Towards an analytic model of security flaws

Chris Tofts, Brian Monahan
Trusted Systems Laboratory
HP Laboratories Bristol
HPL-2004-224
December 10, 2004*

E-mail: chris.tofts@hp.com, brian.monahan@hp.com

security, models, flaws, branching process, analytic

A simple model of the dynamics of flaws within a software security system is presented. We demonstrate how this model can be fully captured by a Galton-Watson branching process and thus can be effectively calculated upon. Using the limit behaviour of a Galton-Watson branching process, we can demonstrate how a multi-layered security system can become secure even with 'poor' flaw correction. Finally we make some observations about how the parameters of our models can be estimated and how further results from branching processes could be exploited within security systems.

* Internal Accession Date Only
Approved for External Publication

Towards an analytic model of security flaws.

Chris Tofts & Brian Monahan
chris.tofts@hp.com & brian.monahan@hp.com
HP Laboratories, Bristol

November 26, 2004

Abstract

A simple model of the dynamics of flaws within a software security system is presented. We demonstrate how this model can be fully captured by a Galton-Watson branching process and thus can be effectively calculated upon. Using the limit behaviour of a Galton-Watson branching process, we can demonstrate how a multi-layered security system can become secure even with 'poor' flaw correction. Finally we make some observations about how the parameters of our models can be estimated and how further results from branching processes could be exploited within security systems.

1 Introduction

Security is essentially a gamble. Controlled access in some manner is given to an object, which carries some benefit, but as a consequence there is the prospect of undesired users exploiting the access to gain unauthorised access (i.e. reading or writing), of the protected object. The decision as to whether to grant access and the mode whereby it is granted is fundamentally a gamble based on the difference between the expected pay off for granting access and the expected losses caused by unauthorised accesses. The problem in this setting is being able to evaluate in a consistent manner the relationship between these two values.

One of the major challenges in performing such an evaluation is the complexity of security systems. Modern security systems tend to have a large software element as part of their construction and these are notoriously difficult to validate with any particular degree of confidence. However, stretching the gambling analogy, if one was offered a bet on the repeated toss of a fair coin whereby for every pound staked the counter-party would give you three pounds if the coin showed heads, then one would hope that any reader of this document would readily accept such a bet. The toss of a coin is a highly complex system, and predicting the outcome of any particular toss is effectively impossible. However, there are situations where the phenomenological observation that the number of heads is approximately equal to the number of tails is sufficient information to make rational decisions.

Unlike other software elements security systems actually present us with a major advantage, since we can effectively layer them – so-called "defence in depth". In other kinds of software product, it can either be very difficult or even ridiculous to produce a layered solution, where if only one layer functions then the totality functions – consider attempting to do so when simply editing text for instance. In the security context it is not unusual to have many *independent* layers

attempting to ensure security. The intended effect of each layer is to significantly increase the amount of advance information and effort required by the attacker to overcome and penetrate that barrier. Therefore, all defence barriers would need to be overcome by an attacker for them to acquire unfettered access. With this in mind, we may change both our approach and analysis to the presence of flaws within such systems.

As in many scientific challenges, it is relatively straightforward to provide a set of desirable properties for a model of security flaws which:

- is underpinned by obtainable data, which can be exploited to derive falsifiable predictions;
- provides an understanding of the dynamics;
- provides a computationally effective and repeatable method of calculation;
- gives an ability to support the decisions we need to make in respect of these modules
  - what are the long term outcomes of using and maintaining a particular module;
  - at what point do we abandon a module and replace it by starting afresh;
  - which of our modules can achieve, or has achieved, a secure state;
- potentially gives an ability to quantify, track and predict the level of system security exposure over time.

There are also other trade-offs to consider such as:

- do we improve our software engineering or
- do we improve the security system configuration, the pattern and interaction of modules

In this paper we shall present an approach based on Galton-Watson branching processes [3] to calculating the potential number of flaws within any security element as a function of the time it has been exposed to malicious intervention. We exploit this approach as it gives us the possibility of

2 Our model's assumptions

Sadly all human constructions are prone to error, and this seems to be particularly true of large software constructions. After the failure of human processes, failings of the software elements, due either to mismatches of implemented function against what is needed or due to misconfiguration, are one of the major causes of security breaches. For tractability, we suppose that software failings are a consequence of flaws within the current code supporting particular security functions. Our model can be summarised as follows:

1. any particular software (security) element begins its life, as exposed to malicious intervention, containing a number of flaws (occasionally zero);
2. flaws are then discovered after release, typically by security developers and professionals who may:

   (a) report it in a responsible, confidential manner to the software maintainer (commercial or open-source), typically in exchange for the vendor giving them credit later for having found the issue, once the patch is released by the vendor.
   (b) publish it more widely, probably anonymously, in a grey hat forum, but without directly alerting the vendor.
   (c) keep it quiet and exploit it in some fashion (i.e. black-hat).

3. Over time, certain flaws may eventually come to be exploited by hackers in terms of virus's etc.;

4. When a flaw is brought to the attention of the software maintainer, usually after having been exploited, it will be repaired after some delay. This repair is then released in the form of a *patch*;

5. But, being code, these repairs are also not perfect. The code may correct the flaw and not introduce new ones, however experience tells us that not only may it leave the flaw in a different form, but it may introduce many new ones.

In practice, there might be a considerable time lag between discovery of a flaw and the subsequent installation and application of the repair by end-users. In the interests of simplicity and tractability, we assume that repairs are applied assiduously by users as soon as they are made available.

The term flaw refers to any aspect of the system that exposes the users and deployer to behaviour from third parties that they do not desire. These flaws can arise from many sources: incorrect usage of a module; incorrect coding of a module; incorrect configuration of a module; inappropriate behaviour by users; and poorly understood requirements. Whilst, at this point, we do not address the issue of changes induced as a result of requirement evolution, it is clear that the general model we present in the sequel could be exploited to add the flaw effects of these modifications.

As a further assumption we can assume that the number of hackers is large and that the probability of any individual finding and exploiting a flaw is small, and therefore that an exponential rate will be a reasonable description of the frequency with which any particular flaw is exploited and that exploitation detected.

3 Basic Theory of Galton-Watson Branching Processes

Galton-Watson branching processes [2, 3, 6] were originally introduced to explain the persistence of family names amongst the British aristocracy. Their fundamental view is a system of particles that are capable of *independent* replication. At each generation each particle is replaced with a number of particles in accordance with some probablity distribution. An example of a branching process is presented in figure 1

At this point it is convenient to introduce the notion of a **probability generating function** (pgf) which for a discrete probability distribution is defined as follows:

$$g(s) = p_0 + p_1 s + p_2 s^2 + \ldots + p_n s^n + \ldots$$

which has many useful properties in this context. For an introduction to pgf's see [2], but the main property we are interested in is that the n^{th}generation of a branching process starting with one

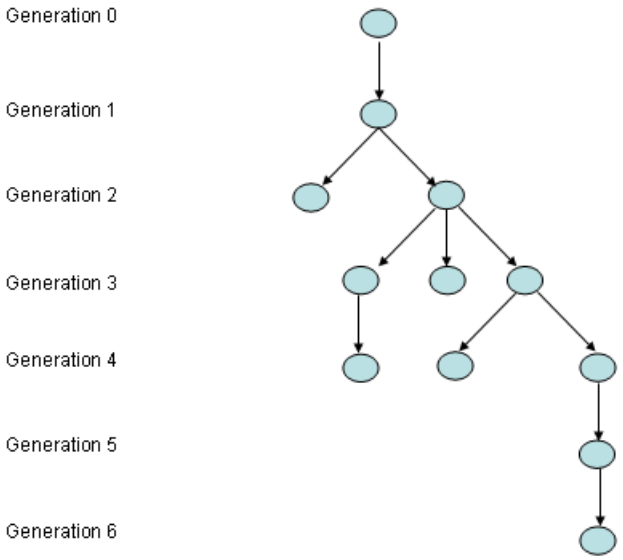

Figure 1: An example branching process reaching extinction at generation 7.

particle is given by

$$g(g(\ldots n \text{ times } (g(s)) \text{ or } g^n(s)$$

that is the n fold application of the pgf to itself. For our example branching process, in figure 1 given the presented data we can estimate the pgf as being the following:

$$\frac{5}{13} + \frac{5}{13}s + \frac{2}{13}s^2 + \frac{1}{13}s^3$$

given the observed reproductive frequencies.

In this context there are two straightforward properties that we can examine.

1. does the population grow in the limit
2. with what probability does the population go extinct

The first question can be answered by evaluating the mean of the replication pgf μ if this is less than 1 then extinction is certain, if it is greater than 1 then the population expectedly growing unboundedly, however it can still have a non-zero probability of extinction. This is in contrast with continuous models of replication, where populations with an effective replication rate greater

than one will always grow unboundedly. The observation of extinction plays a key role in security systems since this could be taken to indicate the absence of flaws in the system. The extinction probability can be found as the lower fix point solution of the generating function $g(s) = s$ []. In our example case the mean is:

$$\frac{5}{13} + 2 * \frac{2}{13} + 3 * \frac{1}{13} = \frac{12}{13}$$

so we should expect the process to eventually die out.

4 Our model as a branching process

Considering the elements of the branching process to be the flaws within our system, it is clear that we can represent the elements within our model as a branching process. Let us suppose that the results of trying to fix a software flaw result in a number of new flaws with a probability generating function $g(s)$. Whilst this pgf may well be itself dependent on the complexity and type of the particular software element for the moment we shall consider that there is one representative pgf for software.

From the security standpoint we anticipate that we only fix a flaw after a successfully detected attack on that flaw. If we take the probability of such an attack as p_a then the appropriate pgf for a security software element is:

$$ss(s) = (1 - p_a)s + p_a g(s)$$

4.1 Some Observations

The first important fact is the long term growth in the number of flaws for that we need the mean μ_{ss} of $ss(s)$ which we can calculate as follows, writing the mean of $g(s)$ as $\mu = 1 + \delta_\mu$:

$$\begin{aligned} \mu_{ss} &= (1 - p_a) + p_a\mu \\ &= (1 - p_a) + p_a(1 + \delta_\mu) \\ &= 1 + p_a\delta_\mu \end{aligned}$$

Consequently, μ_{ss} is greater or less than one if and only if μ is. More significantly, the rate of attack is unimportant in determining wether the number of flaws is likely to grow unboundedly.

Whilst ideally we would expect to reduce the number of flaws in a system, this may not be a limit on the use of such systems. Recall that the lower solution for $g(s) = s$ tells us the extinction probability even if the expected number of flaws is growing with each fix. Solving for the fix point of $ss(s)$, assume that we have s^* with $g(s^*) = s^*$ then we have:

$$\begin{aligned} s &= (1 - p_a)s + p_a s^* \\ p_a s &= p_a s^* \\ s &= s^* \end{aligned}$$

Hence, again, the extinction probability of a flaw is independent of the rate of attack.

The later observation is particularly important if we have a layered security system. Whilst we may not be capable of expectedly reducing the number of flaws in a system each time we fix, we may still be able to achieve a desirable level of security in the long run.

Consider a software security system with N independent layers, then for each layer we have a long run probability of flaw extinction of p_e then the probability of successfully hacking the system, that is simultaneously breaking *all* of the security layers is:

$$(1 - p_e)^N$$

hence for any non-zero p_e there is an N for which the exposure probability can be below any particular required level. The good news is that, even in the presence of poor flaw control, it is possible in the limit to eventually achieve arbitrarily good security.

4.2 Beyond Repair

The probability of extinction above is dominated by the early successful repair of modules. So a reasonable question to ask is at what point do we abandon a module. A reasonable answer to that question is when the probability of flaw extinction has become too small. As a simple answer to this problem, we recall that to begin the branching process with n initial flaws, we may simply substitute s^n into the start of the process. Alternatively, if we have a probability of flaw extinction of p_e with one flaw then we know that the probability of extinction starting with k flaws is p_e^k. Assuming that we start with one flaw then, after repairing f flaws, we have a pgf for the number of flaws given by $g^f(s)$ with probability $p_f(i)$ that the number of flaws is i. So, the probability of successful flaw extinction now is given by:

$$\sum_{i=0}^{i=\infty} p_f(i) p_e^i$$

and consequently we can make the decision as to whether it is appropriate to consider continuing to use this module, or that the probability of success has fallen such a low value as to no longer be worth while.

5 How to obtain data about security flaws

In a nutshell, the problem is how to obtain reliable data about the presence of security flaws from original software developers and maintainers - who would naturally like to keep this information to themselves and their immediate set of customers and users. Fortunately, this issue has been recognised as affecting public confidence in software systems and related services. Accordingly, a number of public sources of information about incidents and flaws in common infrastructure components (operating systems, databases, firewalls, etc.) have become widely available.

5.1 Attack software and exploits

Unfortunately, the information about specific individual flaws doesn't come in a pure distilled form, but is often bundled together with other information in the form of "incident reports" concerning exploits.

Various CERT organisations (Computer Emergency Response Teams) exist around the world to provide a public forum for reporting this information. They have taken the lead in aggregating and publishing "incident alerts" in a public manner. The incident alerts tend to announce information

about generic threats or packaged attacks such as worms and virus's. These are usually based on flaw-enabled attacks or Trojans that depend upon problems (e.g. buffer overflows or common misconfiguration) in common software, such as operating systems, that most systems will rely upon.

National CERT organisations (incident alerts):

- US-CERT : `http://www.us-cert.gov/`
- UNIRAS, UK Government CERT : `http://www.uniras.gov.uk/`
- AusCERT : `http://www.auscert.org.au/`

Anti-Virus companies - data bases of known viruses. - Symantic, McAffee, Sophos,

5.2 Security flaws and vulnerabilities

Actual security-related flaws in common software are publicly documented and declared by entities like CVE and OVAL (both hosted by US Government contractor, The MITRE Corporation) or through organisations like the Open Source Vulnerability Database. Reported flaws are graded and assessed for security-related impact. In many ways this is the purest and most direct public source of data about flaws in the sense that the reports relate to specific systems and their systems internals. As will be seen from the sources below, many actual flaws are not only functional errors in software, but also simply unanticipated or creative new ways in which combinations of software component systems can be put. Systems generally have people somewhere in the loop, whose judgement will used in some way. It is true (but unhelpful) to observe that if only we knew all the ways in which our software would or could be used, then we could suitably plan and protect our assets for all eventualities. However, in the absence of a perfect oracle for the future, software functionality will continue to be exploited. This is because the one thing we can be sure of is that complex software will have dependencies on other software, ensuring that the potential for making mistakes and causing misuse/abuse is ever present. A somewhat less defeatist position is to say that the more informed we are about software and its various dependencies, the greater the possibility there is for reducing the impact of and the potential for software flaws. For an excellent wide ranging discussion of many issues related to safety (and hence, implicitly, security) in software, see [5].

Vulnerability data bases:

- CVE (Common Vulnerability and Exposures) : `http://www.cve.mitre.org/`
- OVAl (Open Vulnerability Assessment Language) : `http://oval.mitre.org/`
- Open Source Vulnerability Database : `http://www.osvdb.org/`

Developer-oriented security incidents and assessments:

- Bugtraq : `http://www.securityfocus.com/archive/1`
- SecurityFocus Vulnerability Database : `http://www.securityfocus.com/bid`

Data from Commercial Software Manufacturers:

- Microsoft Knowledge Base : `http://www.microsoft.com/technet/security/current.aspx`

5.3 Estimating branching process parameters

Clearly obtaining direct data on flaws within systems is difficult. It is self evidently the case that if it is known that a flaw is present then it will be removed. So the nature of the data that can be obtained about the presence and dynamics of flaws will inevitably dictate how we approach estimating the parameters [1] within the branching process description. The most readily available data is the interval between flaws, and clearly we will need to exploit this in order to estimate the parameters of our branching process model. Unfortunately, this data is the conflation of the repair success distribution and the attack rate, so we shall obviously need to combine data between modules to estimate these two different structures.

In an ideal world there would be experimental data on the effectiveness of error correction by programmers. Whilst we are aware of data on the introduction of errors into new coding projects [4], we are not aware of matching data on the effectiveness of repair.

6 Conclusions and further work

Whilst we have concentrated on the dynamics of flaws within the software elements of a security system, there is no reason to suppose that the same approach may not equally apply to both the hardware and human process elements of any system. If the life history of such mechanisms and procedures was available we could exploit that to estimate the appropriate probability generating functions and then apply the branching process techniques in those settings equally as in this one.

There is considerable scope for augmenting the complexity of the branching process models we employ. In no particular order we could consider:

- Given the available data, can we derive a valid strategy for the maintenance and replacement of security software elements;
- What would such a strategy suggest on the balance between improving maintenance processes and the construction of the security systems, layers versus flaw survival reduction;
- Can we explain the time variation in the flaw dynamics from the attack probabilities alone, or are different flaws exposed to different repair distributions;
- Time varying flaw introduction pgf's, that is as a module ages as a result of increasing complexity it may become more and more difficult to achieve error free flaw correction, c.f. the experience of IBM360 operating system;
- We could develop a full calculus whereby the independence and interdependence of various security elements is directly represented within the branching process model;
- We could add types to flaws, either representing severity of incident or difficulty of correction, and this would sit well within the multi-type branching process;
- A detailed analysis of databases of flaws both within security systems and software systems could be used to try to more accurately estimate the parameters within our models;
- Develop parameter estimation techniques which exploit the fact that we do not necessarily have a single instance of a branching process but potentially many independent observations.

- We could use the continuous time models and then exploit some of the effective approximation techniques to calculate our exposure probabilities.

Branching processes offer a natural and computationally effective method of evaluating the exposure levels within complex security systems, and the further exploitation of these methods may give significant insight into the operation and construction of such systems.

One of the implications of adopting this model of security flaws is on the tracking of data about the evolution of systems. It firstly means that we need to agree on what is a flaw, what the component of analysis for flaws should be and so on. Importantly, when a flaw within a component is fixed, we should try and ensure that the *identification* between flaw and repair is recorded as well. The current data gathering on security problems is in the form of natural history and carefully observed, recorded behaviours. As such, this forms a vital starting point for any analysis of evolving software systems. However, the generation of agreed models can additionally have an enormous impact both on the nature of the data gathered and obviously on its interpretation. The view of the dynamics of security flaws presented in this paper gives a potential underpinning for an hypothesis-led scientific approach to these dynamics.

Acknowledgements

Chris thanks Dale E. Tanneyhill for being such a pain about branching processes and Mel Hatcher for keeping him under control during this time. Brian thanks Adrian Baldwin for helpful discussions concerning the nature of security flaws and the assumptions underlying our model.

References

[1] P. Guttorp, *Statistical inference for branching processes*, Wiley, New York.

[2] W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1, (3rd Ed.), Wiley, 1968

[3] T. E. Harris, *The theory of branching processes*, Dover Phoenix Editions, 1989.

[4] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumptions of independence in multiversion programming. IEEE Transactions on Software Engineering, SE-12(1):96-109, Jan 1986.

[5] N. G. Leveson, *Safeware – System Safety and Computers*, Addison Wesley, 1995.

[6] D. E. Tanneyhill, A.M. Dunn and M. J. Hatcher, The Galton-Watson branching process as a quantitative tool in parasitology, Parasitology Today, 15(4):159-165, 1999.