The Internet of Backdoors

Sam L. Thomas\textsuperscript{1}
(joint work with Aurélien Francillon\textsuperscript{2}, Flavio D. Garcia\textsuperscript{1} & Tom Chothia\textsuperscript{1})

\textsuperscript{1}University of Birmingham
United Kingdom

\textsuperscript{2}EURECOM
France

March 7, 2018
1. What is a backdoor?
2. How do we find them?
What is a backdoor?
Zhang & Paxon (2000):

A backdoor is a mechanism surreptitiously introduced into a computer system to facilitate unauthorized access to the system.
What is a backdoor?

Schuster & Holz (2013):

A backdoor is a hidden, undocumented, and unwanted program or a program modification/manipulation that on certain triggers bypasses security mechanisms or performs unwanted/undocumented malicious actions.
What is a backdoor?

Can we do better?
What is a backdoor?

1. How do we recognise a backdoor-like construct within a system?
2. How do we differentiate between a backdoor and vulnerability?
Taking a step back

Two different perspectives to consider for a given system, or program
Taking a step back

Two different perspectives to consider for a given system, or program

- Developer/designer
- End-user
Taking a step back

Different views of a given system
Taking a step back

Different views of a given system

- Developer: source code, design specification, etc.
Taking a step back

Different views of a given system

- Developer: source code, design specification, etc.
- End-user: user-manual, program binary/device, etc.
Taking a step back

This leads to:

- Developer: **intentions** encoded into the system
- End-user: **expectations** of that system
At some level of abstraction we can model a system, or program using a finite state machine (FSM), where state transitions are labelled by events.
Different views of the same system

Developer/designer will have a conceptual model of their system:

- “Developer” FSM (DFSM)
Different views of the same system

The DFSM will be transformed by some process, e.g., first source code, then compilation, to produce the actual system:

- “Actual” FSM (AFSM)
Different views of the same system

The end-user will have expectations of the system (AFSM):

- “Expected” FSM (EFSM)
Different views of the same system

If the user analyses the AFSM (by reverse engineering) they transform their EFSM:

- "Reverse-engineered" FSM (RFSM)
Recognising a backdoor

Now we have different ways of viewing a system, but:

- How do we obtain a RFSM from the \textit{black-box} view we have?
- How do we recognise a backdoor within the RFSM?
Perform different types of analysis of the system:

- Static analysis
- Dynamic analysis
We obtain a RFSM due to refining our EFSM by analysing the AFSM
Recognising backdoors

What do all backdoors have in common?
Towards a framework for recognising backdoors

From the definitions earlier:

- Trigger mechanism
- Some sort of (unexpected) privileged access
Towards a framework for recognising backdoors

We need some way to get input to the system in order to satisfy the trigger:

- We call this the input source
Towards a framework for recognising backdoors

That input will serve to satisfy trigger condition(s):
- We call this the backdoor trigger
Towards a framework for recognising backdoors

To reach the final privileged access:

- Some computation or transition will be made: we call this the backdoor payload
- Privileged access is obtained in what we call the privileged state
Thus, we have four components for any backdoor:
A framework for recognising backdoors

- Adding a backdoor to our EFSM is a refinement process
A framework for recognising backdoors

- Adding a backdoor to our EFSM is a refinement process
- We add new states or state transitions to the EFSM, based on analysing the AFSM
A framework for recognising backdoors

Two ways to add states and transitions:

- Directly from analysis methods (i.e. we discover them); we call these explicit
A framework for recognising backdoors

Two ways to add states and transitions:

- Directly from analysis methods (i.e. we discover them); we call these explicit
- We create our own; we call these non-explicit
Input source:

- Could be: system clock, input buffer, socket, etc.
- It is the origin of environment
Trigger mechanism:

- Conditions required to transition to the payload (i.e. collection of states and state transitions)
- Transition to payload is explicit; states are explicit
Trigger mechanism:

```
1. strcmp(username, "backdoor")
2. if (vulnerable_password_check(...))
   vulnerable_password_check(...)
else if (username != "backdoor")
   safe_user_auth(username, ...)
else
   authenticated
```

- Transition to payload is not explicit
- We call this case a bug-based backdoor
Payload:

```c
if (strcmp(password, "_BACKDOOR_") == 0 \n   || is_valid_password(password)) {
   // Authenticated
} else {
   // Not authenticated
}
```

- Essentially the solution to a puzzle.
- How to get from satisfaction of the trigger conditions to the privileged state
- Most basic case is a single transition
Other basic case: explicit transition to *payload*, where *payload* has explicit components

Situation where intermediate computation is required after the trigger to get to the privileged state
Payload:

- Non-explicit transition to *payload*, where *payload* has both explicit and non-explicit components
- Situation created via a bug-based trigger: allows creation of non-explicit states and transitions
- Similar to, e.g. ROP-based exploit payload
Explicit transition to *payload* with non-explicit *payload* components

Program performs processing of some input to the *payload*; is able to enter states controlled by that

E.g., there is some interpreter embedded within the program “unlocked” by the backdoor trigger
Framework in detail

Privileged state:

- Can be reachable under normal system execution (explicit)
- Can be reachable only by backdoor (can be either explicit or non-explicit)
What is a backdoor?

- We have a framework to decompose backdoor-like functionality
- What if we find a backdoor or vulnerability in our RFSM?
What is a backdoor?

- We have a framework to decompose backdoor-like functionality
- What if we find a backdoor or vulnerability in our RFSM?
- Still no way to tell if that functionality is actually a backdoor or vulnerability
Different views of the same system

Recall:

- We have four ways to view a system: DFSM, AFSM, EFSM, RFSM.
A vulnerability is an *accidental* construct:

- Can also be recognised by decomposing it using our framework
A vulnerability is an *accidental* construct:

- Can also be recognised by decomposing it using our framework
- Is present in the AFSM and RFSM, but **not** the DFSM
A backdoor is an *intentional* construct:

- Can be recognised by decomposing it using our framework
A backdoor is an *intentional* construct:
- Can be recognised by decomposing it using our framework
- Is present in the DFSM, AFSM and RFSM
Backdoor or vulnerability?

How do we know if the backdoor-like functionality was in the DFSM?
Backdoor or vulnerability?

How do we know if the backdoor-like functionality was in the DFSM?

- Consider explicit vs. non-explicit states and state transitions
How do we know if the backdoor-like functionality was in the DFSM?

- Consider explicit vs. non-explicit states and state transitions
- If trigger transition is *explicit* there is an indication of intent:
  - A backdoor
Backdoor or vulnerability?

How do we know if the backdoor-like functionality was in the DFSM?

- Sometimes we can’t tell from analysis. . .
- A deniable backdoor will be indistinguishable from an *accidental* vulnerability
- Compare different versions of software, i.e. its lineage:
  - What can we infer?
Backdoor detection
An Ideal Situation
An Ideal Situation
An Ideal Situation
An Ideal Situation
A Real-world Situation
A Real-world Situation
Challenge

- How do we reduce the manual effort required to identify undocumented functionality and backdoors within software?
Challenge

- How do we reduce the manual effort required to identify undocumented functionality and backdoors within software?
Motivation

- Undocumented functionality? Backdoors?
  - Authentication bypass by “magic” words.
  - Hard-coded credential checks.
  - Additional protocol messages that activate unexpected functionality.
  - Common services that perform non-standard functionality.
Focus on IoT devices:

- Lots of devices, lots of firmware, different architectures.
- Devices are attached to our networks often without regard for how secure they are.
- Can’t manually analyse every firmware image.
Tools

- **HumIDIFy**: detects undocumented, non-standard functionality in common services.
- **Stringer**: detects hard-coded credentials and undocumented protocol messages.
Both tools:

- Lightweight analysis.
- Reduce time required and expertise to perform analysis.
Method – Overview

- Uses machine learning to identify common executable classes (e.g. FTP server, Web server, ...).

- Tests to see if these identified common services perform more than their expected functionality (e.g. a Web server that listens for commands on a high UDP port and executes them as root on the device).
Method – Machine Learning

- Uses semi-supervised learning: train a classifier using some labelled instances and a larger amount of unlabelled instances.
- Uses an technique called self-training: iterates until some stability is reached on the performance of the classifier.
- On real-world test data (manually labelled, independent from training set): 96.4523% correctness.
High-level domain-specific language (DSL) to encode expected program functionality.

- DSL interpreter processes *functionality profile* and target executable.
- Have a *functionality profile* for each type of common service – they have known, well-defined behaviour.
Example rules written in the DSL:

```java
rule handles_socket() =
  function_ref("socket")

rule handles_tcp() =
  handles_socket() && (function_ref("recv") ||
  function_ref("send"))
```
Tenda Router web-server analysis with HumIDIFy:

$ ./HumIDIFy model/BayesNet httpd
]] HumIDIFy: version 1.0 ,-. 
]------------------------|-'
[i] performing feature extraction...
[i] classifying binary...
-> File : httpd
-> Profile : webserver (with confidence 100.00%)
[i] checking binary’s functionality...
-> Warning : udp-based api usage detected
-> Judgement : potentially anomalous
Method – Overview

- Assigns scores to static data and functions to indicate their relevance/potential.
- Generates a summary report of the executable using scoring for faster, simpler analysis.
Method – Overview

- Automatically identifies potential static data comparison functions.
- Extracts the arguments passed to those functions when the function call influences a branch condition.
- Maximises scores to static data based on how much CFG functionality they guard.
Motivating Example

HTTP protocol parser from mini_httpd binary:
Call-site Properties

**Argument references**: at least one argument refers to the data/read-only data section:
Call-site Properties

**Function arity**: (number of arguments passed): usually 2-3:
Call-site Properties

Branching properties: boolean comparison (i.e. matches or not):

```c
if (strncasecmp(s1, "GET") == 0) {
    ...
} else {
    ...
}
```

```c
if (strncasecmp(s1, "HEAD") == 0) {
    ...
} else {
    ...
}
```

```c
if (strncasecmp(s1, "POST") == 0) {
    ...
} else {
    ...
}
```
Call-site Properties

Local call frequency: (for parsers: use same comparison function many times with different static data):
Data Properties

Identify static data properties (with parsers in mind):

- No format string-like data.
- Restricted use of whitespace (no tab, vertical tab).
- Printable characters (i.e. no control sequences).
$ ./Stringer td3250

*** attempting to locate comparison functions...

[h] 15669 functions analysed; comparison functions:
[c] strcmp (1388.100000)
[c] strncmp (773.326250)
...

*** computing scores...
...

[f] 556.59: _ZN9CLoginDlg5LogInEPKcS1_b
  288.35: admin (via: strcmp)
  60.92: ppttzz51shezhi (via: strcmp)
  49.83: 6036logo (via: strcmp)
...

Case studies
Hard-coded Credentials in Ray Sharp DVR Firmware

Identification of hard-coded credential pair in Ray Sharp DVR firmware:

<table>
<thead>
<tr>
<th>Comparison Function</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcmp</td>
<td>5170.30</td>
</tr>
<tr>
<td>sub_1C7EC (strcmp wrapper)</td>
<td>1351.96</td>
</tr>
<tr>
<td>strncmp</td>
<td>1109.73</td>
</tr>
<tr>
<td>strstr</td>
<td>353.93</td>
</tr>
<tr>
<td>memcmp</td>
<td>222.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Score</th>
<th>Static Data</th>
<th>Function</th>
<th>Depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30.23</td>
<td>664225</td>
<td>strcmp</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>2.77</td>
<td>root</td>
<td>strcmp</td>
<td>[664225]</td>
</tr>
</tbody>
</table>

March 7, 2018
Identification of a hard-coded credential backdoor in DVR firmware –
different behaviour for each hardcoded password:

<table>
<thead>
<tr>
<th>Comparison Function</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcmp</td>
<td>1464.70</td>
</tr>
<tr>
<td>strncpy</td>
<td>779.33</td>
</tr>
<tr>
<td>CRYPTO_malloc (FP)</td>
<td>685.10</td>
</tr>
<tr>
<td>_ZNKSs7compareEPKc</td>
<td>376.20</td>
</tr>
<tr>
<td>strstr</td>
<td>306.00</td>
</tr>
<tr>
<td>strcasecmp</td>
<td>196.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Score</th>
<th>Static Data</th>
<th>Function</th>
<th>Depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>171.39</td>
<td>admin</td>
<td>strcmp</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>58.92</td>
<td>ppttzz51shezhi</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
<tr>
<td>3</td>
<td>45.13</td>
<td>6036logo</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
<tr>
<td>4</td>
<td>42.14</td>
<td>6036adws</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
<tr>
<td>5</td>
<td>37.54</td>
<td>6036huanyuan</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
<tr>
<td>6</td>
<td>35.21</td>
<td>6036market</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
<tr>
<td>7</td>
<td>31.05</td>
<td>jiamijiami6036</td>
<td>strcmp</td>
<td>{[admin]}</td>
</tr>
</tbody>
</table>
Web-server with thread running UDP-based service executing user-input commands, unauthenticated as root user:

```
./HumIDIFy model/BayesNet _US_W302RRA_.../bin/httpd
]] HumIDIFy: version 1.0 ,-.
]------------------------|-'
[i] performing feature extraction...
[i] classifying binary...

-> File : _US_W302RRA_.../bin/httpd
-> Profile : webserver (with confidence 100.00%)

[i] checking binary’s functionality...

-> Warning : udp-based api usage detected

-> Judgement : potentially anomalous
Tenda Web-server “Management Service” (cont.)

Web-server with thread running UDP-based service executing user-input commands, unauthenticated as root user:
TrendNet HTTP Authentication with Hard-coded Credentials

HTTP authentication check with comparison against hard-coded credential values:

<table>
<thead>
<tr>
<th>Comparison Function</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcmp</td>
<td>1635.01</td>
</tr>
<tr>
<td>strstr</td>
<td>481.20</td>
</tr>
<tr>
<td>nvram_get (FP)</td>
<td>413.10</td>
</tr>
<tr>
<td>strncmp</td>
<td>265.45</td>
</tr>
<tr>
<td>sub_A2D0 (FP)</td>
<td>131.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static Data</th>
<th>Score</th>
<th>Function</th>
<th>Depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>emptyuserrrrrrrrrrrrrrrrrr</td>
<td>132.17</td>
<td>strcmp</td>
<td>{...}</td>
</tr>
<tr>
<td>emptypassworddddddddd</td>
<td>128.61</td>
<td>strcmp</td>
<td>{...,emptyuserrrrrrrrrrrrrrrr}</td>
</tr>
</tbody>
</table>
Recovery of SOAP-based Command Set

We are also able to recover the command sets of proprietary protocols, in this case a SOAP command set:

<table>
<thead>
<tr>
<th>Comparison Function</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcmp</td>
<td>380.52</td>
</tr>
<tr>
<td>safestrcmp (custom string comparison)</td>
<td>221.00</td>
</tr>
<tr>
<td>strstr</td>
<td>185.00</td>
</tr>
<tr>
<td>strcasecmp</td>
<td>184.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Score</th>
<th>Static Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.64</td>
<td>EnableTrafficMeter</td>
</tr>
<tr>
<td>2</td>
<td>7.64</td>
<td>SetTrafficMeterOptions</td>
</tr>
<tr>
<td>3</td>
<td>7.64</td>
<td>SetGuestAccessEnabled</td>
</tr>
<tr>
<td>4</td>
<td>7.64</td>
<td>SetGuestAccessEnabled2</td>
</tr>
<tr>
<td>5</td>
<td>7.64</td>
<td>SetGuestAccessNetwork</td>
</tr>
<tr>
<td>6</td>
<td>7.64</td>
<td>SetWLANNoSecurity</td>
</tr>
<tr>
<td>7</td>
<td>7.64</td>
<td>SetWLANWPAPSKByPassphrase</td>
</tr>
</tbody>
</table>
Performance
HumIDIFy

- Attribute extraction: 1.31s.
- Classification of single binary: 0.291s (not including time taken to invoke the Java virtual machine).
- Performance of DSL interpreter is dependent upon the complexity of the binary under analysis (number of functions and complexity of those functions): 1.53s on average.
- Time to process an “average” firmware image: 970.61s.
- Performance analysis does not take into account the human factor in final manual analysis.
• Average processing time for a binary: 1.3s.
• Some take longer - depends upon number of functions and CFG complexity:
  Q-See DVR firmware took 46.043 with 15,669 functions.
Backdoor Detection Methodologies
What does our definition/framework tell us about tools/methods?
What does our definition/framework tell us about tools/methods?

- Provides a means to compare them from a high-level perspective without worrying about technical details
What does our definition/framework tell us about tools/methods?

- Provides a means to compare them from a high-level perspective without worrying about technical details
- Provides a basis for developing future backdoor detection techniques
## Reasoning about backdoor detection methodologies

<table>
<thead>
<tr>
<th>Tool</th>
<th>Input source</th>
<th>Trigger</th>
<th>Payload</th>
<th>Privileged State</th>
</tr>
</thead>
<tbody>
<tr>
<td>HumIDIFy</td>
<td>Partial</td>
<td>No</td>
<td>Partial</td>
<td>No</td>
</tr>
<tr>
<td>Stringer</td>
<td>No</td>
<td>Partial</td>
<td>Partial</td>
<td>No</td>
</tr>
</tbody>
</table>
HumIDIFy

- Input source: what should/should not exist?
- Payload: what should/should not exist that might put us in a privileged state?
• Trigger: what possible hard-coded values can be used?
• Payload: which comparisons are likely to lead to a privileged state?
Conclusion

- Provide a means of reasoning about backdoors:
  - Componentisation of their key features.
  - Distinguish them from *accidental* vulnerabilities.
Conclusion

- Provide a means of reasoning about backdoors:
  - Componentisation of their key features.
  - Distinguish them from *accidental* vulnerabilities.
- Present two tools implementing different methodologies for detecting backdoors in IoT devices:
  - Runtime of both tools satisfies lightweight property: each tool takes seconds to perform analysis.
  - Successfully identified a number of (real-world) backdoors and instances of undocumented functionality.
The Internet of Backdoors

Sam L. Thomas
(joint work with Aurélien Francillon, Flavio D. Garcia & Tom Chothia)

1 University of Birmingham
United Kingdom

2 EURECOM
France

March 7, 2018