PatchLink Unix Agent
Verification and Assessment

David Gilliam, John Powell:
Jet Propulsion Lab (JPL)

Matt Bishop: University of California at Davis

Chris Andrew, Sameer Jog:
PatchLink Corporation
Acknowledgement

- This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.
  - The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program lead by the NASA Software IV&V Facility.
  - This activity is managed locally at JPL through the Assurance and Technology Program Office.
- This research was conducted with the cooperation of PatchLink Corporation under a Non-Disclosure Agreement (NDA).
Agenda

- Introduction
- Software Assurance
- Verification Activities
  - Model-Based Verification of Specifications
  - Property-Based Testing of Software
- Verification Results
- PatchLink Response
- Non-Root Unix Agent
- Questions
Introduction

- PatchLink and Unix Systems
  - Unix systems typically run enterprise services and critical operations at JPL
    - Current released version is 6.0067
    - Requires Sun JRE 1.2.2 or higher
    - Initially installs and runs as root privileged
    - NICE value is 10 but can be changed
    - If SSL used, use certificates from known root authorities such as Verisign
  - Viewed by JPL as a potential security risk area – especially for flight operations systems
Introduction (Cont.)

- JPL Request for Security Verification of Agent
  - Critical flight operations systems
  - Inter connectivity between systems via NFS
- NASA request for verification and results
  - NASA CIO to be informed of results
  - Other NASA Centers that are heavy Unix environments request for JPL verification results
PatchLink Response to Request
- Acceptance of JPL team with NDA
- Participation by PatchLink in verification activities
- PatchLink made changes to Unix agent based on verification results

Goal: Verify Security of Unix Agent
- Verify critical security properties
- Agent does not pose a security risk to JPL, especially to its Flight Operations Systems
Final Report on Verification Activities Delivered

- PatchLink Corporation
  - Provided response to verification findings
  - Provided information on issues uncovered
- Jet Propulsion Lab (JPL) Management
- NASA CIO and IV&V Center
Software Assurance and Verification

![Diagram showing JPL Verification Activities]

- PBT
- FMF
- THEOREM PROVING
- STATIC CHECKING
- RUNTIME ANALYSIS
- TESTING
- TRADITIONAL
Software Assurance

Software Security Assurance Instruments

- Formal Modeling
  - Stanford Research Institute: John Rushby - PVS/SAL/ICS
  - Jet Propulsion Lab: John Powell – FMF/SPIN

- Code Analysis
  - Klocwork – static code analyzer – excellent GUI interface
  - DevPartner Security Checker – compile, runtime and integrity analyzer for known security problems
  - CodeAssure – John Viega – code analyzer, but not specifically built for security - generic
  - Property-Based Tester (PBT) – Matt Bishop, UC Davis - dynamic & developed for security
Software Assurance (Cont.)

- Penetration testing – scanners
- Resources Required for Performing Verification:
  - Specifications and Design Supplied to JPL by PatchLink
  - JPL on-site at PatchLink with Engineers
  - Instruments to validate PL design and code
    - Model-Based Verification Flexible Modeling Framework (FMF) – used with SPIN
    - Property-Based Tester (PBT)
Verification Activities

- Time-frame for Verification
  - Pre-site visit
    - Modeling – one week for modeling preparation
    - Property-Based testing – one week for preparation
  - On-site verification activities spanned one week
Verification Activities (Cont.)

- Security Properties Identified for Verification
  - 18 security properties identified as critical
    - Subset of security properties
    - Focus on agent operations
  - Properties ranged from agent-server communication to job/package acceptance and installation processes
### Key Unix Agent Security Properties

<table>
<thead>
<tr>
<th>Agent Properties</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The agent and server shall be capable of secure communication</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>2. The agent and server shall have an identification that uniquely mutually associates them</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>3. The agent and server shall authenticate to each other using their unique identification</td>
<td>MBV – Verified - logically Implied by 1 and 2</td>
</tr>
<tr>
<td>4. The agent shall validate all packages that they are from its associated server</td>
<td>MBV - Verified – logically Weaker version of 3</td>
</tr>
<tr>
<td>5. The agent shall validate that the package is un-tampered (like using an MD5 checksum)</td>
<td>MBV/PBT – Verified Logically</td>
</tr>
<tr>
<td>6. The agent shall recognize packages that do not complete their installation</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>7. The agent shall have a recovery process for packages that have partial installation or otherwise fail during installation</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>8. The agent shall run at low priority</td>
<td>PBT – Verified to Hold</td>
</tr>
<tr>
<td>9. The agent shall recognize conflicts with other processes that generate high CPU utilization</td>
<td>Verify by other means: Kernel function</td>
</tr>
<tr>
<td>10. The agent shall go to sleep when CPU utilization is high</td>
<td>Verify by Other Means: Kernel function</td>
</tr>
<tr>
<td>11. The agent shall monitor activity for system resources</td>
<td>Verify by Other Means: Kernel function</td>
</tr>
<tr>
<td>12. The agent shall recognize conflicts with use of JAVA resources</td>
<td>Verify by Other Means: Kernel function</td>
</tr>
<tr>
<td>13. The agent shall go to sleep when it detects conflicts with JAVA resources</td>
<td>Verify by Other Means: Kernel function</td>
</tr>
<tr>
<td>14. The agent shall only accept connections that it has initiated</td>
<td>MBV/PBT - Verified to Hold</td>
</tr>
<tr>
<td>15. The agent shall have a network session time-out</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>16. The agent shall have a package installation time-out</td>
<td>MBV - Verified to Hold</td>
</tr>
<tr>
<td>17. The agent shall provide logging of all its events</td>
<td>Verified by Other Means: Inspection</td>
</tr>
<tr>
<td>18. The agent shall be capable of running as non-root and maintain reporting capabilities</td>
<td>Verify by Other Means</td>
</tr>
</tbody>
</table>
Model-Based Verification and the Flexible Modeling Framework
Model-Based Verification

- Model-Based Verification Requires Building a State-Based Model of the System
- Requires Identifying Properties To Be Verified
- Performs Automated Checking of the Model for Property Violations
- Model Checkers Perform Exhaustive Search of State Space

Model-Based Verification (Cont.)

- The objective - verify a model of a system’s security over its corresponding state space (the subset of reachable states).
- The Model Checker’s function - determine if a given model of a system’s behavior satisfies its security requirements.
- Models expressed in a suitable language
  Properties expressed in suitable logic (e.g. Linear Temporal Logic – LTL)
- The goal is to find errors as opposed to proving correctness
Model-Based Verification (Cont.)

- State space is the set of total reachable system states represented in the model
- A given state consists of all variables in the model and their associated values at a given point in time
- Software model checkers explore all paths from a start state by examining transitions to determine reachability of a state that violates the property
- When properties are violated, checker gives counterexample and stops
- Properties are verified as holding or not holding for each transition
- Does not obviate need for experts since development of verification model is non-trivial
Information Technology Security

Model-Based Verification (Cont.)

![Diagram]

Figure 2: Model Component Combination Tree and Components Model Checked [19]

- ACD → AD → A = Potential Violation
- ABC → AB & AC → A = Mitigation to A from ACD
Property 5: The agent shall validate that the package is un-tampered

The formal property is:

\[
\text{It is always the case that}\ \\
\text{if (Bad_Msg Received) then (do not}} \\
\text{Receive_Next_Msg) until (Bad_Msg Detected))}
\]

The Linear Temporal Logic (LTL) property is:

\[
\Box ((\text{Patch_Type}==17) \rightarrow ((\text{Patch_Type}==17) U \\
\text{(Bad_Message==1)}))
\]
SPIN Model-Checker Output for Property 5:
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
depth 72: Claim reached state 11 (line 290)
(Spin Version 4.2.1 -- 8 October 2004)
  + Partial Order Reduction

Full statespace search for:
  never claim +
  assertion violations + (if within scope of claim)
  acceptance cycles + (fairness disabled)
  invalid end states - (disabled by never claim)
State-vector 52 byte, depth reached 58057, errors: 0
421689 states, stored (466713 visited)
376840 states, matched
843553 transitions (= visited+matched)
  0 atomic steps
hash conflicts: 166086 (resolved)

Stats on memory usage (in Megabytes):
  25.301 equivalent memory usage for states (stored*(State-vector + overhead))
  22.210 actual memory usage for states (compression: 87.78%)
  State-vector as stored = 45 byte + 8 byte overhead
  2.097 memory used for hash table (-w19)
  32.000 memory used for DFS stack (-m1000000)
  31.816 other (proc and chan stacks)
  0.092 memory lost to fragmentation
  56.216 total actual memory usage
Property-Based Testing (PBT)
Tester’s Assistant
Property-Based Testing

- Goal is to validate that an implementation satisfies its specifications
  - Many errors in software are caused by generalizable flaws in the source code
  - Property-based testing assures that a given program is free of specified these flaws
  - Property-based testing uses property specifications and a data-flow analysis of the program to guide evaluation of test executions for correctness and completeness
Property-Based Testing (Cont.)

- Property-based testing tool – Tester’s Assistant (Matt Bishop, UC Davis)
  - Perform code slicing on applications for a known set of vulnerabilities
  - Test for vulnerabilities in code on system or whenever computing environment changes
- Compare program actions with specifications
  - Create low-level specifications
  - Instrument program to check that these hold
  - Run program under run-time monitor
  - Report violations of specifications
Property-Based Testing (Cont.)

Knowledge of Security

- Specification of Security Model
- Property Specifications
- Assurance

Property-based Testing

- Property
- Slicing
- Testing
- Validation of Property

PBT Overview Process

PBT Specific Process
Property-Based Testing (Cont.)

Property 8: Agent Shall Run at Low Priority

- Accomplished in the script “detect”
- Spec file contains invariant "check(nice > 0)" where nice is the priority
- Script read priority number from a configuration file
- Stored priority in the variable NV
- Just before the shell code to lower the priority, the instrumenter added line: echo “assert (nice = $NV)” >> em_trace
  - The trace file was named “em_trace”
  - Test Execution Monitor was given spec file and trace file
- TEM reported that the invariant was satisfied (nice = 10)
Denial of Service (DoS) Finding

- Potential for Denial of Service (DoS) Attack by Connection to “Wakeup” Port 25252 on Workstation (Property 14 Violation: The agent shall only accept connections that it has initiated)
  - Purpose of port is to verify agent status
  - Probing port causes agent to wake up and call into PatchLink PLUS server
  - Server limited to responding to 200 simultaneous agent connections (2-processor, 3GB)
  - Probing port 25252 across network could cause large number of agents to connect to PLUS server simultaneously resulting in DoS
Verification Results and PatchLink Response
Verification Results

- Modeling with FMF Resulted in One Minor Finding: Potential Denial of Service Weakness in Communication
  - Weakness mitigated by secure communications
- PBT Resulted in Two Weaknesses Uncovered in Code
  - CRC Checksum of patch provided, failed resulting in potential for bad patch package to be received
Verification Results (Cont.)

- Denial of Service weakness in ‘wakeup’ port where network probes could cause agents to flood server with connection response requests

- Property 18: One finding was that Unix agent runs only with root privileges
Verification Results (Cont.)

- While the verification does not prove that the agent is secure, it does provide a higher level of confidence in the security of the agent operating within a secure environment.

- Take Precautions to Run PatchLink Securely
  - Use SSL and firewalls
  - Lockdown PLUS Servers
    - Disable unneeded services and ports
    - Apply system configuration security controls
    - Use of monitoring software, MOM, Tripwire, Tivoli
  - THINK “SECURE”!
Verification Results (Cont.)

- Configuration Manage (CM) PatchLink PLUS servers

- Security Verification Assessment
  - Unix Agent is assessed to be secure *iff*
    - PatchLink PLUS server and agents are not run in open environments
    - Secure communications used
    - Lockdown protections are in place on systems

- PatchLink Unix agent accepted for use at JPL
- Results accepted by NASA IV&V Center
PatchLink Response

- Property #4 (The agent shall validate all packages that they are from its associated server): SSL communication layer guarantees the authenticity of the server that communications are going to (that is to say, the agent trusts the issuing certificate authority (eg: Verisign))

- Property #5 (The agent shall validate that the package is untampered): Files downloaded across SSL pipe are checksum verified as they are decompressed by the PatchLink agent
PatchLink Response (Cont.)

- Property #8 (the agent shall run at low priority): Difficult to determine exact CPU utilization by java process
  - Can verify CPU utilization when patch is deployed

- Property #14 (The agent shall only accept connections that is has initiated): The 'wakeup' port feature is now disabled by default in UNIX agent install
  - PatchLink changed default ‘wake up port’ after discussion of potential for exploit
PatchLink Response (Cont.)

- **Property #17** (The agent shall provide logging of all its events): Agent keeps verbose event logs
  - Verified by inspection

- **Property #18** (the agent shall be capable of running as non-root and maintain reporting capabilities): This is a new feature that the PatchLink team has implemented in coordination with JPL for NASA
Non-Root Unix Agent
Security Property 18
Non-Root Unix Agent

- Property 18: The agent shall be capable of running as non-root and maintain reporting capabilities.
- Default install for Unix Agent is Root Privileged.
- Installation Script Provided by PatchLink for Installing and Running Unix Agent as Non-Root:
  - Allows for reporting only
  - Does not allow acceptance of jobs/packages from PatchLink PLUS server
  - Useful for highly configuration managed systems (e.g., flight operations systems)
Non-Root Unix Agent (Cont.)

- Script Wrapped With Pre and Post Scripts by JPL (Tom Wolfe): Script Available on Request
  - Two modes:
    - Silent – all parameters entered on command line
    - Manual – user is prompted for input
- Pre-Install Scripts
  - Check for previous installation
    - If YES – where installed and where to re-install?
    - If directory is changed, MUST un-install existing agent first
Information Technology Security

Non-Root Unix Agent (Cont.)

- Setup standard enterprise user (UID) and group id (GID) to be used by agent

- Post-Install Script
  - Change NICE value to 19 (lower priority – default is 10)
  - Setup CRON job for regular, periodic reporting (determined at time of installation)
    - Requires setting user to allow execute of detect.csh

- Also provide
  - Example command line parameters for silent install
  - Start and stop CRON job scripts
  - Problem/Issue workarounds
Final Comments

- Thanks go to PatchLink for their cooperation and participation in this verification activity.
- The willingness to allow an outside organization to perform software security verification on one of their key software components, the Unix Agent, coupled with their responsiveness in acknowledging and addressing the minor findings is both commendable and responsible to one of their key customers and the Internet community at large.
- This type of working relationship between vendor and customer represents a major paradigm shift and step forward in establishing a strong trust relationship between partnering entities.
Contact Information

David Gilliam, JPL
400 Oak Grove Dr., MS 144-210
Pasadena, CA 91109
Phone: (818) 354-0900
Email: david.p.gilliam@jpl.nasa.gov

John Powell, JPL (Modeling and Flexible Modeling Framework)
MS 125-233
Phone: (818) 393-1377
Email: john.d.powell@jpl.nasa.gov

Professor Matt Bishop, UC Davis (Property Based Testing)

phone: +1 (530) 752-8060
fax: +1 (530) 752-4767
email: bishop@cs.ucdavis.edu
PBT:
Questions?

BAD

GOOD