Network Vulnerability Measurement – A Novel Approach

Lee Badger
Tim Grance
Karen Scarfone

Dec. 3, 2009
# Measurement Scales and Bold Assertions

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio</td>
<td>A zero point exists where none of the attribute is present</td>
</tr>
<tr>
<td>Interval</td>
<td>Magnitudes of differences between values are meaningful</td>
</tr>
<tr>
<td>Ordinal</td>
<td>Values have &lt;, &gt;, and = relationships</td>
</tr>
<tr>
<td>Nominal</td>
<td>Values have no firm numerical ordering, but = scale values mean equal attribute values</td>
</tr>
</tbody>
</table>

Credit: S.S. Stevens, Wikipedia
Measurement Scales and Bold Assertions

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio</td>
<td>A zero point exists where none of the attribute is present</td>
</tr>
<tr>
<td>Interval</td>
<td>Magnitudes of differences between values are meaningful</td>
</tr>
<tr>
<td>Ordinal</td>
<td>Values have &lt;, &gt;, and = relationships</td>
</tr>
<tr>
<td>Nominal</td>
<td>Values have no firm numerical ordering, but = scale values mean equal attribute values</td>
</tr>
</tbody>
</table>

Recall: (simplified)

Informal Scale/Metric Assignment

Legend:
- Sample metric
- Logical predicates on state.
- Best-effort scores from a scoring system.
- Information Flow
- Operations per second
- Application specific
- Ratings scores
- Ratings scores
- Ratings scores

Information Flow

Operations per second

Logical predicates on state.

Ratings scores

Ratings scores

Ratings scores

Measurement in the very Small (< 10,000 SLOC)

Measurement in the Large (millions of SLOC)

Credit: S.S. Stevens, Wikipedia
# 3 NIST Scoring Systems

Available at:  [http://csrc.nist.gov/publications](http://csrc.nist.gov/publications)

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
<th>NIST #</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVSS</td>
<td>The Common Vulnerability Scoring System…</td>
<td>IR 7435</td>
<td>Method to express the characteristics and impacts of software flaw vulnerabilities. The scoring basis for the National Vulnerability Database, maintained at NIST (nvd.nist.gov).</td>
</tr>
<tr>
<td>CCSS</td>
<td>The Common Configuration Scoring System (DRAFT)</td>
<td>IR 7502</td>
<td>Method to measure the vulnerability of security settings of a system.</td>
</tr>
<tr>
<td>CMSS</td>
<td>The Common Misuse Scoring System… (DRAFT)</td>
<td>IR 7517</td>
<td>Method to measure the vulnerability of the <strong>intentional functions</strong> of a system. Measure trust assumptions.</td>
</tr>
</tbody>
</table>
The Common Misuse Scoring System (CMSS)

**Base metrics**
- **Access Vector**
  - (L) Local
  - (A) Adjacent Network
  - (N) Network
- **Authentication**
  - (M) multiple
  - (S) single
  - (N) none
- **Access Complexity**
  - (H) high
  - (M) medium
  - (L) low
- **Exploitability**
  - (N) none
  - (P) partial
  - (C) complete
- **Confidentiality**
  - (N) none
  - (P) partial
  - (C) complete
- **Integrity**
  - (N) none
  - (P) partial
  - (C) complete
- **Availability**
  - (N) none
  - (P) partial
  - (C) complete

**Temporal metrics**
- **General Exploit Level**
  - (N) None
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined
- **General Remediation Level**
  - (N) None
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined
- **Confidentiality/Integrity/Availability Requirements**
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined

**Environmental metrics**
- **Local Vulnerability Prevalence**
  - (N) None
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined
- **Perceived Target Value**
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined
- **Local Remediation Level**
  - (N) None
  - (L) Low
  - (M) Medium
  - (H) High
  - (ND) not defined

**Exploitability**
- **Collateral Damage Potential**
  - (N) None
  - (L) Low
  - (LM) Low-Medium
  - (MH) Medium-High
  - (H) High
  - (ND) not defined
- **Impacts**
  - **Environmental equation**
  - **Temporal equation**
  - **General Exploit Level**
  - **General Remediation Level**

Thanks to Peter Mell for clarifying discussions.
Metrics Idea in a Nutshell

Today: unstructured system
unknown information flows

Tomorrow: structured system
known information flows

Question:
Is Sensitive Info
Exfiltrated or Corrupted?

Add mediation/observation layers: get restricted topology
Analyze source code of new layers to get constraints
Formalize attacker goal as an attack graph

Solve goal + constraints, if possible
FALSE means attack not feasible
OTHERWISE, get constraints attacker must satisfy.
Augment system to constrain runtime behavior, increase observability

Many hooking techniques are now available:
- System Call Wrappers
- Library Wrappers
- Protocol Wrappers
- Object Wrappers
- Instruction Wrappers
- File System Wrappers
- Device Wrappers
- Translation-based Wrappers

1. I.e., balkanize the system using wrappers, or the sandboxing built into some operating systems

2. Balance with risk of incompatibility
Use Attack Graphs

An attack graph is an abstraction of a network (system).
A node represents network configuration and attacker capabilities held (e.g., root access on host n)
An edge represents an action taken to move to an attacker goal.

action IIS-buffer-overflow is
intruder preconditions
plvl(S) >= user
plvl(T) < root

network preconditions
w3svc_T
R(S,T,80)

intruder effects
plvl(T) := root

network effects!
w3svc_T

Host (Source) → Host (Target)

1. Nodes X services X known-vulnerabilities
many possible scenarios

2. Analysis limited to known vulnerabilities
(e.g., CVE records)

We wish to handle unknown vulnerabilities, too…

## Traditional Attack Graphs vs Our Approach

<table>
<thead>
<tr>
<th>Traditional</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>(Hopefully) More Precise Results For (more Restricted) Systems.</td>
</tr>
<tr>
<td>Approximate Results</td>
<td></td>
</tr>
<tr>
<td>For Current (more Complex) Systems</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>State:</strong></td>
<td></td>
</tr>
<tr>
<td>Connectivity</td>
<td>Connectivity</td>
</tr>
<tr>
<td>Host Vulnerabilities</td>
<td>Source code, for selected services</td>
</tr>
<tr>
<td>Attacker privileges/goals</td>
<td>Attacker privileges/goals</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5 hosts $\times$ 8 exploits $\rightarrow$ 5,948 nodes</td>
<td>3 hosts $\times$ code $\times$ (1 or a few) services</td>
</tr>
<tr>
<td>Monotonicity assumption</td>
<td>SAT problem: size still unknown</td>
</tr>
</tbody>
</table>


Symbolic Execution: Brief Synopsis

Concept

- **x** is read from the environment
- **Variable x unchecked**
- Branch on predicate **P**
- \((\neg P) \land \text{x is unchecked}\)
- \(P \land \text{x is unchecked}\)
- Branch on **check(x)**

Use

- **Legacy app**
- **Legacy middleware**
- **mediation layer**
- **Legacy network**

- **Packet in**
- **Packet out**
- **packet constraints**

- **Mediation Layer could be:** a virtual machine
  an app proxy
  an app wrapper
  ...

- Mechanically generated

Credit: this legacy idea is in the Stanford Saturn system: see [http://saturn.stanford.edu](http://saturn.stanford.edu), and others.
Focus Analysis using Slices


Slice layer with respect to selected output statements (e.g., sendmsg())

Instead of generating all statements in the slice, generate boolean expressions at output statements.

Predicates on: values per o_i, ordering, relations on o_i, bindings to external events (e.g., authentication).

Specify upstream outputs to be “trustable” by downstream inputs.

Abstract system trace: o_1, o_2, o_3, .....
Nuts, Bolts, first Steps

Experimenting with the LLVM compiler infrastructure (www.llvm.org).

And with the LLVM-based CLANG (C-family) compiler (clang.llvm.org).

Static Single Assignment gives use/def chains helpful for slicing and symbolic analysis.

Pass management framework makes it pretty easy to develop the analysis as a compiler analysis/ transformation pass.

First service chosen: rsh:

<table>
<thead>
<tr>
<th>Library</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rshd</td>
<td>393 sloc</td>
</tr>
<tr>
<td>Libutil</td>
<td>5,365</td>
</tr>
<tr>
<td>Libpam</td>
<td>5,383</td>
</tr>
<tr>
<td>Libc</td>
<td>175,367</td>
</tr>
</tbody>
</table>

“easy”

Less “easy”
Backup
Informal Example

Possible Scenario:
1) attacker triggers buffer overflow in IIS, gaining control of IIS
2) captured IIS sends malicious JPEG to host A, gaining control of A
3) host A sends “rcp” command to host B
4) host B “trusts” host A and returns sensitive file
5) host A sends file to the captured IIS
6) captured IIS tunnels file through firewall to attacker

Analysis
1) attacker’s goal is to retrieve the data, i.e., “there exists a sequence of write(src,dest) operations such that write(sensitive-data, d1), write(d1, d2), … write(dn, attacker)” must be satisfiable for the attacker to succeed
2) P is: write(sensitive-data, x) is in the trace only if x is authenticated
3) Q is: if a controlled endpoint reads a complex object, its authentication is subsequently “none”
4) R is: an object passed via HTTP is tagged by its complexity score
Candidate Inputs and Outputs for Measurement

Inputs:

- **Asset Inventory**: List of resources needing protection.
- **Network Topology**: A topological model of the target system showing boundary controllers and where new layers can be transparently inserted to restrict attack paths.
- **Attacker Victory Conditions**: A first-order predicate calculus statement defining attacker victory.
- **Assumed Attacker Starting Positions**: External network access only vs intruder code launched from USB devices vs rogue laptops.

Outputs:

- **Attacker’s Required Constraint Set**: Conjunctive normal form boolean expression, Possibly with a proof of unsatisfiability (it’s FALSE).
  - Conjunctive normal form constraint set:- it can be Large (e.g., STP has solved a expressions with **2 million** variables for software analysis.
- **Analysis limitations**: Set of simplifying assumptions.

Note: STP is Simple Theorem Prover; see Vijay Ganesh and David Dill, "A Decision Procedure for Bit-Vectors and Arrays"