evolve in terms of the SDLC and security debt management, the same may not be true for upstream providers of software components. As a result success in being able to mitigate, remediate and respond to security issues is directly related to the abilities of these providers. Offsetting the risk of latent security debt is often related to the maturity of the processes and procedures around secure software development that dependent vendors adopt. The maturity of a supplier and how mindful they are of security and debt management, will directly relate to

[7] Caveat: Unknown latent security debt can be manageable where there are defensive mitigations in place; examples include operating system or developer tool chain defences

the estimated volume of latent debt that can be tolerated undiscovered in their components.

Through peer education and knowledge transfer there exists the opportunity to remediate, mitigate and potentially transfer latent security debt. These improvements become possible through the education of suppliers based on the experience of both secure development and prudent security debt management. Through this peer education, organisations can gain increased confidence in the process and reaction that will occur when a supplier is asked to pay back latent debt.

4 Debt Management

The concept of security debt has been introduced, and that it is identified through assessment activities and acknowledged through risk management processes. These risk analysis processes must be continually exercised and can be considered analogous to servicing of the debt. The reasons for an organisation choosing to service security debt can vary, but the most common reasons are typically:

- **Client Expectation**: Where the reputation of a product is directly related to security and as a result it is critical to minimise security debt.
- **Regulatory Requirements**: Where a product operates in a formal industry where there are regulatory requirements around product security in terms of lack of vulnerabilities or time to resolution once identified.
- **Increasing Cost of Debt**: As an organisation develops a broader portfolio of products; older products have the potential to attract increased debt. The debt is typically incurred relative to the stagnation of the product and the natural staff attrition within the organisation reducing the retained product knowledge.
- **External Capability Evolution**: A fault that is believed today to be difficult or complex to exploit does not remain a constant. With increased research into both software and hardware attacks (and countermeasures) comes increased capability and the likelihood that new exploitation techniques will be discovered.
- **Increased External Focus**: As the install base of a product increases so too does the likelihood of external scrutiny. Scrutiny can come from a number of sources, including clients using internal or external capability to assess the security of a product or externally funded researchers.

Many of the above items can be thought of as mini business cases, where proactive payment can avoid forced repayment of known security debt. If an organisation is not proactive there is the likelihood, as the products become more widely adopted, that a forced repayment event will occur. Recx previously discussed this problem and the vulnerability life cycle of software in our article titled: *Breaking the Inevitable Niche/Vertical Technology Security Vulnerability Lifecycle*[1].

4.1 Assigning Interest Rates to Security Debt

As with the repayment of financial debt a responsible approach looks towards paying the most expensive debt in terms of interest rates first. As a result it can be useful to assign all known security debt an interest rate (priority).

[1] http://recxltd.blogspot.com/2011/12/breaking-inevitable-nichevertical.html

Existing risk assessment methodologies[2] may use something similar to:

Threat = f (Motivation, Capability, Opportunity, Impact)

Where:

- **Motivation**: The degree to which a threat agent is prepared to implement a threat.
- **Capability**: The degree to which a threat agent is able to implement a threat.
- **Opportunity**: The requirements of access to be in a position to exploit.
- **Threat Agents**: Used to denote an individual or group that can manifest a threat.

Microsoft uses the DREAD[3] model to calculate risk.

- **Damage potential**: How great is the damage if the vulnerability is exploited?
- **Reproducibility**: How easy is it to reproduce the attack?
- **Exploitability**: How easy is it to launch an attack?
- **Affected users**: As a rough percentage, how many users are affected?
- **Discoverability**: How easy is it to find the vulnerability?

Where as CVSS[4] uses a combination of different properties to describe:

- **Base:** The intrinsic and fundamental characteristics of a vulnerability that are constant over time and user environments.
- **Temporal:** Characteristics of a vulnerability that change over time but not among user environments.
- **Environmental:** The characteristics of a vulnerability that are relevant and unique to a particular user's environment.

CVSS describes the need to use all three of these categories to represent the actual risk in a specific environment:

> "*The purpose of the CVSS base group is to define and communicate the fundamental characteristics of a vulnerability. This objective approach to characterising vulnerabilities provides users with a clear and intuitive representation of a vulnerability. Users can then invoke the temporal and environmental groups to provide contextual information that more accurately reflects the risk to their unique environment.* "

[2] http://www.dcs.gla.ac.uk/ johnson/infrastructure/slides/Vidalis.ppt
[3] http://msdn.microsoft.com/en-us/library/ff648644.aspx
[4] http://www.first.org/cvss/cvss-guide

Recx believe that experience prioritisation of security debt is based on a number of typical technical and business factors, many of which can increase or decrease the interest rate. These factors can be seen as less prescriptive than CVSS, more detailed than typical high-level risk models and with real world subjective reasoning due to the inclusion of certain business factors, including:

- **Impact**: What is the impact of the issue if exploited?
- **Distribution**: How widespread is the products use, and into which markets?
- **Disclosure**: How was the issue reported or discovered, and how well known is it?
- **Likelihood of discovery**: What is the potential for the issue to be discovered outside of the organisation's control?
- **Presence of mitigations**: Are their any effective mitigations to reduce the impact if exploited?
- **Complexity of exploitation**: What factors and knowledge are required for successful exploitation?
- **Access requirements for exploitation**: Are there certain circumstances or criteria that have to be met (outside of the product) before an attack can be brought to bear?
- **Customer expectation of security**: How security aware are users of the product, and what is their typical risk profile or appetite?

These inputs make up a mature vulnerability scoring system similar in nature to CVSS or Microsoft's DREAD but will act more like Microsoft's bug bar [5]. The resulting score (which can change over time) should then act as a guide to their priority in the remediation process.

An organisation should ideally look to continually repay debt, whilst actively looking for ways to reduce the interest rate associated with it. By looking to reduce the interest rates the debt becomes cheaper to sustain as the risk is reduced. This is because the likelihood of a forced repayment event caused by that debt is itself reduced. For example, take a critical security issue that has been risk managed to an impact level of medium via the deployment of defensive mitigations. Effectively the interest rate applied to the debt is reduced to a degree that even if it externally reported, an out of cycle patch would not be required.

[5] http://msdn.microsoft.com/en-us/library/windows/desktop/cc307404.aspx

While outside the scope of this paper to describe in detail how to manage latent unknown debt, consideration should be given to the applicability and re-use potential of the strategy adopted to repay known debt. Applying a similar approach when thinking about how to deal with the unknown debt problem can result in a suitable process. However, even with a latent debt strategy organisations should over time, through diligent and comprehensive review, strive to reduce the amount of latent unknown debt by converting it into a known manageable form. Combining these approaches and the education of suppliers can greatly reduce the volume, impact and ease of which latent issues can be dealt with.

4.2 Repayment

Repayment is the process by which an organisation reduces the security debt within its operations. It comes in a number of forms or strategies. These strategies are discussed in the following sections.

4.2.1 Repayment Through New Version Requirements

The requirements phase for a new version of a product is one place where an organisation can look to strategically reduce the interest they're incurring. For example, an organisation may choose to reduce the interest associated with memory corruption issues by raising a requirement that defensive memory corruption mitigations will be mandatory. While this does not pay back the security debt on the specific instances of memory corruption, it does reduce the urgency that they may need to be addressed with thus lowering the associated interest rate.

Generally new version requirements should not be used to address a specific piece of debt or issue but instead used to address or mitigate issue classes. That is not to say however that specific items of debt cannot be targeted through the plan of record for a particular version. However, requirements are generally at a higher level and not bug specific. As a result attempting to call out individual issues at the requirement phase is likely to be suboptimal.

4.2.2 Repayment By Severity Prioritisation in Future Releases

One of the most logical ways to reduce security debt via repayment, on a per release basis, is to prioritise all deferred security issues based on risk, as previously discussed. The output of the risk analysis process described earlier in this document would ideally look to place debt into one of the following five release buckets:

1. Next release (any type[6]).

[6] Typically, software has interim (patch), minor (service pack) and major release variants.

2. Next release (major version).
3. Next release +1 (any type).
4. Next release +2 (any type).
5. Next release +3 (any type).

While the volume of issues will need to be carefully considered, a bucketing approach allows a clear communication on which issues development must plan on resolving and the expected delivery vehicles. This approach allows for accurate time and resource planning, as these buckets will apply to issues found in previously released versions of the products. It also has the benefit of guaranteeing that issues will only exist for a maximum of four releases of that software product or component. It also does not derail existing product development by allowing the distribution of issues over a number of releases. As new initiatives are undertaken and new issues are discovered they too can be distributed over future releases on a rolling basis.

In order for this strategy to be successful organisations need to be mindful when issues are discovered late in the development process and a planned release with its associated issues is already under development. It is likely that if active security assessment work is undertaken during this time that additional vulnerabilities will be discovered. These new vulnerabilities although discovered prior to a planned release should not in the majority of cases be added to the work plan of the impending release. Instead they should be planned for a "*Next +1*" release at a minimum. The reason for taking this approach is to reduce release creep in terms of effort and resources. The danger of allowing release creep for late breaking security issues (unless catastrophic) is that the patch quality will be likely substandard. Additionally, there is the risk that by requesting fixes that other already planned features or bug-fixes may risk being either delayed or deferred. Or in the worst cases the entire release could be delayed.

4.2.3 Repayment By Percentage Reduction in Future Release

An alternative approach is the percentage to be resolved model. In this model instead of assigning release vehicles during the risk analysis process there are instead percentage goals for each new release. This model ensures that not only critical and serious issues are serviced but also a proportion of lower impact issues on an ongoing basis.

The downsides to this approach include that the severity of issues will be a result of the technical risk analysis process. As a result, issues found late in the development cycle may qualify for the next release. Also there is a key difference with the previous model, the development team will not have visibility during the planning phase of the exact volume of issues that must be fixed in that release. The benefit however is, that in theory at least, it guarantees releases without known critical security issues.

The table that follows provides examples of possible percentage targets for issues that would be resolved in each subsequent release of the product:

Severity	Percentage to be resolved
Critical	100%
Serious	50%
Moderate	30%
Low	20%

Classifying the issues as per the table above is one approach, however it does not account for volume. As such it may be appropriate to weight each of the severity levels (based on an analysis of the product and its user base) to allow a more detailed classification. Furthermore, when an issue is reported, where that event occurs within the development life cycle is also a key consideration. If the product is widely deployed and coming under increasing scrutiny, the flow rate of new issues between releases may increase to a point where it is unmanageable or impractical to have percentage targets.

It should also be noted, that the repayment of security debt is likely to be finite due to there being a finite number of introductions of new debt over a finite amount of time. However vulnerability volumes are not guaranteed to decrease in a steady fashion. That is, as with any software development process, it is possible to introduce additional flaws when fixing others and thus accrue more debt. Although difficult to gauge, the potential for incurring additional debt is ever present throughout any development process.

4.2.4 Maturing Debt - Forced Repayment

One subject mentioned in this paper but not discussed in detail is forced repayment. This is where the credit line is effectively withdrawn around a security issue due to external disclosure or active exploitation. The debt would thus immediately mature with not insignificant consequences. The awareness of external knowledge of vulnerabilities can potentially require immediate repayment if the severity of the security issue is sufficiently high or it is being actively exploited. As a result the overall cost of the issue becomes the development cost plus the costs associated with the response; such as: technical analysis, triaging, public relations, external communications, patch distribution, increased customer support and any out of band release effort. All of which will likely have a significant impact on an existing development schedules.

As a result forced repayment on someone else's terms is the worst-case scenario in terms of cost for most organisations. With inflated costs and disruption to existing development schedules it can become expensive quickly. Managing the volumes of issues that if discovered lead to these forced repayment events is what becomes the key goal of the risk management process.

4.3 Debt Expiry

Unlike typical financial debt it is possible for software security debt to expire without needing to get ones creditors to agree. This expiration will occur primarily because of discontinuation of the software product or replacement of components that have previously caused the organisation to accrue debt. The arguments around customer deployed products and the need to service security debt even after a product is no longer available for purchase is a contentious one. Customer expectation that software products will be serviced with free security updates for decades has historically been a taboo subject. However with mature vendors such as Microsoft dealing with the problem head-on by publicly declaring products firstly *end-of-life* and then *security update end-of-life* end user organisations are realising that security updates won't be produced forever[7].

Of course once an organisation declares a product end-of-life, it is only relinquishing its responsibility for that products ongoing technical and security debt (effectively it transfers to the user along with the risk). It is likely that a widely deployed product will continue to accrue new known debt (and potentially at an increased rate as new attack techniques develop and hoarded capabilities are released or exercised post end of life) for a considerable amount of time. There is also the potential for market pressure on expired debt that may force an organisation's hand with regard to either continuing support or releasing ad-hoc patches after a product is considered end-of-life.

4.4 Debt Overhang

Stuart Myers first introduced in his paper titled *Determinants of Corporate Borrowing*[8] (1977) the concept of debt overhang. This paper introduced and demonstrated the problem that large quantities of risky debt can lead to underinvestment in the immediate term. Some of these same concepts translate to software security debt. For example, if a large volume of security debt has been accrued then there is the danger that once external individuals become aware of the debt mountain and start

[7] Some companies (notably Microsoft) do offer paid for support following a product end-of-life. However, these in our experience are not the norm, and where they are provided, are done at prohibitively high cost and with limited guarantee of duration.

[8] Mirrored here: http://www.recx.co.uk/downloads/DeterminantsOfCorpororateBorrowing.pdf

to actively exploit then no more can be accumulated at a reasonable rate of interest. In software security debt the accrual is the shipping of software with a volume of known critical or serious vulnerabilities and the reasonable rate of repayment is resolving these vulnerabilities over future releases.

The reason for not being able to accrue more debt at a reasonable rate once the existing level of debt is known by customers, external security researchers or malicious individuals, is that there is a far higher risk of forced repayment events occurring. What this means in real terms, is that companies may be slowed down in terms of their innovation and time to market due to the need not to, or inability to, accept additional risks associated with security debt. Additionally, development resources that would otherwise be proactively repaying security debt at both strategic (requirements, design and architecture) as well as tactical (code) levels; are instead tasked with dealing with maturing debt overhang. As a result it becomes a tactical and not strategic repayment plan which ultimately is more costly.

4.5 Strategic Debt Restructuring

Taking a step back from the tactical and instead focusing on the strategic, software development organisations do have the opportunity to restructure their debt. In a similar vein to driving mitigations through new software requirements to reduce the interest rates, organisations can take the long view on debt and restructure accordingly. For example: if the expected lifetime of a component or product is only two years then that may be deemed uneconomical to repay or insufficient time may exist to downgrade the associated debt (the need to fix, is overridden by near term planned debt expiry). While the risk of a forced repayment event increases in the short term, the benefits of potentially deferring the repayment until its planned expiration may be economically viable or preferable for an organisation in the medium term.

The danger of taking this approach is twofold. Firstly, unless the component or product is guaranteed to be entirely replaced then the debt may not fully expire. This eventuality may only become apparent once technical development activities have begun or worst-case during the verification phase. Secondly, if component re-use has subsequently occurred across other development teams, components or products, then by not addressing the debt when the opportunity first arose, the debt has instead multiplied instead of being reduced.

In Recx's opinion, this is a high risk strategy, and should generally be avoided unless appropriate safe guards around component re-use and due diligence, to ensure complete removal, are adopted.

4.6 Bankruptcy

In his February 2011 post, Chris Wysopal referred to products being declared security debt bankrupt and sited the example of Microsoft performing a ground-up rewrite of Internet Information Server (IIS) between versions 5.0 and 6.0. Although a plausible strategy, there are few companies with the resources of Microsoft able to maintain a debt laden enterprise product, whilst developing an entirely new code base.

Furthermore, with current mitigations against certain classes of issues and defensive strategies, this approach does not need to become a reality for many companies. Instead most software vendors will likely prefer to continue with a risk managed approach to the debt whilst either engaging internal or external security code auditing specialists to help refinance any long term exposure. Organisations will also look leverage platform or developer tool chain defences effectively or alternatively force debt expiry by replacing vulnerable components over time.

In summary while Microsoft is often considered a role model by many quarters of when declaring security debt bankruptcy can be good; and whilst it is agreed that it can be the best approach at times, it is unlikely to be economically viable for most organisations.

4.7 Non Repayment - Consequence Planning

While preparing this paper for final release an interesting perspective on vulnerability repayment came to light. This perspective presents an interesting and disruptive counter perspective to the approaches discussed.

The original quote from Government Security News[9] is:

> "*We may be at the point of diminishing returns by trying to buy down vulnerability,*" *the general observed. Instead, he added,* "*maybe it's time to place more emphasis on coping with the consequences of a successful attack, and trying to develop networks that can* "*self-heal*" *or* "*self-limit*" *the damages inflicted upon them.* "

This perspective is included as 'food for thought' as one potential route to how security debt and risk management may change in the future. However, Recx do not consider this as sufficient reasoning to not adopt or abandon an SDLC or to fail to consider or manage security debt.

[9] http://www.gsnmagazine.com/node/25682?c=cyber_security

5 Conclusions

In this paper the concept of security debt has been covered, with the focus on its applicability to software development. Types of debt and their sources have been described, coupled with debt accrual and strategies for servicing and repayment over multiple releases based on risk.

For mature software development organisations such as large independent software vendors, out sourced development houses, internal teams or those who service high security industries (such as those who comprise the critical national infrastructure) the understanding and effective management of software security debt is the next logical step in process evolution. If effective management strategies are not adopted, then the reality is that over time an increasing volume of low to medium severity issues will likely continue to grow while new critical and serious security issues will not have an agreed release cycle.

While there can never be a panacea to the software security problem, understanding how security debt is incurred, and how it can effectively be managed, increases the chance of a sustainable on-going process. The risk of not having such a set of processes in place is that the debt mountains of different severities of issue will become so large that when combined they outweigh in terms of impact traditional critical and serious security issues.

5.1 Further Reading on Software Security

If you're interested in reading more about Recx's perspective on software security we've published a number of articles on our blog[1]:

- The Business v Security Bugs - Risk Management of Software Security Vulnerabilities by ISVs[2]
- Breaking the Inevitable Niche/Vertical Technology Security Vulnerability Lifecycle[3]
- The Cost of Following an SDL[4]
- Musings on Secure Software Development[5]
- Risk Appetite - The Need for Security SLAs[6]

[1] http://www.recx.co.uk/blog.php
[2] http://recxltd.blogspot.com/2011/12/business-v-security-bugs-risk.html
[3] http://recxltd.blogspot.com/2011/12/breaking-inevitable-nichevertical.html
[4] http://recxltd.blogspot.com/2012/01/cost-of-following-sdl.html
[5] http://recxltd.blogspot.com/2012/01/musings-on-secure-software-development.html
[6] http://recxltd.blogspot.com/2012/01/risk-appetite-need-for-security-slas.html

6 Thanks

Prior to publication this paper was sent to a number of our industry peers for review and comment. Recx would like to recognise and thank the following people for their comments and suggestions.

- Haroon Meer - Thinkst.
- Dr. Tim Watson, Director of the Cyber Security Centre - De Montfort University.
- Phil Huggins - Black Swan Security[1].
- The people who took the time to read and provide comment and feedback, yet didn't want to be credited.

[1] http://blog.blackswansecurity.com/

COPYRIGHT 2009-2012 RECX LTD.