The Hyperion System: Cybersecurity through Software Behavior Computation

“What does your software do?”

Rick Linger, Stacy Prowell, Stephen Lindberg
Cyber Security and Information Intelligence Research Group
Oak Ridge National Laboratory
202-701-6257, lingerr@ornl.gov
Customer Need: Assure Software Function and Security
The Value of Assured Software

Cyberspace Sciences & Information Intelligence (CSIIR) Group

- Software vulnerability announcements cost vendors $860M in market value.¹
- Average cost of security incident: $300K among surveyed firms.²


source: Verizon Data Breach Investigation Report 2012
Our adversaries are students of our software

- Study and execute our code
- Accumulate and share knowledge
- Know more than we do
- Discover and exploit behavior unknown to us

Behavior discovery is a big business
- Most software is out of intellectual control
  - Full behavior is not known by us

- Attack and defense
  - All about understanding and exploiting behavior

- Software is complex
  - That’s no excuse – our adversaries figure it out
• Unknown behavior means unknown ...
  – Errors
  – Vulnerabilities
  – Security properties
  – Risks
  – Opportunities for adversaries

• At the core of the cyber security problem
Today’s Response

- Set security standards
- Install security tools
- Scan code using signatures
- Train security personnel
- Conduct risk management
- Analyze supply chains

- All good, but we’re losing the battle
Technical Approach: Compute the Behavior of Software
The Idea of Behavior Computation

• New technology for understanding software (malware)

• What is computed behavior?
  – What a program does in all circumstances of use
  – The “as-built” specification

• What are key properties?
  – Operates on program semantics, not syntax
  – Analyzes binaries to approach ground truth
  – Mathematical precision, no heuristics
  – Doesn’t look for things in code
  – Zero day makes no difference – just more behavior
Key Concept: Program Structuring

Unstructured (obfuscated) spaghetti logic:

if x > y goto A
goto B
C: t := x;
x := y;
y := t
goto D
A: t := x;
x := y;
y := t;
goto C
B: x := x + y;
y := x - y;
x := x - y
C: goto D
D:

Transformation to structured form:

if x > y then
t := x;
x := y;
y := t
endif;
if x > y goto A
go to B
C: t := x;
x := y;
y := t
go to D
A: t := x;
x := y;
y := t
go to C
B: x := x + y;
y := x - y;
x := x - y
go to C
D:

Structure Theorem

Defines transformation from complex logic into function-equivalent structured form expressed in sequence, ifthenelse, and whiledo control structures.

Coelesces and aggregates related code into a systematic structure
Key Concept: Behavior Computation

Program:
```plaintext
do
  x := x + y;
  y := x - y;
  x := x - y
enddo
```

Computation:

<table>
<thead>
<tr>
<th>assignment</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x := x + y</td>
<td>x1 = x0 + y0</td>
<td>y1 = y0</td>
</tr>
<tr>
<td>2 y := x - y</td>
<td>x2 = x1</td>
<td>y2 = x1 - y1</td>
</tr>
<tr>
<td>3 x := x - y</td>
<td>x3 = x2 - y2</td>
<td>y3 = y2</td>
</tr>
</tbody>
</table>

Derivations:

\[
\begin{align*}
  x3 &= x2 - y2 \\
  &= x1 - (x1 - y1) \\
  &= y1 \\
  &= y0
\end{align*}
\]

\[
\begin{align*}
  y3 &= y2 \\
  &= x1 - y1 \\
  &= x0 + y0 - y0 \\
  &= x0
\end{align*}
\]

Computed behavior:

true \(\Rightarrow\)
```plaintext
  x := y
  y := x
```

(swaps values of x and y)

(Conditional concurrent assignment (CCA))

Correctness Theorem

Defines mathematical transformations from procedural logic expressed in sequence, ifthenelse, and whiledo forms into behaviorally-equivalent, functional forms.

Transforms procedural logic into non-procedural as-built specification
The Behavior Computation Process

Input: software/malware binaries

Transform instructions to semantic form

Instruction semantics repository

Defines functional effect of instructions

Transform code to structured form

Behavior expressed as Conditional Current Assignments (CCAs)

Compute code behavior

Behavior Specification Unit (BSU) repository

Defines precise behavior abstractions

Analyze behavior functionality

Behavior abstracted to Behavior Specification Units (BSUs)

Output: program behavior and analysis
An Example Behavior Computation

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Input code

Structured code

Computed behavior expressed as a Conditional Concurrent Assignment (CCA)
Customer Benefits: Understanding What Software Does
What is the Value Proposition?

- Understand behavior at machine speeds
  - Intellectual control over software
- Cheap, repeated validation
  - Confidence in mission readiness
- Know more than our adversaries
  - We understand behavior before they do
What are the Markets?

• Malware detection and analysis
• Vulnerability detection
• Mobile device validation
• Rigorous software development
• Supply chain, anti-tamper analysis
• Forensic investigation
• Hardware analysis
• ...
Competition: The Behavior Computation Advantage
How is Software Analyzed Today?

- **Human review:**
  - Expensive, fallible

- **Execution testing**
  - Expensive, inconclusive

- **Syntactic scanning**
  - Cheap, inconclusive

- No good means for understanding full behavior
Comparing Technologies

Population of executions:
Dots are test cases, area not covered is untested executions in the population.

Population of instructions:
Dots are recognized signatures, squares are problems that do not have signatures or are obfuscated.

Population of behaviors:
Disjoint partitions are behavior cases that cover the entire population.
Status: Evolving the Technology

- Next-generation Hyperion system under development
  - More powerful semantic processing
  - HPC potential for parallel computation
  - Addressing scale-up for larger programs
  - Currently Intel x86, other languages can be supported
  - Can customize for sponsor requirements

Demonstration:
Behavior computation for malicious code that attacked ORNL