Towards a Rapid-Alert System for Security Incidents

Stefan Rass

Department of Applied Informatics, System Security Group,
Klagenfurt University, Austria
stefan.rass@uni-klu.ac.at

IMF 2011
6th International Conference on IT Security Incident Management & IT Forensics

May 11th, 2011

Contents

1. Introduction
2. Preliminaries
   - Security as a Game
   - Incorporating Trust
3. The Framework
   - Belief Models and Bayesian Updating
   - Results
4. Conclusion and Outlook

Contents

1. Introduction
2. Preliminaries
   - Security as a Game
   - Incorporating Trust
3. The Framework
   - Belief Models and Bayesian Updating
   - Results
4. Conclusion and Outlook

The Problem

As everyone knows...

- Attacks never become less sophisticated,
- Security breaches are hardly foreseeable and regularly reported for many standard software tools,
- Long-term security is often more a hope than a fact.

The question

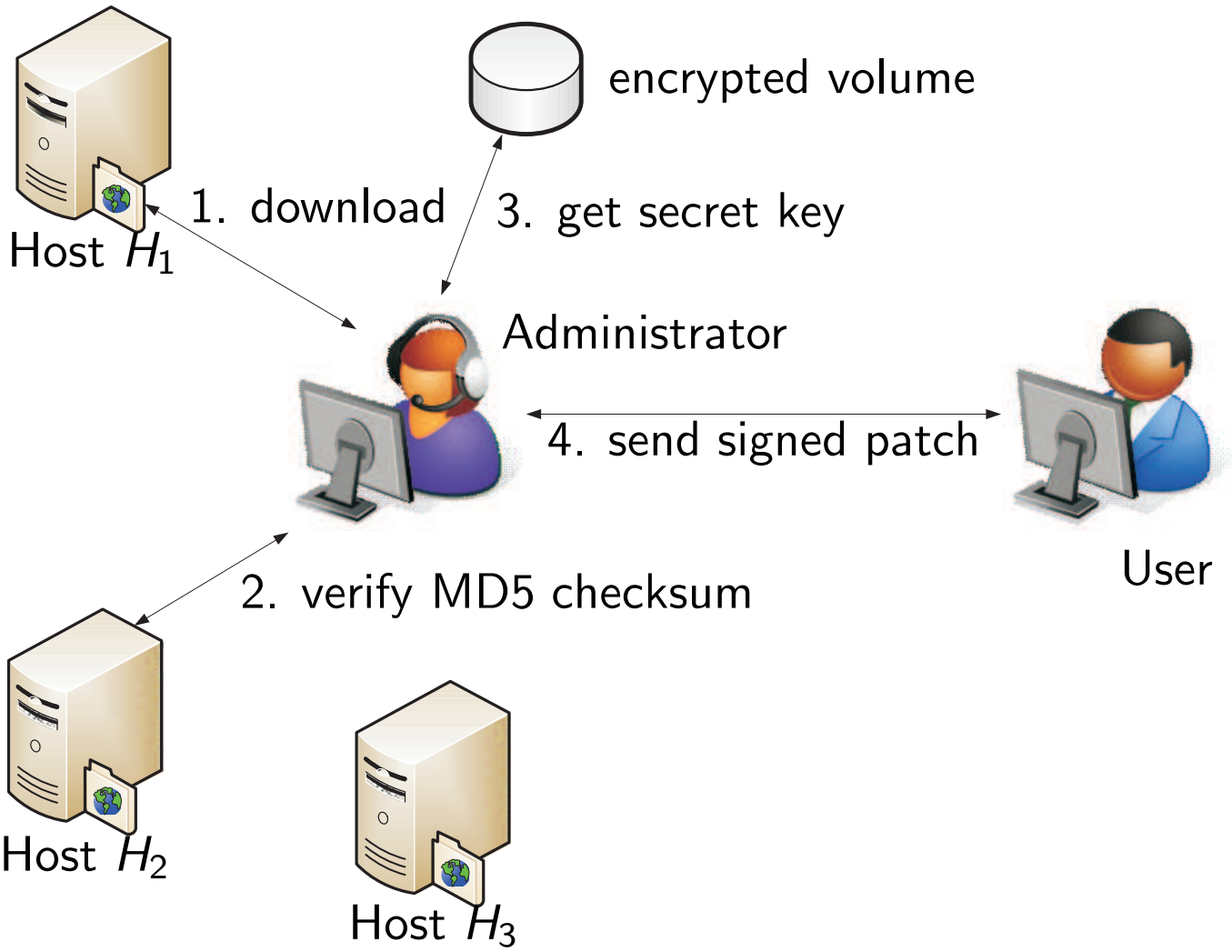
Lacking a properly formalized notion of trust, can we somehow give a reasonable estimate of "how much longer" we can consider our security infrastructure as "secure"?

Example – Software Patches I

To illustrate the method and results, consider the following scenario:

- a system administrator regularly distributes patches for some software
- the patches can be downloaded from three mirrors H_1, H_2 and H_3
- he downloads the patch from one mirror and verifies its MD5-checksum against the data from another mirror.
- he sends the digitally signed patch to the user for installation

Example – Software Patches II



Contents

1 Introduction

2 Preliminaries
- Security as a Game
- Incorporating Trust

3 The Framework
- Belief Models and Bayesian Updating
- Results

4 Conclusion and Outlook

Quantifying Security using Game-Theory I

Security is nothing else than a competitive two-player game:

- Player 1: the honest user.
- Player 2: the adversary.
- Payoffs: player 1 wins the game (payoff $= 1$), if no security breach occurred. Otherwise, player 2 wins (payoff for player 1 equals 0).

Why games?

As eloquently noted by T. Alpcan and T. Başar[1]

> *In a zero-sum setting, if the defender adopts a Nash-equilibrium strategy, it does not matter whether the attacker is rational since any deviation from Nash-equilibrium solution will decrease the cost of the defender and benefit of the attacker. Therefore, when properly implemented by the members of the organization or by security systems, Nash-equilibrium solutions (in zero-sum games) represent the defense strategies against competent attackers in the worst case.*

[1] Tansu Alpcan and Tamer Başar: *Network Security: A Decision and Game Theoretic Approach*, Cambridge University Press, 2010.

Quantifying Security using Game-Theory II

Actions for Player 1...

...comprise all parameters that he can freely choose when using the infrastructure. Examples are: proxy servers (for multi-hop/multi-path relay), different encryption and signatures suites, download mirrors, etc.

Actions for Player 2...

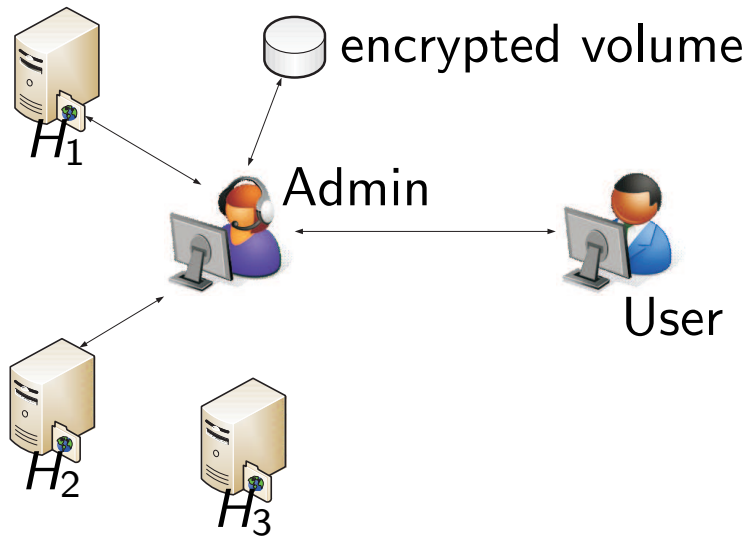
...comprise all ways in which the system can be attacked. Identification of such strategies is up to topological vulnerability analysis and appropriate decision support tools (security auditing, etc.).

Security as a Game – Example I

Going back to our previous example, both players have three strategies, assuming that at most two mirrors can be hacked simultaneously.

Strategies for player 1:

- $s_1^{(1)}$: Query mirrors M_1, M_2
- $s_2^{(1)}$: Query mirrors M_1, M_3
- $s_3^{(1)}$: Query mirrors M_2, M_3

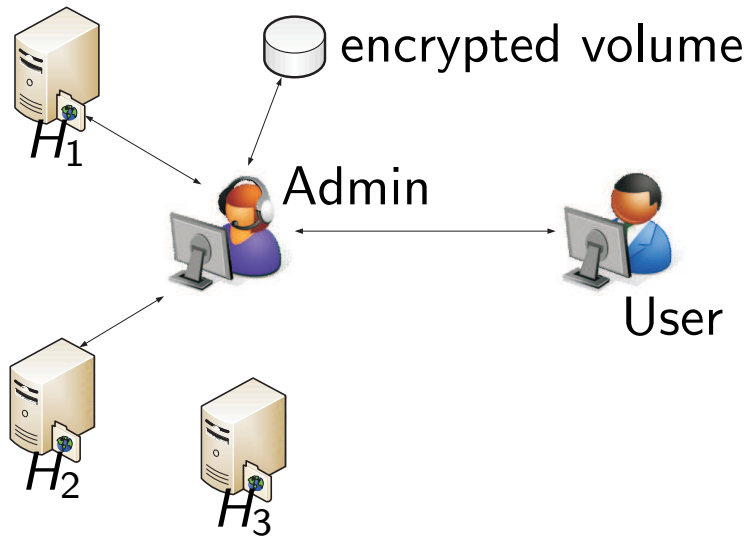


Security as a Game – Example II

Assuming that at most two mirrors can be hacked simultaneously, player 2 (the adversary) may act as follows:

Strategies for player 2:

- $s_1^{(2)}$: Compromise mirrors M_1, M_2
- $s_2^{(2)}$: Compromise mirrors M_1, M_3
- $s_3^{(2)}$: Compromise mirrors M_2, M_3



Security as a Game – Example III

Player 1 wins if and only if both mirrors that he accessed have been hacked.
We get the following payoff structure for each scenario $(s_i^{(1)}, s_j^{(2)})$:

$$A := \begin{array}{c} \\ s_1^{(1)} \\ s_2^{(1)} \\ s_3^{(1)} \end{array} \begin{array}{c} \begin{array}{ccc} s_1^{(2)} & s_2^{(2)} & s_3^{(2)} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array},$$

which along with the strategy lists completes the game-theoretic model.

Results from the Game-Theoretic Model I

Central idea of game-theoretic security assessment

Play both roles (honest user and attacker) simultaneously, and seek the optimal strategy for both parties, assuming that the adversary's goal is causing maximal damage (zero-sum game).

What do we mean by "optimal"?

- if the game is played only once, then
  payoff indicates success or failure.
- if the game is repeated, then the
  expected payoff = probability of being successful,
  if we draw our moves from a probability distribution x^*.

Results from the Game-Theoretic Model II

The security game

- is an endlessly repeated competition in which...
- ...the honest user draws his actions from some probability distribution x^* (randomized decisions).
- The attacker as well acts randomly according to y^*.
- The expected payoff $=$ Pr[player 1 wins] $= (x^*)^T A y^*$, where A is the payoff matrix.

We seek optimality, i.e. a Nash-equilibrium (x^*, y^*) satisfying

$$x^T A y^* \leq \underbrace{(x^*)^T A y^*}_{=v(A)} \leq (x^*)^T A y$$

for all randomized decision rules x, y.

What does this mean?

Let $A \in \{0,1\}^{n \times m}$ be a payoff matrix, then

- $v(A) \leq \Pr[\text{Player 1 wins}]$, regardless of how player 2 behaves (worst-case defense).
- x^* is the optimal worst-case defense (probabilistic behavior)
- y^* is the adversary's best way of attacking (the above bound is sharp!)

How to bring trust into the game? I

Suppose that mirrors are not fully trustworthy, and let us express our belief as a probability

$$p = \Pr[\text{two mirrors have been hacked}],$$

then our payoff is slightly changed, because

- if exactly one of the two mirrors could not be compromised, then the adversary cannot hope to win, but
- if both mirrors have been attacked, then with a chance of $1 - p$, we will still detect the attack and the adversary looses.

How to bring trust into the game? II

The payoff matrix for our game is now depending on the parameter p that accounts for our trust into the system:

$$A(p) := \begin{array}{c} \\ s_1^{(1)} \\ s_2^{(1)} \\ s_3^{(1)} \end{array} \begin{array}{c} \begin{array}{ccc} s_1^{(2)} & s_2^{(2)} & s_3^{(2)} \end{array} \\ \begin{pmatrix} 1-p & 1 & 1 \\ 1 & 1-p & 1 \\ 1 & 1 & 1-p \end{pmatrix} \end{array},$$

Sources of Trust and Scepticism

If a probability p shall capture the belief into our system, there are several sources that might help finding an appropriate choice for p. A few examples include

- RSS feeds
- software patches
- personal experience and communication

We shall pursue a *Bayesian* updating strategy for incorporating incoming information in our model $A(p)$.

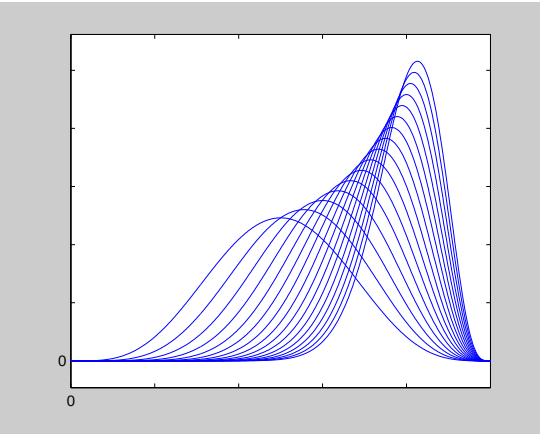
A Bayesian Belief-Model I

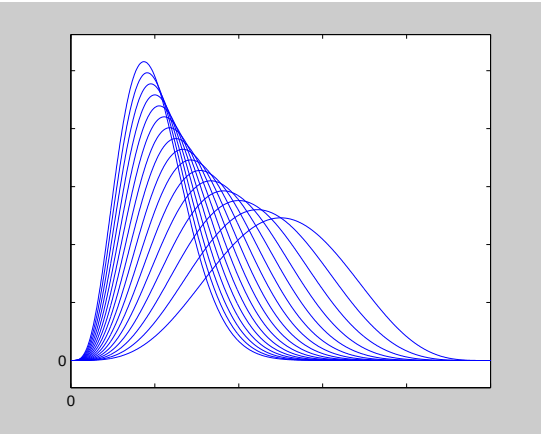
- Parameter to be modeled: probability
- The most convenient model for this: Beta-prior Beta(a, b)
- Hyper-parameters for the Beta-prior: $a, b > 0$, where
  - a counts the negative reports (e.g. incoming bug-reports).
  - b counts the positive reports (e.g. patches, successfully blocked intrusion attempts, etc.)

A Bayesian Belief-Model II

Three observations to substantiate our choice $p := \mathbb{E}\{\text{Beta}(a, b)\}$:

1. $p \to 0$ as the number a of negative updates approaches infinity while b remains constant (strongest suspiciousness)
2. $p \to 1$ as the number b of positive updates approaches infinity while a remains constant (full confidence)
3. $\text{Beta}(1, 1) = \mathcal{U}[0, 1]$ models a non-informative prior.





Contents

1 Introduction

2 Preliminaries
- Security as a Game
- Incorporating Trust

3 The Framework
- Belief Models and Bayesian Updating
- Results

4 Conclusion and Outlook

The complete model

Our forecasting system uses the following building-blocks:

component	**modeled by**	**quantified through**
security system	game-matrix $A(p)$	saddle-point value $v(A)$
parameter p	Beta-prior $\mathrm{Beta}(a, b)$	$p = \mathbb{E}(\mathrm{Beta}(a, b)) = \frac{a}{a+b}$

where

- a: is the number of negative updates,
- b: is the number of positive updates.

Updating our beliefs (the Bayesian way)

What we have: a prior belief model Beta(a, b) and new evidence
(a new patch, a bug report, etc.)

What we need: the posterior model:
Beta$(a, b|$most recent information$)$

How we do it: if the latest update is...

- ...positive, then the posterior belief is

$$\text{Beta}(a, b + 1).$$

- ...negative, then the posterior belief is

$$\text{Beta}(a + 1, b).$$

what could be simpler?

Forecasting with the model

Let us put together the pieces:

- We know that

$$\Pr[\text{successful attack}] \leq 1 - v(A(p)) =: \text{risk}(A(p))$$

- We have a belief model $p \sim \text{Beta}(a, b)$ that we can efficiently adjust towards more or less trust in our system based on what updates come in.
- We can ask for
  1. how much longer to wait until the first negative update arrives?
  2. how much more negative updates until the chance of an attack becomes unacceptably high?

Time to wait for the first negative update

If updates happen independently, and our belief model is $p \sim \text{Beta}(a, b)$, then the time to wait for the first negative event to happen (i.e. a significant security incident) is geometrically distributed with parameter p. The expected time to wait is therefore found as

$$N_1 = \mathbb{E}(\text{Geo}(p)) \qquad \text{with} \qquad p \sim \text{Beta}(a, b),$$

and comes to

$$N_1 = \frac{b}{a-1}.$$

This is the expected number of updates before a negative update will come in. The actual time to wait is easily calculated using the average update frequency.

Time to wait until using the system becomes too risky I

Let's say that we are willing to accept a certain small chance of failure, say

$$\Pr[\text{successful attack}] \leq \varepsilon.$$

We know that $\Pr[\text{successful attack}] \leq \text{risk}(A(p))$ and $p \sim \text{Beta}(a, b)$, so that we can ask for the smallest parameter a such that

$$\text{risk}(A(\mathbb{E}(p))) = \text{risk}\left[A\left(\frac{a}{a+b}\right)\right] \geq \varepsilon,$$

in which case the possible that the adversary attacks with a chance of at least $\varepsilon > 0$. The smallest a is easily found by trial-and-error.

Time to wait until using the system becomes too risky II

Let us put

$$r_0 := \min_n \left\{ \text{risk} \left[A \left(\frac{a+n}{a+n+b} \right) \right] \geq \varepsilon \right\},$$

then r_0 is the minimal number of negative updates required to make $1 - v(A(p)) \geq \varepsilon$. The negative binomial distribution with parameters r_0 and p tells how many updates (in total) we need until this minimal number r_0 is observed. As before, we have

$$N_{\text{ex}} = \mathbb{E}(\text{NB}(r_0, p)) \qquad \text{with} \qquad p \sim \text{Beta}(a, b),$$

giving

$$N_{\text{ex}} = r_0 \cdot \frac{b}{a-1} = r_0 \cdot N_1.$$

Example (revisited) I

Let us assume $a = 3, b = 200$, i.e. $p \sim \text{Beta}(3, 200)$ with $\mathbb{E}(p) = 0.9804$ modeling our trust. The value N_1 is

$$N_1 = \frac{200}{3-1} = 100,$$

telling that no less than hundred negative updates will shake our confidence (towards less than $p = 0.5$), based on the previous experience of 200 positive and only 3 negative reports.

Attention

The chosen numbers are artificial, and serve mere illustration purposes. They are not meant to reflect reality!

Example (revisited) II

The time to wait for the chance of an attack to exceed, say 5%, is found by first searching for

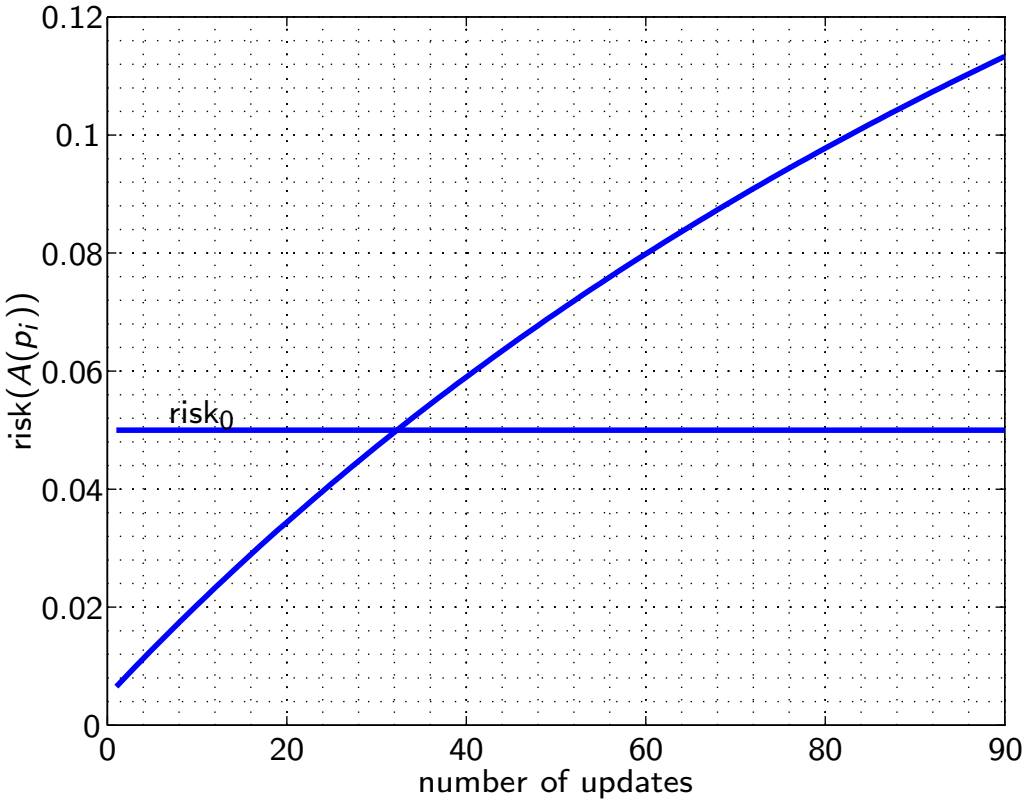
$$r_0 := \min_i \left\{ \text{risk} \left[A \left(\frac{a}{a+i+b} \right) \right] \geq 0.05 \right\} = 33,$$

and then calculating $r_0 \cdot N_1 = 3300$, which is the number of updates until the risk becomes unacceptably large.

Attention

The chosen numbers are artificial, and serve mere illustration purposes. They are not meant to reflect reality!

Example (revisited) III



Contents

1. Introduction
2. Preliminaries
   - Security as a Game
   - Incorporating Trust
3. The Framework
   - Belief Models and Bayesian Updating
   - Results
4. Conclusion and Outlook

Open Problems and Issues I

The model does have some shortcomings, such as

- It cannot monitor itself
- Its practicability remains to be investigated in field-trials
- Automated extraction of relevant information from RSS feeds or other sources is a problem of data-mining and information-retrieval, and as such highly nontrivial on its own.

The model and approach can be extended in various ways, such as

- using other belief-models than the Beta-prior
- using models for multiple belief-parameters

Thank you for listening!

Questions?

Stefan Rass

Department of Applied Informatics, System Security Group,
Klagenfurt University,
Universitätsstrasse 65-67, 9020 Klagenfurt, Austria
stefan.rass@uni-klu.ac.at