A General Definition of Malware
(SRM Seminar)

Simon Kramer
(j.w.w. Julian C. Bradfield, U Edinburgh)

University of Luxembourg
Institute of Mathematical Sciences, Chennai

May 17, 2010

Infamous malware examples

Conficker detected in November 2008—still active—can cause a computer under the Windows operating system to become a component of a remote-controlled botnet against the user’s will—on an infected computer, it causes a buffer overflow in which harmful excess code is executed by the operating system—the excess code downloads more code that hijacks the server services of the operating system, in order to update and spread the worm via the network—variant code inhibits also the security services of the operating system and connections to anti-malware websites...affected European military systems...


World-wide malware impact

World-wide malware-induced damage in 2006 = $13.3 \cdot 10^9$

[Computer Economics Inc., 2007]
Motivation, Goal, and Methodology

**Motivation** An open problem [FHZ06]: find a general definition of malware (= malicious software), e.g., botnets, rootkits, Trojan horses, viruses, worms, etc.

**Goal** Obtain a formal solution to the problem.

**Methodology** Formulation of the solution as a single sentence in a computational modal fixpoint logic.

Malware as harmful software

**What is malware?**
- Informally, \( \text{malware} = \text{malicious software} \)
  - Malicious intention is not generally directly observable!
  - How to distinguish unawareness (juvenile hacking, accidental anti-hacking) from malice?
  - Users don’t care: all that matters is (harmful) effect, not (malicious) intention.
  - Malice is immaterial!
  - Psychological “definition”
- Intuitively, \( \text{malware} = \text{harmful software} \)
  - Harmful effect is observable!
  - Scientific definition
Preliminaries

Definition (Damaging software)
A software system \( s \) damages a correct software system \( s' \) by definition if and only if \( s \) (directly or indirectly) causes incorrectness to \( s' \). Formally,

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

Example: Sorting

- **Given:** a program \( s \) for sorting an array \( A \) of \( l \) integers
- **Sought:** a correctness definition for \( s \)
  - Pre := \( A : \text{Array}\{\mathbb{N}\}(2) \)
  - Post := \( \forall (1 \leq i \leq l) \forall (1 \leq j \leq l) (i \leq j \rightarrow A[i] \leq A[j]) \)
  - Is that strong enough?
- **Variations:** add necessary conditions (e.g., exact algorithmic complexity), stipulate proof-carrying code, etc.

Prerequisites

Theorem (Knaster-Tarski fixpoint theorem)
Let \( \langle L, \leq \rangle \) designate a complete lattice\(^1\) and \( f : L \rightarrow L \) a monotonic map\(^2\) on \( L \). Then,

\[
g := \bigvee \{ a \mid a \in L \text{ and } a \leq f(a) \}
\]

is the greatest fixpoint of \( f \), and, dually,

\[
l := \bigwedge \{ a \mid a \in L \text{ and } f(a) \leq a \}
\]

is the least fixpoint of \( f \).

\(^{1}\forall S \text{ (lub)} \) and \( \bigwedge S \text{ (glb)} \) exist for arbitrary \( S \subseteq L \)
\(^{2}\)for all \( a, b \in L, \text{ if } a \leq b \text{ then } f(a) \leq f(b) \)

Definition (Repairing software)
A software system \( s \) repairs an incorrect software system \( s' \) by definition if and only if \( s \) (directly or indirectly) causes correctness to \( s' \). Formally,

- \( s \) repairs \( s' \) :iff \( \text{not correct}(s') \) and \( \text{correct}(s(s')) \)
- \( s \) repairs\( ^0 \) \( s' \) :iff \( s \) repairs \( s' \)
- \( s \) repairs\( ^0+1 \) \( s' \) :iff there is \( s'' \) s.t. not \( s'' \) repairs\( ^0 \) \( s' \) and \( s(s'') \) repairs\( ^0 \) \( s' \)
- \( s \) repairs\( ^0 \) \( s' \) :iff \( \bigcup_{n \in \mathbb{N}} s \) repairs\( ^n \) \( s' \).
Malware Logic

Definition (MalLog)

Let \( \mathcal{M} \) designate a countable set of propositional variables \( M \), and
\[
\Phi := M \mid \neg \phi \mid \phi \land \phi \mid \forall D(\phi) \mid \forall R(\phi) \mid \nu M(\phi)
\]
the language \( \Phi \) of MalLog where all free occurrences of \( M \) in \( \phi \) of \( \nu M(\phi) \) are assumed to occur within an even number of occurrences of \( \neg \) to guarantee the existence of (greatest) fixpoints (expressed by \( \nu M(\phi) \)) [BS07].

Malware Logic (continued)

Further, \( \phi \lor \phi' := \neg (\neg \phi \land \neg \phi') \), \( \top := \phi \lor \neg \phi \), \( \bot := \neg \top \), \( \phi \Rightarrow \phi' := \neg \phi \lor \phi' \), \( \phi \leftrightarrow \phi' := (\phi \rightarrow \phi') \land (\phi' \rightarrow \phi) \), and
\[
\exists D(\phi) := \neg \forall D(\neg \phi) \\
\exists R(\phi) := \neg \forall R(\neg \phi) \\
\mu M(\phi(M)) := \neg \nu M(\neg \phi(\neg M)).
\]

Finally,
- for all \( \phi \in \Phi \) and \( s \in \mathcal{S} \), \( s \models \phi \) iff \( s \in \llbracket \phi \rrbracket_\mathcal{F} \)
- \( \models \phi \) iff for all \( s \in \mathcal{S} \), \( s \models \phi \)
- for all \( \phi, \phi' \in \Phi \),
  - \( \phi \Rightarrow \phi' \) iff for all \( s \in \mathcal{S} \), if \( s \models \phi \) then \( s \models \phi' \)
  - \( \phi \leftrightarrow \phi' \) iff \( \phi \Rightarrow \phi' \) and \( \phi' \Rightarrow \phi \).

Malware Logic (continued)

Then, given the (or only some sub-) class \( \mathcal{S} \) of software systems (not just pieces of software) \( s \) and an interpretation \( \llbracket - \rrbracket : \mathcal{M} \rightarrow 2^\mathcal{S} \) of propositional variables, the interpretation \( \llbracket - \rrbracket_\mathcal{F} : \Phi \rightarrow 2^\mathcal{S} \) of MalLog-propositions is:
\[
\llbracket M \rrbracket_\mathcal{F} := [M] \\
\llbracket \neg \phi \rrbracket_\mathcal{F} := \mathcal{S} \setminus \llbracket \phi \rrbracket_\mathcal{F} \\
\llbracket \phi \land \phi' \rrbracket_\mathcal{F} := \llbracket \phi \rrbracket_\mathcal{F} \cap \llbracket \phi' \rrbracket_\mathcal{F} \\
\llbracket \forall D(\phi) \rrbracket_\mathcal{F} := \{ s \mid \text{for all } s' \text{, if } s \text{ damages } s' \text{ then } s' \in \llbracket \phi \rrbracket_\mathcal{F} \} \\
\llbracket \forall R(\phi) \rrbracket_\mathcal{F} := \{ s \mid \text{for all } s' \text{, if } s \text{ repairs } s' \text{ then } s' \in \llbracket \phi \rrbracket_\mathcal{F} \} \\
\llbracket \nu M(\phi) \rrbracket_\mathcal{F} := \bigcup \{ S \mid S \subseteq \llbracket \phi \rrbracket_\mathcal{F} \}
\]
where \( [\cdot]_{\mathcal{F} \rightarrow \mathcal{S}} \) maps \( M \) to \( \mathcal{S} \) and otherwise agrees with \( [\cdot] \).

Basic properties of MalLog

Fact

1. \( \models \phi \Rightarrow \phi' \) iff \( \phi \Rightarrow \phi' \) (By expansion of the definitions.)
2. \( \models \phi \leftrightarrow \phi' \) iff \( \phi \leftrightarrow \phi' \)
3. MalLog is a member of the family of \( \mu \)-calculi over the modal system \( K_2 \), which is characterised by the validities of propositional logic and the modal laws
   \( \models \square(\phi \Rightarrow \phi') \Rightarrow (\square \phi \Rightarrow \square \phi') \) and “if \( \models \phi \) then \( \models \square \phi \)
   where \( \square \in \{ \forall D, \forall R \} \).
Basic properties of MalLog (continued)

Corollary

1. If damages\(^2\) and repairs\(^2\) are decidable on a given software systems domain then the satisfiability problem for MalLog, i.e., “Given \(\phi \in \Phi\), is there \(s \in S\) s.t. \(s \models \phi\)?”, (and thus also the model-checking problem, i.e., “Given \(\phi \in \Phi\) and \(s \in S\), is it the case that \(s \models \phi\)?”) is decidable.

2. MalLog is axiomatisable by the following Hilbert-style proof-system:
   - 2.1 the axioms/rules of the modal system \(K\) for each \(\forall D\) and \(\forall R\)
   - 2.2 the axiom \(\phi(M) \rightarrow M(\phi(M))\)
   - 2.3 the rule \(\phi(\phi') \rightarrow \phi'\)

---

An iterative paraphrase

- Everything is malware (better be safe than sorry)
- except for (throw out what is clearly safe) the following systems:

0. non-damaging systems (CP)
1. systems that damage only systems that damage CP (ATF1)
2. systems that damage only systems that damage ATF1 (ATF2)
3. systems that damage only systems that damage ATF2 (ATF3)
4. etc.
An iterative paraphrase

- Nothing is benware (again, better be safe than sorry)
- except for (throw in what is clearly safe) the following systems:

  0. non-damaging systems (CP)
  1. systems that damage only systems that damage CP (ATF1)
  2. systems that damage only systems that damage ATF1 (ATF2)
  3. systems that damage only systems that damage ATF2 (ATF3)
  4. etc.

Good&Bad distinction induced by the existence of a population that is (perceived as) non-damaging

**Definition (Anti-malware)**

A software system \( s \) is **anti-malware** by definition if and only if \( s \) damages no benware (safety) and \( s \) neutralises\(^4\) malware (effectiveness). Formally,

\[
\text{antimal}(s) \iff s \models \neg \exists D \text{(BEN)} \text{ and there is } s' \text{ s.t. } \text{mal}(s') \text{ and not } \text{mal}(s')
\]

where \( \text{BEN} := \mu M (\forall D \exists D (M)) \).

---

\(^3\) no friendly fire

\(^4\) Damage is insufficient!
Tasks, Tools, and Techniques for fighting Malware

<table>
<thead>
<tr>
<th>Task</th>
<th>Tool</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>detection</td>
<td>satisfaction relation</td>
<td>Model Checking</td>
</tr>
<tr>
<td>comparison</td>
<td>language &amp; bisimulation equivalence</td>
<td>Equivalence Checking</td>
</tr>
<tr>
<td>classification</td>
<td>characteristic formulas</td>
<td>MC, EC</td>
</tr>
</tbody>
</table>

Malware Comparison (declarative)

Definition (Language equivalence)
For all $s_1, s_2 \in \mathcal{S}$,

1. $s_1 \equiv_\Phi s_2$ :iff for all $\phi \in \Phi$, if $s_1 \models \phi$ then $s_2 \models \phi$
2. $s_1 \equiv_\phi s_2$ :iff $s_1 \equiv_\Phi s_2$ and $s_2 \equiv_\Phi s_1$.

Malware Classification

Definition (Characteristic formula)
Let $S \subseteq \mathcal{S}$, $s \in \mathcal{S}$, $D(S, s) := \{ s' \mid s' \in S \text{ and } s \text{ damages } s' \}$, $R(S, s) := \{ s' \mid s' \in S \text{ and } s \text{ repairs } s' \}$, and $M_\Phi \in \mathcal{M}$. Then, the characteristic formula $\chi(s, \mathcal{S})$ of the software system $s$ w.r.t. $S$ is the solution of the equation system

$$M_\Phi \models \forall D(s' \in D(S, s)) M_{s'} \land \forall R(s' \in R(S, s)) M_{s'} \land \left[ \bigwedge s' \in D(S, s) \exists D(M_{s'}) \land \left[ \bigwedge s' \in R(S, s) \exists R(M_{s'}) \right] \right],$$

( where $\bigwedge \equiv \land$ and $\exists \equiv \lor$ ) obtained [BS07] by translating each equation $M' \equiv \psi(s)$ into a formula $\nu M'(\psi(s))$ and recursively substituting these formulae for the corresponding free variables in the first formula $\nu M(\psi(S))$.  

A General Definition of Malware

Simon Kramer (j.w.w. Julian C. Bradfield, U Edinburgh)
Characterisation result

Theorem

For all \( s, s' \in S \),

\[ s \equiv \Phi \iff s \approx s' \iff s \models \chi(s', S). \]

Assessment

Our approach:

1. malware-versus-benware arms race confined to formal systems engineering
2. malware detection \( \rightarrow \) automated systems verification
3. system security \( \rightarrow \) system correctness
4. generic (predicate correct is a plug-in)
5. **hacker-safe:** no recipe for malware construction derivable

Related work

About viruses only, not hacker-safe (constructive):

1. Adleman: Gödel-numberings [Adl88]
2. Cohen: Turing-machines [Coh87]
3. Bonfante et al.: Kleene Recursion Theorem [BKM06]

Future work

Refinements:

- add **time** (temporal modalities): malware evolution
- add **measure**: degrees of damage, malware cost
Bibliography


Contact

Email: simon.kramer@epfl.ch

Homepage: http://www.simon-kramer.ch/