Dataflow Anomaly Detection

Presented By
Archana Viswanath
Computer Science and Engineering
The Pennsylvania State University
Anomaly Intrusion Detection Systems

- Model normal behavior.
- Attack - Any digression from normal behavior

Important Questions…

- How is normal behavior modeled?
- Is the modeling of normal behavior complete????
Modeling of Normal Behavior

- **Model program behavior** – System calls, Strings, Finite State Automata, Push-down automata.
- **How is this program behavior learnt?** – Training, static analysis of code or binary, etc.
- **Weakness** – Singular focus on control flows with little emphasis on data flows involving system call arguments.
Attacks

Race Condition Attacks
✓ They do not change system calls.
✓ They change only the interpretation of their operands.

Mimicry Attacks
✓ Attack is modified to closely mimic program behavior.

Non-control-flow hijacking Attacks
✓ Targets manipulation of security critical data.

Control flow modeling of program behavior are susceptible to the above types of attacks.
Contribution of this paper…

✓ IDS that is based on learning temporal properties involving arguments of different system calls.

✓ Dataflow property relationships.

✓ Efficient algorithm – captures control flow context in data flow arguments.

✓ Experimental evaluation of attack detection, model precision and false alarm rates.
Data Flow Behavior

Modeling
✓ Program behavior is defined in terms of externally observable events generated by the program.
✓ This is modeled in terms of system calls.

Definitions
- *Execution trace* for a program P-denoted by $T(P)$, is the sequence of all the system calls executed by P during its execution. It includes information about system call arguments.
- *System call tracer* - Records system calls made by P.
- *Trained behavior* of P – Set $\mathcal{T}(P)$ of all traces generated by P during its training runs.
- *Behavior model* for P - Automaton that accepts traces.
Labeled Traces

Motivation
✓ Used to encode control flow context into learning data flow properties.

Example:
L1: fd1 = open("/etc/passwd", O_RDONLY);
(...) /* perform authentication */
L2: fd2 = open("/tmp/out", O_RDWR);

Partition sets of arguments based of the same system call based on control flow.
Control context is encoded by giving names for event arguments.

open@L1 X = "/etc/passwd" Y = "read"
open@L2 Z = "/tmp/out" W = "write"
Dataflow Relationships

Unary Relation
- Capture properties of a single argument.
- Represented using the form $X \ R \ c$, where $X$ is an argument name, $R$ denotes a relation, and $c$ is a constant value.
- Eg: equal, elementOf, subsetOf, range, isWithinDir, hasExtension.

Uniqueness of work
- All previous work focused only on unary relations.
- Use of control-flow context to support accurate learning.
Dataflow Relationships

Binary Relation

- Capture relationships between two event arguments.
- Eg: equal, contains, hasSameDirAs, hasSameBaseAs, hasSameExtensionAs, isWithinDir.

Definitions:
(i). $X \ R_T Y$ - Relation holds iff, for each occurrence of $X$ and its closest preceding occurrence of $Y$ in $T$, $X \ R_T Y$ holds.

Eg: Labeled trace $T$

$Y = 1, Z = 2, X = 1, Y = 2, X = 2$. 
Dataflow Relationships

Binary Relation – Contd…

(ii). $X \underleftarrow{R_T} Y$ holds iff $X R Y$ holds for each pair $X, Y$ in $T$ without an intervening $X$ or $Y$.

Eg: For $\text{isWithinDir}$ relationship:
$Y = "/tmp", X = "/tmp/f1", X = "/f2", Y = "/var", X = "/var/g1", X = "/g2$ 

(iii). $X \underleftarrow{R^n_T} T Y$ holds iff $X R Y$ holds for each occurrence of $X$ and its $n+1$th preceding occurrence of $Y$.

Eg: For the trace $T$
$X = 1, Y = 0, X = 2, Y = 1, X = 3, Y = 2, ....$
Clearly, the value of $Y$ equals the value of the last but one preceding $X$. 
Example

```
1. int main(int argc, char **argv) {
2.   source_dir = argv[1]; target_file = argv[2];
3.   target_fd = open(target_file, WR);
4.   push(source_dir); // uses a global stack
5.   while (dir_name = pop()) != NULL) {
6.     dir = opendir(dir_name);
7.     foreach (dir_entry ∈ dir) {
8.       if (isdirectory(dir_entry))
9.         push(dir_entry);
10.      else {
11.         source_fd = open(dir_entry, RD);
12.         read(source_fd, buf);
13.       }
14.     }
15. }
16. }
17. }
18. close(target_fd);
19. exit(0);
20. }
```
### Example – Sample Trace

<table>
<thead>
<tr>
<th>Operation Traces</th>
<th>Control-Flow Transition</th>
<th>Argument Values</th>
<th>Satisfied Data-Flow Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program started with arguments &quot;./opt/proj&quot;,&quot;/tmp/proj.tar&quot;</td>
<td></td>
<td>{I = &quot;./opt/proj&quot;, O = &quot;/tmp/proj.tar&quot;}</td>
<td></td>
</tr>
<tr>
<td>$\ell_3$: open(&quot;/tmp/proj.tar&quot;, WR)=3</td>
<td>start $\rightarrow$ $\ell_3$</td>
<td>{F_3 = &quot;/tmp/proj.tar&quot;, M_3=WR, F_3D3=3}</td>
<td>$F_3$ equal O, (M_3) elementOf {WR}</td>
</tr>
<tr>
<td>$\ell_6$: opendir(&quot;./opt/proj&quot;)</td>
<td>$\ell_3$ $\rightarrow$ $\ell_6$</td>
<td>{F_6 = &quot;/opt/proj&quot;}</td>
<td>$F_6$ isWithinDir (I)</td>
</tr>
<tr>
<td>$\ell_8$: isdirectory(&quot;./opt/proj/README&quot;)</td>
<td>$\ell_6$ $\rightarrow$ $\ell_8$</td>
<td>{F_8 = &quot;./opt/proj/README&quot;}</td>
<td>$F_8$ isWithinDir (F_6)</td>
</tr>
<tr>
<td>$\ell_{11}$: open(&quot;./opt/proj/README&quot;, RD)=4</td>
<td>$\ell_8$ $\rightarrow$ $\ell_{11}$</td>
<td>{F_{11} = &quot;/opt/proj/README&quot;, M_{11} = RD, F_{11}D_{11}=4}</td>
<td>$F_{11}$ equal (F_8), (M_{11}) elementOf {RD}</td>
</tr>
<tr>
<td>$\ell_{12}$: read(4)</td>
<td>$\ell_{11}$ $\rightarrow$ $\ell_{12}$</td>
<td>{F_{D12}=4}</td>
<td>$F_{D12}$ equal (F_{D11})</td>
</tr>
<tr>
<td>$\ell_{13}$: write(3)</td>
<td>$\ell_{12}$ $\rightarrow$ $\ell_{13}$</td>
<td>{F_{D13}=3}</td>
<td>$F_{D13}$ equal (F_{D3})</td>
</tr>
<tr>
<td>$\ell_{14}$: close(4)</td>
<td>$\ell_{13}$ $\rightarrow$ $\ell_{14}$</td>
<td>{F_{D14}=4}</td>
<td>$F_{D14}$ equal (F_{D11})</td>
</tr>
<tr>
<td>$\ell_8$: isdirectory(&quot;./opt/proj/src&quot;)</td>
<td>$\ell_{14}$ $\rightarrow$ $\ell_8$</td>
<td>{F_{8}' = &quot;./opt/proj/src&quot;}</td>
<td>(F_{8}') isWithinDir (F_6)</td>
</tr>
<tr>
<td>$\ell_6$: opendir(&quot;./opt/proj/src&quot;)</td>
<td>$\ell_8$ $\rightarrow$ $\ell_6$</td>
<td>{F_6 = &quot;./opt/proj/src&quot;}</td>
<td>(F_6) isWithinDir (I)</td>
</tr>
<tr>
<td>$\ell_8$: isdirectory(&quot;./opt/proj/src/a.c&quot;)</td>
<td>$\ell_6$ $\rightarrow$ $\ell_8$</td>
<td>{F_8 = &quot;./opt/proj/src/a.c&quot;}</td>
<td>(F_8) isWithinDir (F_6)</td>
</tr>
<tr>
<td>$\ell_{11}$: open(&quot;./opt/proj/src/a.c&quot;, RD)=4</td>
<td>$\ell_8$ $\rightarrow$ $\ell_{11}$</td>
<td>{F_{11} = &quot;./opt/proj/src/a.c&quot;, M_{11} = RD, F_{11}D_{11}=4}</td>
<td>(F_{11}) equal (F_8), (M_{11}) elementOf {RD}</td>
</tr>
<tr>
<td>$\ell_{12}$: read(4)</td>
<td>$\ell_{11}$ $\rightarrow$ $\ell_{12}$</td>
<td>{F_{D12}=4}</td>
<td>$F_{D12}$ equal (F_{D11})</td>
</tr>
<tr>
<td>$\ell_{13}$: write(3)</td>
<td>$\ell_{12}$ $\rightarrow$ $\ell_{13}$</td>
<td>{F_{D13}=3}</td>
<td>$F_{D13}$ equal (F_{D3})</td>
</tr>
<tr>
<td>$\ell_{14}$: close(4)</td>
<td>$\ell_{13}$ $\rightarrow$ $\ell_{14}$</td>
<td>{F_{D14}=4}</td>
<td>$F_{D14}$ equal (F_{D11})</td>
</tr>
<tr>
<td>$\ell_{18}$: close(3)</td>
<td>$\ell_{14}$ $\rightarrow$ $\ell_{18}$</td>
<td>{F_{D18}=3}</td>
<td>$F_{D18}$ equal (F_{D3})</td>
</tr>
<tr>
<td>$\ell_{19}$: exit(0)</td>
<td>$\ell_{18}$ $\rightarrow$ $\ell_{19}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning Relations

Unary Relations

✓ For each event argument the algorithm maintains a list of all the values encountered in all the traces.
✓ If the number of values exceeds a threshold then the algorithm approximates the set.

Binary Relations

1. `procedure LearnRelations(EvArg X, Value V) {`
2. \( \mathcal{Y} = \text{ValTable.lookup}(V, R); \)
3. \( \text{CurRel}[R][X] = \text{CurRel}[R][X] \cap \mathcal{Y}; \)
4. \( \mathcal{Y}_n = \mathcal{Y} \cap \text{NewArgs}(X); \)
5. \( \text{CurRel}[R][X] = \text{CurRel}[R][X] \cup \mathcal{Y}_n; \)
6. \( \text{ValTable.update}(X, V); \)
7. `}`
Implementation

✓ Consists of an online and offline component.

✓ Online component – tracer.

✓ Offline component – log file parser (reconstructs the system call events and feeds them to a learning module).
Detection of Attacks

*WU-FTPD : Corruption of user identity data*

✓ Involves the following code in getdatasock() function.

\[
\begin{align*}
\text{L1: } & \text{ seteuid}(0); \\
& \text{ setsockopt(...)}; \\
& \text{ ...} \\
\text{L2: } & \text{ seteuid(pw->pw_uid);}
\end{align*}
\]
Detection of Attacks

Netkit Telnetd: Corruption of filename to be executed

- At the beginning of each client connection, the telnet daemon authenticates its user with an external program.
- The name of this program is stored in a variable `loginprg`.
- A heap overflow vulnerability is used to overwrite this variable with the value `/bin/sh`.
- Subsequent authentication by a user will result in a root shell.
- `loginprg` always has the value `/bin/login`.
- Attack detected as a violation of the value normally observed as the argument of `execve`.
Detection of Attacks

**GHTTPD**: Directory traversal by corrupting filename

- Stack overflow in GHTTPD web server can be used to evade the path name check, and execute an arbitrary program.
- Variable `ptr` is a pointer to a text string of the URL requested by a remote client.
- Attack occurs in the following code fragment in the `serverconnection` function.
- The function `Log()` has a buffer overflow vulnerability.
- `Ptr` can be changed to point to `/cgi-bin/../../../../../bin/sh`
- Our system knows that the file *isWithinDir* CGI-BIN.

```c
if (strstr(ptr, "/.."))
    return ... //reject request
Log(...);
L1: if (strstr(ptr, "cgi-bin")) execve(ptr, ...)
```
Detection of Attacks

_Fingered symlink vulnerability_

✓ Symlink vulnerability in BSD _fingerd_
✓ This server uses a local _finger_ client program to serve remote requests.
✓ Server and client run with root privilege.
✓ A user can create a symbolic link in his home directory that points to a file readable only to root.
✓ By running a finger on himself the user can see the contents of this file.
✓ This is detected in our approach – violation between the name of the user to be fingered and the directory of the filename being opened.
Detection of Attacks

**Race Condition Attack**

- These occur when applications incorrectly assume that a sequence of operations on files is atomic.
- Consider `rm -r /tmp/a/`, `a` contains a subdirectory `b`.
- `rm` descends in and out of a directory using `chdir(“..”)`
- When `rm` descends into `/tmp/a/b`, the attacker can rename `/tmp/a/b` to `tmp/b`.
- Now when `rm` executes `chdir(..)` it will go to `/tmp` and start deleting files from there.
- This implementation detects that the arguments that are given to `rmdir` should be within the directory name given by the command-line argument.
Conclusion

✓ This approach aims at enhancing the accuracy of host-based intrusion detection systems.

✓ This approach is effective as it incorporates the control flow context in data flow properties.