**Journal papers:**

# **Title:** FuzzDocs: An Automated Security Evaluation Framework for IoT(IEEE Access Volume 10)

Problem:

Lack of formal documentation in IoT devices used worldwide leading to difficulties in using simple fuzzing tests.

Described Solution:(Methodology)

Addressing IoT Security concerns by building an automated tool to pre-emptively detect vulnerabilities.

The solution described uses a method known as fuzzing, this method involves testing against large amounts of data to ascertain the reliability of IoT security against various attacks.

The project creates a simplified framework to test fuzzing by having vendors create API documentation as basic guidelines to ensure fuzzing tests only “valid-enough” inputs, massively improving test times.

The project uses NLP to convert documentation into a framework for testing.

Issues mitigated:

Firmware based fuzzing:

The flaw in firmware based fuzzing is the fact that most companies fail to reveal their communication API’s and disable/do not keep debugging ports, this makes device firmware impossible to extract automatically.

Network based fuzzing:

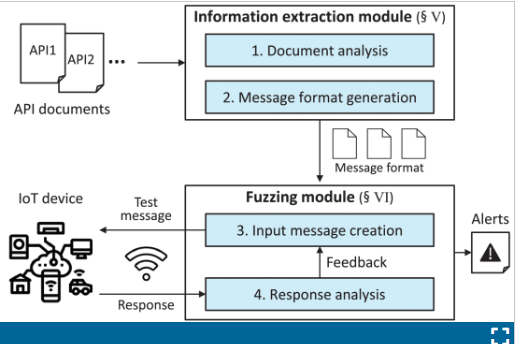
Network based fuzzing uses randomly generated messages in order to test the device’s security measures. However, the devices tend to be selective with inputs making the brute force method, rather hard and time consuming to achieve.

Companion app based fuzzing:

This involves testing with the companion app. However, the companion app may not invoke all of the device’s API’s leading to imperfect security coverage.

Design:

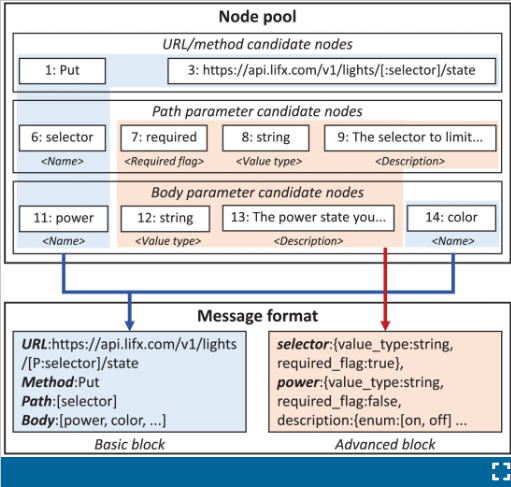
1. Automating extraction: Extraction from documentation
2. Creating Effective Message Formats: Automated message creation
3. Monitoring the device: Automated service to monitor device state



[**I have skipped the next few pages as it describes building an NLP algorithm for reading the documents**]

Message Generation:

The message format is extracted from documentation on which fuzzing is tested.

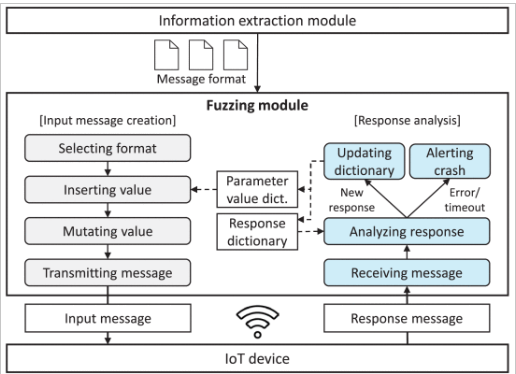
****

Once the documents are extracted and a message body is created, fuzzing can be performed with various parameters to test device resilience.

This part consists of two subparts:

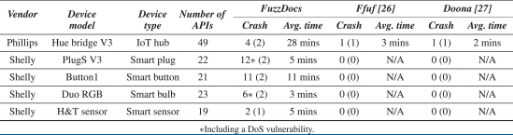
* Basic block: Essentials of an input message: analysing candidates for URL/ HTTP methods
* Advanced block: Pertains to flags, parameters, values.

Fuzzing:



* Input message creation: The module uses default parameters{All available}, it then edits out some, based on the advanced block. A dictionary is created and referenced to prevent multiple messages and unnecessary effort. It chooses some values to mutate before sending the packets.
* Response Analysis: Post sending the message, there are expected outcomes:
  + Success
  + Timeout
  + Connection Loss
  + Internal server error
* Coverage feedback mechanism: Uses feedback mechanisms to localise errors

Testing:

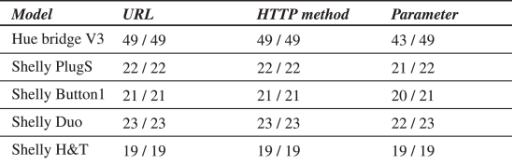


The testing revealed that Fuzzdocks created a much more thorough testing for the various devices tested. This led to better frameworks than other related fuzzing frameworks.

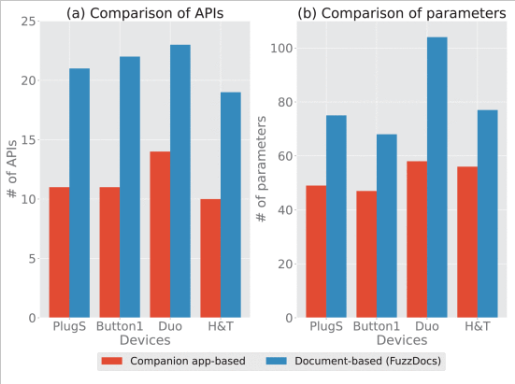
* **Doona** : Doona, an extended version of Bruteforce Exploit Detector (BED) designed to find memory-related bugs such as buffer overflows and format string bugs in network protocol implementations. It finds vulnerabilities by mutating general request/response packets without seed messages.
* **Ffuf** : A popular network application fuzzer that supports various protocols, including HTTP and HTTPS. Unlike Doona, Ffuf receives a predefined word list to mutate request messages. We used the word list provided by Ffuf’s official website.

Results:

Accuracy of Message Format Generation:



Coverage of API:



Disadvantages:

* Can only process HTML documents
* Deals with only HTTP based API’s

Conclusion:

Fuzzdocks provides a significantly better coverage against potential weaknesses compared to most other fuzzing techniques.

This can be extended to work over the disadvantages mentioned and create a complete fuzzing test solution for IoT.

Inferences:

This is a thorough IoT Attack testing model. It goes through very broad attack methodologies without getting into the specifics of it.

However, the methods for packet construction and fuzzing could be implemented while searching for alternate weaknesses in IoT devices.

# **Title:** An Automated and Comprehensive Framework for IoT Botnet Detection and Analysis (IoT-BDA)

Problem:

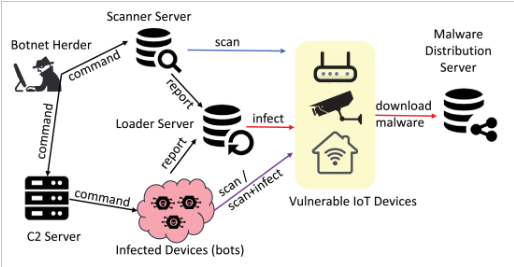
IoT botnet attacks being difficult to detect on the go

Proposed solution:

IoT-BDA Framework, for automated capturing, analysis and identification and reporting of botnets using honeypots integrated with a novel sandbox.

Hurdles:

* Compromised device might not show signs of an infection
* Detection requires specialized tools
* IoT devices do not fit under the same framework as host-based IoT devices
* Operation of a botnet
  + Centralized or p2p
  + Can be accessed by weak authentication or remote code execution
* Botnet detection:
  + Have a specific pattern
  + Uses NIDS/HIDS
  + Conventional methods may not detect zero day exploits
* Identifying features to improve effectiveness
* Botnet infections diversities
* Anti-honeypot/Anti-Analysis



Methodology:

Samples are connected in two ways:

Forensic analysis of infected devices

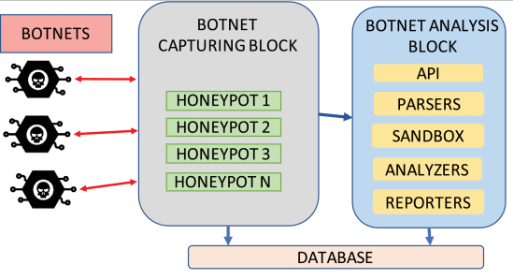
Using software that simulates a vulnerable IoT device, a honeypot

The sample is saved as a ELF File

Framework:

* Real-time botnet capturing, analysis,identification and report sharing
* Identification of features for effective botnet detection, analysis as well as infection remedy such as indicators of compromise and attack
* Facilitation of botnet suspension through integration with a blacklist and abuse report system
* Support for a wider range of hardware and software configurations

IoT-BDA Architecture:



Botnet Capturing Block:

Comprises multiple honeypots simulating different vulnerabilities to maximise samples collected

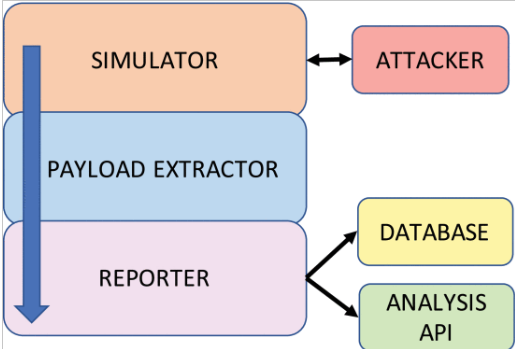
Botnet Analysis Block:

Analyses Botnet Samples, consists of an API,Parsers, a sandbox, analysers and reporters

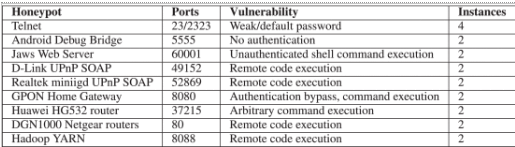
* Facilitating botnet detection: Uses 3 analysers, static, behavioral and network
* Identifying anti-analysis and anti forensic techniques
* Identifying botnet types and variants

P-O-C implementation:

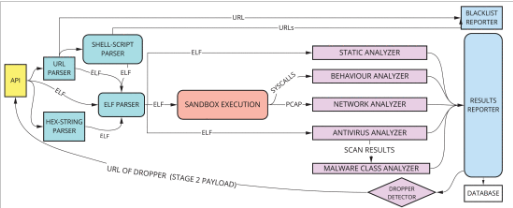
Honeypot design:



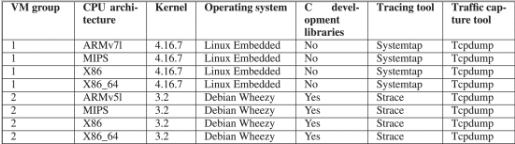
Honeypot deployment:



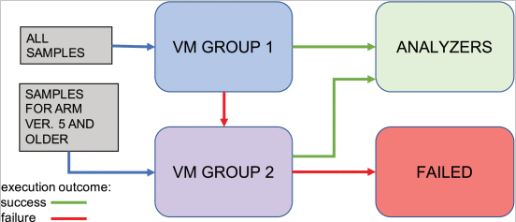
Botnet Analysis block:



VM’s used by sandbox:



Sandbox execution:



Static Analyser:

Analyses without execution

Behavioural Analyser:

Identifies features that may not be analysed by static analysis

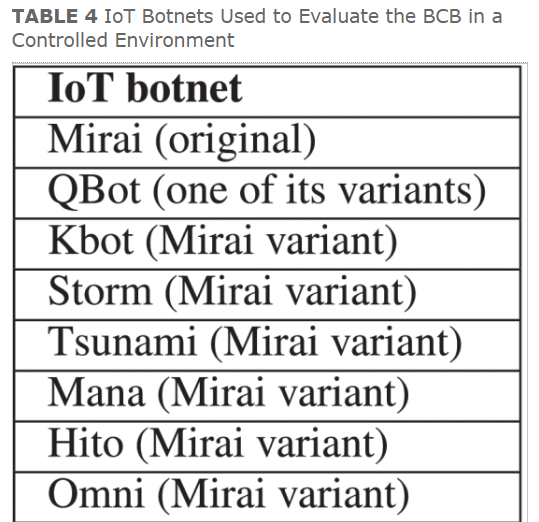
Network Analyser:

Analyses network traffic

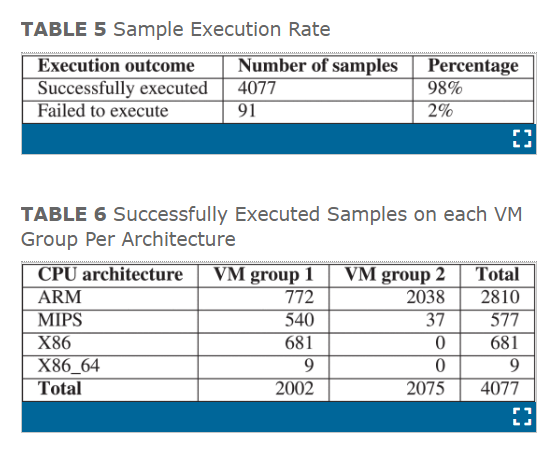
Performance evaluation:

* Capturing botnet samples:

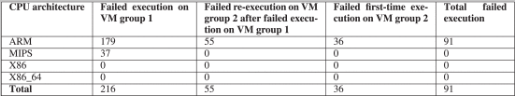
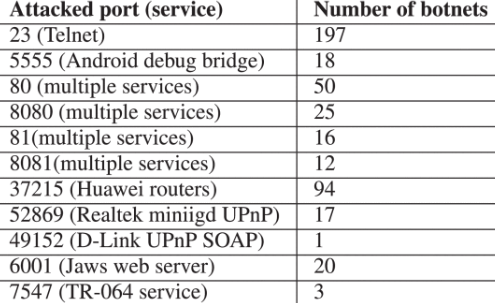




* Botnet Sample analysis: Successful analysis of 98% of data



Samples that failed:

Advantages:

* Faster than conventional botnet detection methods
* More inclusive honeypots

Disadvantages:

* More complicated than conventional means
* Anti-honeypot techniques not fully beaten

Conclusion:

* Design and implementation of the IoT-BDA framework
* List of anti-honeypot techniques
* 4077 unique IoT botnet samples
* Analysis of the captured samples