Program-Analysis-Supported Identification of Applications in Large Networks

Christopher Kruegel

Computer Security Group

ARO MURI Meeting
Arizona State University, October 28, 2013
Real World Enterprise Network

Mission

Cyber-Assets

Analysis to get up-to-date view of cyber-assets

Analysis to determine dependencies between assets and missions

Data

Mission Model

Cyber-Assets Model

Create semantically-rich view of cyber-mission status

Sensor Alerts

Correlation Engine

Impact Analysis

Data

Simulation/Live Security Exercises

Data

Analyze and Characterize Attackers

Data

Predict Future Actions

COAs

ARO MURI Meeting, ASU, October 28, 2012
Motivation

Thrust I: Obtaining an up-to-date view of the available cyber-assets

- Need to know and model assets on your network
  - network services (beyond IP address and ports)

Thrust II: Obtaining understanding of the dependencies between missions and assets

- Find dependencies and redundancies between services
- Find relationships (mappings) between missions and assets
- Find assets and activities critical for network (or particular mission)
Accomplishments

• Year 1
  – models to “fingerprint” specific programs and network services
  – track services and identify bot-infected machines

• Year 2 and Year 3
  – service dependency model
  – algorithms for ranking assets and services
  – techniques to extract (indirect) dependencies between assets

• Year 4
  – develop techniques and tools to automatically infer network-based application fingerprints from program binaries
  – experiments to demonstrate utility of system for application identification (and malware detection)

ARO MURI Meeting, ASU, October 28, 2012
Service Identification

… or, “what applications are running on my network?”

• Manual data entry
  – cumbersome, not up-to-date

• Agents on the hosts
  – difficult to rollout and to maintain, what about hosts connected ad-hoc

• Passive techniques
  – listen for network traffic, analyze protocol
  – regex, application identification (Palo Alto Networks), L7, …
  – works only for services that are “speaking”
Active Service Identification

- Active techniques
  - probe hosts (in essence, port scanning) and look for open ports

- Make decision based on port numbers
  - simple, but limited (dynamic ports, multiple apps on same port, …)

- Response analysis (banner grabbing)
  - connect and send random bytes, analyze response
  - works when service sends recognizable answer (or answer at all)

- We propose a smarter response analysis
  - analyze program to determine what triggers specific response
  - build trigger packets to elicit unique (distinctive) response
Approach

Based on program analysis

• Collect set of relevant application

• Perform combination of static and dynamic binary analysis
  – for each application
    • extract port fingerprint (acts as an initial filter)
    • extract birthmarks (pairs of request – response pairs)
    • determine unique birthmark

• Scan network
  – when a certain port is found open
  – send all corresponding birthmark requests, examine responses
Advantages

• Fast and scalable
  – apply probing technique as fast as network scanning

• Reliable
  – probing content is extracted directly from malware binary
  – can often find control logic that is unique to each application
Extracting Application Models

• Ports can be selected
  – deterministically (static value)
  – at random
  – algorithmically (maybe based on some predictable environmental value, maybe based on some configuration file, …)
  – portprints need to reflect this

• Responses are
  – static responses (strings, …)
  – based (partially) on requests
  – birthmarks need to reflect this
Finding portprints

- Start program in monitored execution environment
- Collect instruction (execution) trace
- \texttt{Wait until \texttt{bind()} call occurs}
- Perform backward slicing to determine sources for port argument
- Extract program logic along the relevant program slice
- This logic represents the \texttt{portprint}
Guided, concolic (concrete/symbolic) execution for birthmark

- Run program until it performs `receive()` and waits for input
- Send some input packet (and mark input bytes as symbolic)
- Continue with forward symbolic execution until program makes decision based on symbolic value(s)
- Attempt to drive execution down a path that leads to a `send()`
  - program might simply not respond in case of invalid input
  - need to see response
  - implement forward looking process to guide execution
Forward Symbolic Execution

• Tracking of symbolic (tainted) input values
  – whenever a tainted value is copied to a new location, we must remember this relationship
  – whenever a tainted value is used as input in a calculation, we must remember the relationship between the input and the result

• Constraint set
  – for every operation on tainted data, a constraint is added that captures relationship between input operands and result
  – provides immediate information about condition under which path is selected
Forward Symbolic Execution

- Constraint set

```c
x = 0
x = read_input();
y = 2*x + 1;
check(y);
print("x = %d, x");
....

void check(int arg) {
    if (arg == 47)
        f();
    else
        g();
}
```

- $x = \text{input}$
- $y = 2x + 1$
- $\text{arg} = y$
- $\text{arg} = 47$

We can solve for alternative branch

- $y = \text{arg} = 47$
- $x = \text{input} = 23$
Forward Looking Process

- Statically analyze program for blocks that contain interesting function invocations (send, exit, ...)
- Propagate this information upward to callers
Forward Looking Process

- Perform (k-block) look-ahead, and pick path that leads to block with highest score

```plaintext
if (arg != 47)
    send()
    f()
    g()
    exit()
```

Graphical representation:
- High preference container
- Low preference container
- Regular code block
Verification Step

• Run request packet against application and ensure expected response
  – this is needed because analysis is not sound
    • we use stiched symbolic execution when backtracking
    • analysis can miss dependencies
    • certain constraints cannot be properly modeled (checksum, …)

• Ensure that response is unique, given the current set of responses for other applications
Evaluation

- Perform analysis for various types of popular server software
  - P2P / FTP / HTTP software including Apache, FileZilla, eDonkey, eMule, Morpheus, Limewire, Kazaa, …

- First experiment
  - install all applications in virtual network
  - ensure that identification works as expected

- Second experiment
  - scan 3 class C university networks (on popular application ports)
  - no incorrect results (no false positives)
  - but few applications were not recognized (incomplete training set)
Evaluation

• Cool additional application: Why not detect malware?
  – in particular, malware that opens listening ports
  – P2P malware, backdoors, …
## Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conficker</td>
<td>P2P Bot</td>
<td>Nugache</td>
<td>P2P Bot</td>
</tr>
<tr>
<td>Phabot</td>
<td>P2P Bot</td>
<td>Sality</td>
<td>P2P Bot</td>
</tr>
<tr>
<td>NuclearRAT</td>
<td>Trojan Horse</td>
<td>BackOrifice</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td>Penumbra</td>
<td>Trojan Hose</td>
<td>Storm/Peacomm</td>
<td>P2P Bot</td>
</tr>
<tr>
<td>NuCrypt</td>
<td>Trojan Horse/Worm</td>
<td>Wopla</td>
<td>Trojan Horse</td>
</tr>
<tr>
<td>WinCrash</td>
<td>Trojan Horse</td>
<td>WinEggDrop</td>
<td>Spyware</td>
</tr>
</tbody>
</table>
## Portprints

<table>
<thead>
<tr>
<th>Malware</th>
<th>Type</th>
<th>Port Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conficker</td>
<td>algorithms</td>
<td>46523/TCP, 18849/UDP</td>
</tr>
<tr>
<td>Nugache</td>
<td>static, random</td>
<td>8/TCP, 3722/TCP</td>
</tr>
<tr>
<td>Sality</td>
<td>algorithms</td>
<td>6162/UDP</td>
</tr>
<tr>
<td>Phabot</td>
<td>random</td>
<td>1999/TCP</td>
</tr>
<tr>
<td>Storm/Peacomm</td>
<td>static</td>
<td>7871, 11217/UDP</td>
</tr>
<tr>
<td>BackOrfice</td>
<td>static</td>
<td>31337/TCP</td>
</tr>
</tbody>
</table>
# Birthmarks

<table>
<thead>
<tr>
<th>Malware</th>
<th># of BM</th>
<th>Malware</th>
<th># of BM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conficker</td>
<td>3</td>
<td>Peacomm</td>
<td>3</td>
</tr>
<tr>
<td>Sality</td>
<td>1</td>
<td>BackOrifice</td>
<td>14</td>
</tr>
<tr>
<td>Phabot</td>
<td>9</td>
<td>NuclearRAT</td>
<td>12</td>
</tr>
<tr>
<td>WinEggDrop</td>
<td>8</td>
<td>Penumbra</td>
<td>13</td>
</tr>
<tr>
<td>Nugache</td>
<td>7</td>
<td>WinCrash</td>
<td>1</td>
</tr>
<tr>
<td>NuCrypt</td>
<td>2</td>
<td>Wopla</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusions

• Work focused on Thrust I

• Automated extraction of application models from binary analysis

• Models can be efficiently leveraged via network scanning

• Experimental evaluation (incl. malware detection)
  – results demonstrate that approach is effective
Future Work

- Semantic analysis and labeling of network assets (what is a network proxy, NAT device, …) based on network behaviors
- Detection of caching
- Host-based analysis
  - agents that can extract relevant information
- Continuous monitoring
Thank You