A general definition of malware - Examples

©2011 by Roland Fuchsberger
July 6, 2011

Abstract

This paper tries to build up examples to *a general definition of malware* proposed by Simon Kramer and Julian C. Bradfield [1]. All the definitions in this paper are from that article. The goal is to help the reader understand the formal definitions by applying them to intuitive examples.

1 Introduction

The aim of the general definition of malware is to classify interacting software-components of a software-systems into the following categories:

- Malware: software causing incorrectness
- Benware: from benign software - opposite of malware
    - Antimalware: "antibodies" against malware
    - Medware: medical software - software repairing incorrect software

This approach is relaying on a definition of correctness and is building up definitions for these categories by a modal fixpoint logic.

2 Basic Concepts

To identify software that is causing incorrectness we need 2 basic concepts: First we need to introduce a predicate correct(s), to identify correct working software components and not correct working systems. Secondly, we need to model the interaction between components in a software system. Therefore we take the concept of functional application

3 Hammer-Example

One real-world example would be a simple hammer that is smashing a window by a hit. To model this example we introduce a set of systems ϑ, the predicate correct(s), where s $\in \vartheta$, and the functional application $\subseteq \vartheta \times \vartheta \to \vartheta$.

$$\vartheta := \{\text{hammer}, \text{tinyHammer}, \text{twiceTinyhammer}, \text{window}, \text{smashedWindow}\}$$

$$\text{correct}(s) := s \in \{\text{hammer}, \text{tinyHammer}, \text{twiceTinyhammer}, \text{window}\}$$

$$\text{hammer}(\text{window}) = \text{smashedWindow}$$

$$\text{tinyHammer}(\text{tinyHammer}) = \text{twiceTinyHammer}$$

$$\text{twiceTinyHammer}(\text{window}) = \text{smashedWindow}$$

Now we have defined our system, the possible interactions and the correct components. The only possible interaction is the application of the *hammer* to the *window* (by a hit on the window). The outcome is a *smashedWindow*. Notice that the correct *window* becomes a not correct *smashedWindow* by appying the *hammer* to it. The application of *tinyHammer* to *tinyHammer* is a way to model a *tinyHammer* that needs two hits to smash a window.

$$(\text{tinyHammer}(\text{tinyHammer}))\text{window} = \text{smashedWindow}$$

4 Definition of damaging/repairing software

The next step to a general definition of malware, benware, medware and anti-malware is to define damaging and repairing software.

4.1 Damaging software

Damaging a software is the causation of incorrectness of a priori correct software via functional application. Formaly,

s damages s'	:iff	**correct**(s') and not **correct**$(s(s'))$
s damages^0 s'	:iff	s damages s'
s damages^{n+1} s'	:iff	there is s'' s.t.
		not s'' damages$^\circ$ s' and
		$s(s'')$ damages^n s'
s damages$^\circ$ s'	:iff	$\bigcup_{n\in\mathbb{N}}$ s damages^n s'

s damages$^\circ s'$ expresses damaging a software s' in any abitrary steps.

4.1.1 Hammer-bomb example

In the hammer-example the *tinyHammer* is damaging the window in step 1 (beginning with 0). Therefor *tinyHammer* damages *window* does not hold, whereas *tinyHammer* damages$^\circ$ *window*.
Now think of a detonator, a bomb and a house:

$$\vartheta := \{\text{detonator}, \text{bomb}, \text{bombWithDetonator}, \text{house}, \text{destroyedHouse}\}$$

$$\text{correct}(s) := s \in \{\text{detonator}, \text{bomb}, \text{bombWithDetonator}, \text{house}\}$$
$$\text{detonator}(\text{bomb}) = \text{bombWithDetonator}$$
$$\text{bombWithDetonator}(\text{house}) = \text{destroyedHouse}$$

Notice, that *detonator* damages$^\circ$ *house* evaluates to true, but *bomb* damages$^\circ$ *house* to false. This represents that a *bomb* without a *detonator* does not damage anything. But a *detonator* in an environment where a *bomb* is present can destroy a *house*.

4.2 Repairing software

Repairing software is the contrary to damaging - not the contradictory (via negation).

s repairs s'	:iff	not **correct**(s') and **correct**$(s(s'))$
s repairs^0 s'	:iff	s repairs s'
s repairs^{n+1} s'	:iff	there is s'' s.t.
		not s'' repairs$^\circ$ s' and
		$s(s'')$ repairs^n s'
s repairs$^\circ$ s'	:iff	$\bigcup_{n\in\mathbb{N}}$ s repairs^n s'

5 Definition of Malware Logic

Excerpt from [1].
Let $\mathcal{M}$ designate a countable set of propositional variables M, and let

$$\Phi \ni \phi ::= M \mid \neg\phi \mid \phi \wedge \phi' \mid \forall\mathbf{D}(\phi) \mid \forall\mathbf{R}(\phi) \mid \boldsymbol{v}M(\phi)$$

designate the language Φ of MalLog. Then, given the (or only some finite or infinite sub-)class ϑ of software systems s (not just pieces of software), and an interpretation $[\![\cdot]\!] : \mathcal{M} \to 2^{\vartheta}$ of propositional variables, the interpretation $\|\cdot\|_{[\![\cdot]\!]} : \Phi \to 2^{\vartheta}$ of MalLog-propositions is:

$$\begin{array}{rcl}
\|M\|_{[\![\cdot]\!]} & := & [\![M]\!] \\
\|\neg\phi\|_{[\![\cdot]\!]} & := & \vartheta \setminus \|\phi\|_{[\![\cdot]\!]} \\
\|\phi \wedge \phi'\|_{[\![\cdot]\!]} & := & \|\phi\|_{[\![\cdot]\!]} \cap \|\phi'\|_{[\![\cdot]\!]} \\
\|\forall\mathbf{D}(\phi)\|_{[\![\cdot]\!]} & := & \{s \in \vartheta \| \text{ for all } s', \text{ if } s \text{ damages}^{\circ}\ s' \\
 & & \qquad\qquad \text{then } s' \in \|\phi\|_{[\![\cdot]\!]}\} \\
\|\forall\mathbf{R}(\phi)\|_{[\![\cdot]\!]} & := & \{s \in \vartheta \| \text{ for all } s', \text{ if } s \text{ repairs}^{\circ}\ s' \\
 & & \qquad\qquad \text{then } s' \in \|\phi\|_{[\![\cdot]\!]}\} \\
\|vM(\phi)\|_{[\![\cdot]\!]} & := & \bigcup\{S \subseteq \vartheta | S \subseteq \|\phi\|_{[\![\cdot]\!]\ [M \to S]}\},
\end{array}$$

where $[\![\cdot]\!]_{[M \mapsto S]}$ maps M to S and otherwise agrees with $[\![\cdot]\!]$.
Further, $\phi \vee \phi' := \neg(\neg\phi \wedge \neg\phi'), \top := \phi \vee \neg\phi, \bot := \neg\top, \phi \to \phi' := \neg\phi \vee \phi', \phi \leftrightarrow \phi' := (\phi \to \phi') \wedge (\phi' \to \phi), \exists\mathbf{D}(\phi) := \neg\forall\mathbf{D}(\neg\phi), \exists\mathbf{R}(\phi) := \neg\forall\mathbf{R}(\neg\phi)$, and, notably, $\mu M(\phi) := \neg vM(\neg\phi(\neg M))$. Finally,

- $s \in \vartheta$ and $\phi \in \Phi, s \models \phi$:iff $s \in \|\phi\|_{[\![\cdot]\!]}$
- for all $\phi \in \Phi, \models \phi$:iff for all $s \in \vartheta, s \models \phi$
- for all $\phi, \phi' \in \Phi$,
    - $\phi \Rightarrow \phi'$:iff for all $s \in \vartheta$, if $s \models \phi$ then $s \models \phi'$
    - $\phi \Leftrightarrow \phi'$:iff $\phi \Rightarrow \phi'$ and $\phi' \Rightarrow \phi$.

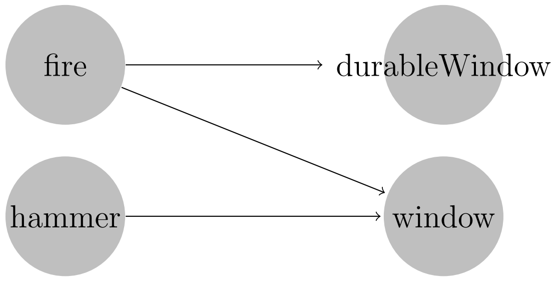
We pronounce $\forall\mathbf{D}(\phi)$ as "necessarily through damage that ϕ", $\forall\mathbf{R}(\phi)$ as "necessarily through repair that ϕ", $\exists\mathbf{D}(\phi)$ as "possibly through damage that ϕ" and $\exists\mathbf{R}(\phi)$ as "possibly through repair that ϕ", $vM(\phi)$ as "the greatest fixpoint of the property M such that ϕ", $\mu M(\phi)$ as "the least fixpoint of the property M such that ϕ", $s \models \phi$ as "s satisfies ϕ", $\models \phi$ as "ϕ is valid", $\phi \Rightarrow \phi'$ as ϕ' is a logical consequence of ϕ", and $\phi \Leftrightarrow \phi'$ as "ϕ is logically equivalent to ϕ'".

5.1 MalLog applied to a modified Hammer-Example

To become a feeling for the definitions, we at first look at the modal operators and secondly at the fixpoint operators by evaluating formulas. Therefor we first modify a little bit the hammer-example: Think of a hammer, a fire, a regular window and an extreme durable window. The extreme durable window could only be destroyed by the fire, but not by the hammer. Formally,

$$\begin{aligned}\vartheta :=&\quad \{\text{fire}, \text{hammer}, \text{window}, \text{durableWindow}, \text{destroyedWindow}\}\\ \text{correct}(s) :=&\quad s \in \{\text{fire}, \text{hammer}, \text{window}, \text{durableWindow}\}\\ \text{fire}(\text{window}) =&\quad \text{destroyedWindow}\\ \text{hammer}(\text{window}) =&\quad \text{destroyedWindow}\\ \text{fire}(\text{durableWindow}) =&\quad \text{destroyedWindow}\end{aligned}$$

5.1.1 The damages$^\circ$ relation



5.1.2 The interpretation $\|\ \|_{[\![\cdot]\!]}$

For a correct typeset of the formulas we need to specify a set of propositional variables $\mathcal{M}$ and a relation $\mathcal{M} \mapsto 2^{\vartheta}$. This could be usefully for example to unite several software components to on subsystem. For example we could define a propositional variable *_badTools* and map *_badTools* to *fire* and *hammer*. To satisfy the type-system and for simplicity we define for every $s \in \vartheta$ one propositional variable M and map them together. Formally,

$$\begin{aligned}\mathcal{M} :=&\quad \{\text{\_fire, \_hammer, \_badTools, \_window, \_durableWindow, \_destroyedWindow}\}\\ [\![\cdot]\!] :=&\quad \{(\text{\_fire, fire}), (\text{\_hammer, hammer}), (\text{\_badTools, hammer}), (\text{\_badTools, fire}),\\ &\quad (\text{\_window, window}), (\text{\_durableWindow, durableWindow}),\\ &\quad (\text{\_destroyedWindow, destroyedWindow})\}.\end{aligned}$$

For Instance, $[\![\text{\_badTools}]\!] = \{\text{hammer, fire}\}$.

5.1.3 The modal operator $\forall\mathbf{D}$

$$\forall\mathbf{D}(\emptyset) = \{\text{durableWindow}, \text{window}, \text{destroyedWindow}\}$$

For a better understanding take a look at the damages° relation: Notice that $\forall\mathbf{D}(\emptyset)$ always evaluates to a set containing those elements that (informally) do not stand on the left side of the relation.
Here are some additional examples:

$$\forall\mathbf{D}(\{\_\text{window}\}) = \{\text{durableWindow}, \text{window}, \text{destroyedWindow}, \text{hammer}\}$$

$$\forall\mathbf{D}(\{\_\text{window}, \_\text{durableWindow}\}) = \{\text{durableWindow}, \text{window}, \text{destroyedWindow}, \text{hammer}, , \text{fire}\}$$

5.1.4 The greatest-fixpoint operator $vM(\phi)$

Let $\vartheta := \{a, b, c\}$ and $R \subseteq \vartheta \times \vartheta$ such that:

$$\begin{tikzcd}
a \arrow[r] \arrow[dr] & a \\
b \arrow[r] & b \\
c \arrow[ur] &
\end{tikzcd}$$

Let $f(X) := \{m \in \vartheta \mid \exists_{x \in X} : xRm\}$.
For Instance, $f(\{a\}) = \{a, b\}$ and $f(\{b\}) = \{b\}$.

$$\text{X is a fixpoint of } f(X) \text{ :iff } f(X) = X.$$

In our example, we can find two fixpoints: {b} (because $f(\{b\}) = \{b\}$) and {a, b} (because $f(\{a, b\}) = \{a, b\}$). The greatest fixpoint is the one fixpoint with the greatest cardinality. Formally, $vM(f(M)) = \{a, b\}$.

6 Formal definition of malware

$$\text{mal}(s) \text{ :iff } s \models vM(\exists\mathbf{D}(\forall\mathbf{D}(M))$$

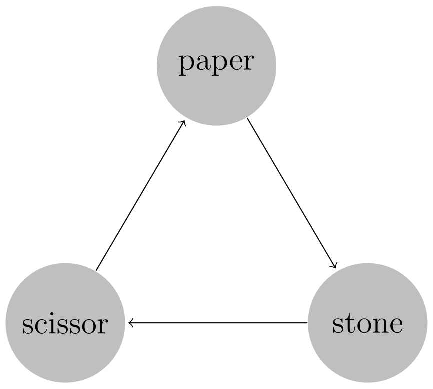
By expanding the definitions:

$$\text{mal}(s) \text{ :iff } s \models vM(\{s \in \vartheta \mid \exists_{s'} : (s \text{ damages}^\circ s' \ \wedge \forall_{s''} : s'\text{damages}^\circ s'' \rightarrow s'' \in M)\})$$

With this definition we are finally able to identify the malware in our examples. The reader is invited to apply this definition to the previous examples. In this section we apply the definition to two more interesting examples. As the definition for malware just relies on the damages° relation, it is also possible to directly define this relations and the set of software systems ϑ.

6.1 Paper-scissor-stone

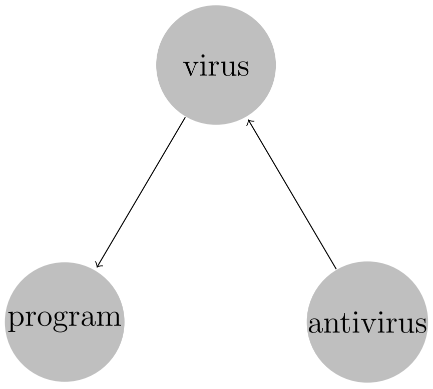
ϑ :={paper, scissor, stone}
s damages$^\circ s'$:=



Applying the definition, the greatest fixpoint of our system is {paper, scissor, stone }. This means, that everything in this system is a malware.

6.2 Program-Virus-Antivirus

Imagine a computervirus that is damaging a computerprogram and a antivirus that is damaging that virus.
ϑ := {program, virus, antivirus}
s damages$^\circ s'$:=



Again, applying the definition of malware, we came up that only the virus is a malware. Software that is damaging a malware is a benware.

7 Formal definition of benware, medware and anti-malware

$$\mathrm{ben}(s) \text{ :iff } \quad s \models \mu M(\forall\mathbf{D}(\exists\mathbf{D}(M)))$$

$$\begin{aligned}\mathrm{antimal}(s) \text{ :iff } \quad & s \models \neg\exists\mathbf{D}(\mathrm{BEN}) \text{ and there is } s' \text{ s.t. } \mathrm{mal}(s') \\ & \text{and not } \mathrm{mal}(s(s')), \\ & \text{where } \mathrm{BEN} := \mu M(\forall\mathbf{D}(\exists\mathbf{D}(M)))\end{aligned}$$

$$\mathrm{med}(s) \text{ :iff } \quad s \models \neg\exists\mathbf{D}(\mathrm{BEN}) \wedge \exists\mathbf{R}(\mathrm{BEN})$$

For more details on that definitions, please look them up in the original paper [1].

8 Conclusion

"Three major advantages of our approach are: *generality*, *genericity*, and *safety* - all thanks to abstractness. Our approach is general and safe because it focuses on what malware effects but abstracts from how malware actually and potentially does so. In particular, our approach is *hacker-safe* in the sense that it does not enable hackers to derive recipes for how to actually construct malware. Our approach is generic in the sense that different generations of MalLog (and thus malware classifications) can be obtained by redefinitions of damages° and repairs°." [1]

References

[1] Simon Kramer and Julian Bradfield. A general definition of malware. *Journal in Computer Virology*, 6:105–114, 2010. 10.1007/s11416-009-0137-1.