Johns Hopkins Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099

RUCKUS: A Cybersecurity Engine for Performing Autonomous Cyber-Physical System Vulnerability Discovery at Scale

September 22, 2020

Bradley Potteiger, Jacob Mills, Daniel Cohen, Paul Velez
The Johns Hopkins University Applied Physics Laboratory
Laurel, MD

Cyber-Physical Systems are NOT Secure

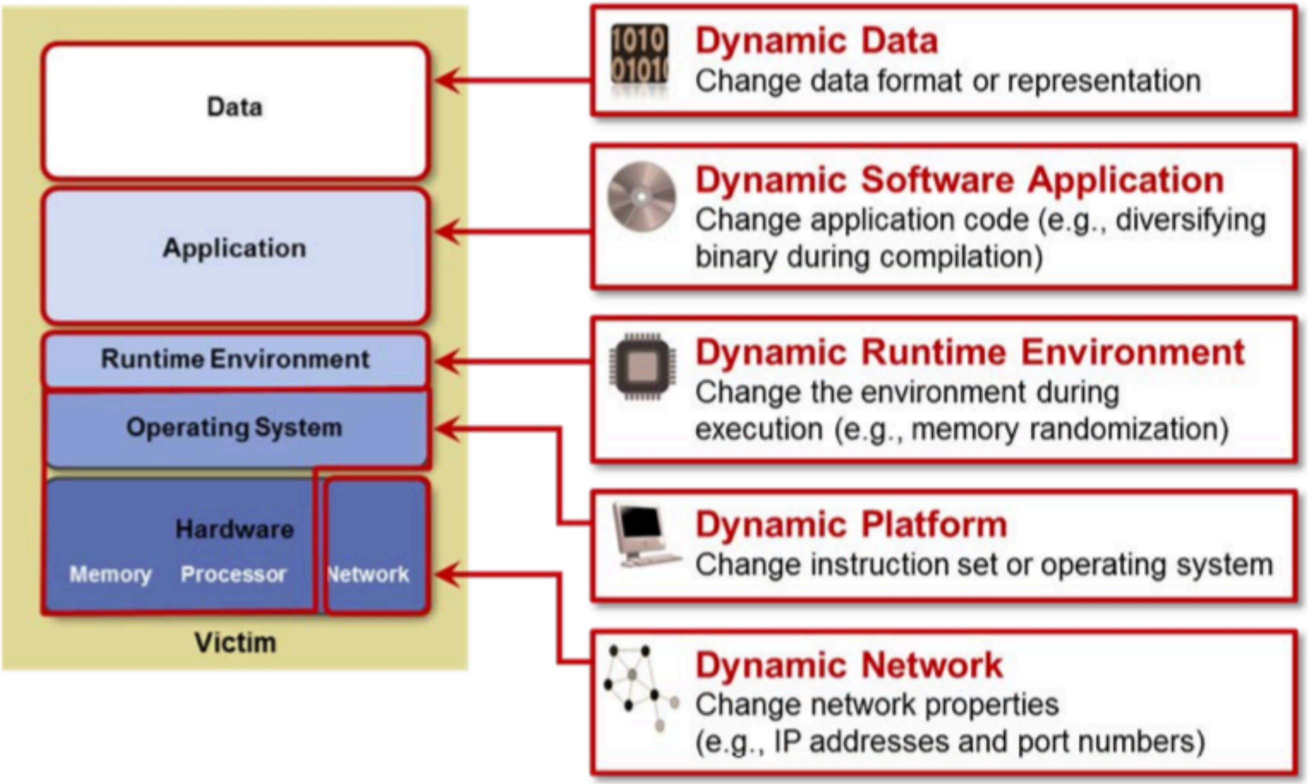
- CPS-IoT are increasingly subjected to sophisticated cyber-attacks

- Several high profile autonomous vehicle accidents demonstrate the tightly coupled nature between the software and physical dynamics

- CPS not only have to maintain integrity while under cyber attacks, but also need to ensure safe behavior and operation

Moving Target Defenses

- Network
  - Software Defined Networking
- Application
  - Instruction Set Randomization
  - Address Space Randomization
  - Data Space Randomization
- Data
  - Database Sharding



http://web.mit.edu/br26972/www/pubs/mt_survey.pdf

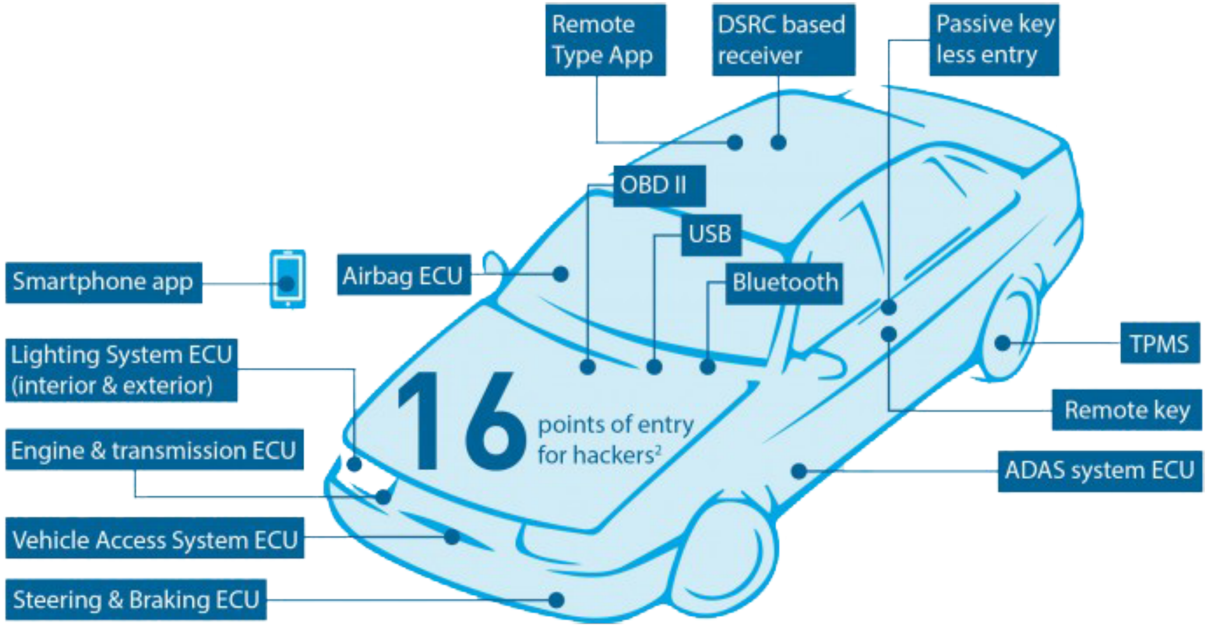
Shifting from Defense to Offense

- DARPA Cyber Grand Challenge
  - Autonomous Capture the Flag Competition in 2016
  - Led to development of and interest in autonomous reverse engineering and exploitation tools within academia, government, and industry (For All Secure, Angr, McSema, Ghidra, etc.)
  - Competition architecture was limited in scope, new problems emerge when looking at scaling approaches to the **REAL WORLD**

- JHU APL
  - 7,000 Employees in Laurel, MD
  - Embedded reverse engineering SMEs
  - Projects often emerge unpredictably with tight deadlines

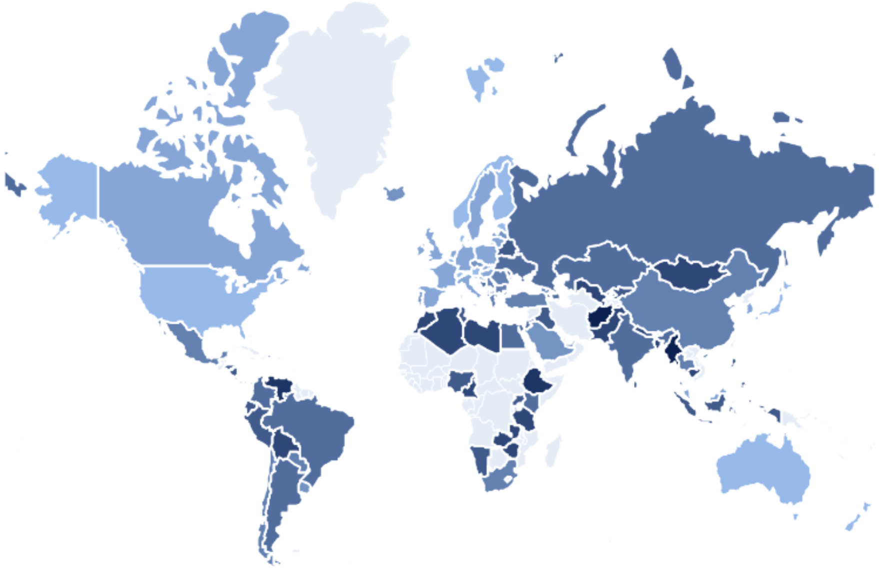
Automotive Security

- Vehicle Statistics
  - 150 Million connected vehicles by 2020
  - 70 ECUs
  - 100 Million lines of code
- Significant Vulnerabilities
  - ECU Legacy Code
  - Connection of non-critical systems to safety-critical network
  - Unprotected communications
- Memory Corruption
  - Code Injection
  - Code Reuse
  - Non-Control Data



Threat Model

- System
  - CPS Automotive Firmware
  - Communication Interface
  - Security through Obscurity Approach
- Vulnerability
  - Memory corruption vulnerability in CPS controller
  - Common software
  - Millions of same model around the world



Problem Formulation

Background

- Proprietary software currently leverages a security through obscurity approach
- There is a large set of previously discovered vulnerability data within open source software and previously reverse engineered proprietary software
- Proprietary software often relies upon open source libraries
- Most impactful vulnerabilities seem to be most common and simplest

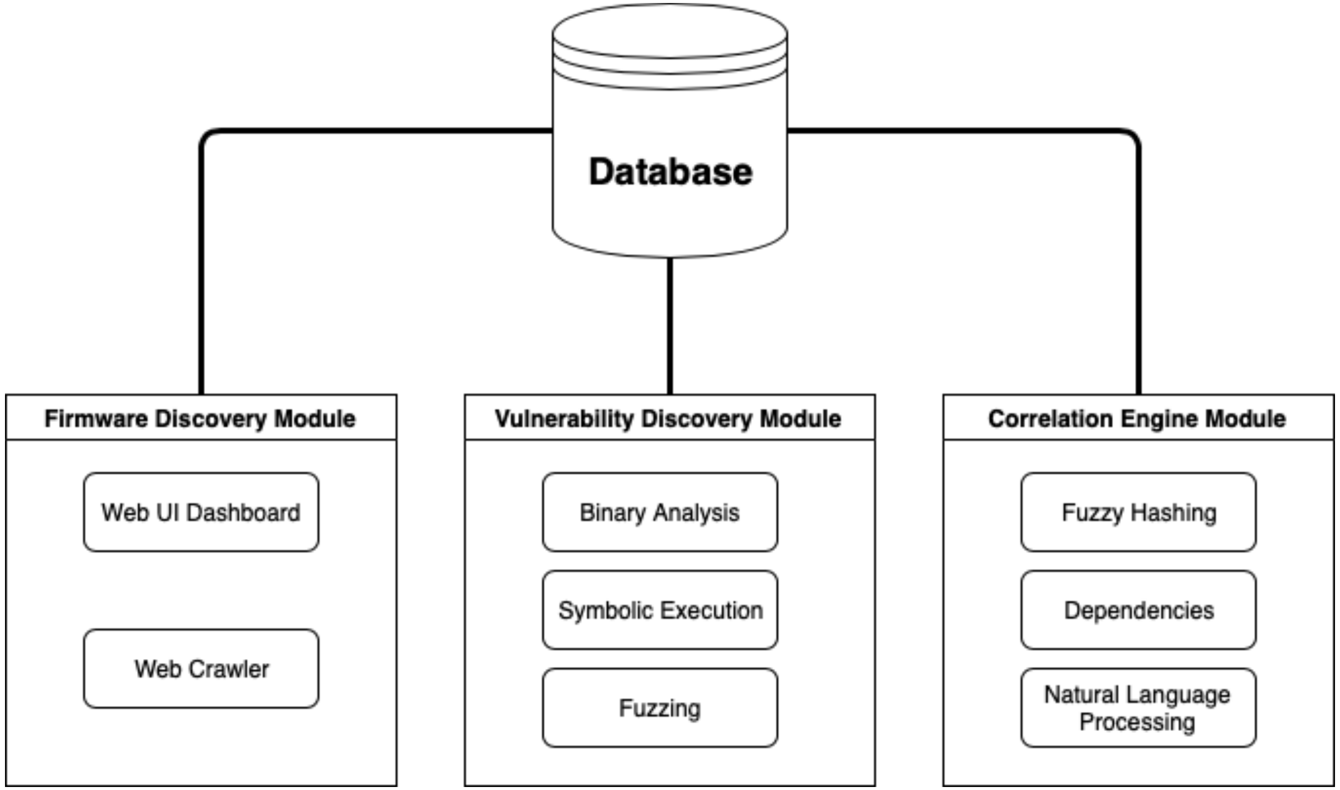
Problem

- How do you speed up the time to reverse engineer mission critical systems?
- How similar and at risk is proprietary software to open source library vulnerabilities?

Hypothesis: Leveraging software similarity as a heuristic can significantly speed up time to reverse engineer and exploit proprietary software.

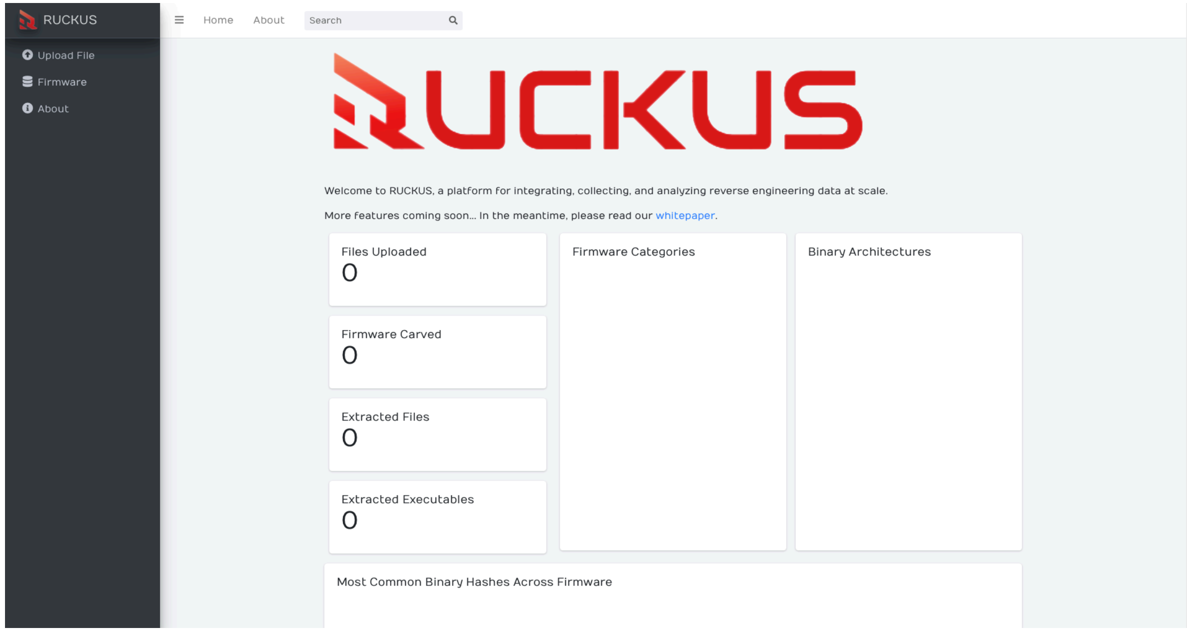
Ruckus Architecture

- Hybrid Human + Autonomous Approach
  - Human expertise + in depth analysis
  - Autonomous scalability
- Software similarity heuristic
  - Similar firmware will contain similar vulnerabilities
  - Centralized location to reuse previously discovered vulnerabilities
  - Should start with lowest hanging fruit first



Firmware Discovery Module

- Input
  - Manual Input
  - Web Crawler
- Filesystem is carved to accumulate all files and libraries of interest
- Output
  - Set of binary files
  - Firmware properties



Vulnerability Discovery Module

- Hybrid approach
  - Manual – Fine grained inspection
  - Autonomous – Rapid high level analysis
- Binary Analysis
  - Disassembly
  - Control flow graph generation
  - Metadata extraction
- Symbolic Execution
  - Angr
- Fuzzing
  - Targeted approach with symbolic execution results fed as input

Correlation Engine Module

- Fuzzy Hashing
  - Binary signatures
  - Vulnerabilities
- Dependencies
  - Shared libraries
- Natural Language Processing
  - Filenames
  - Symbol and function names

Algorithm 1 Compute correlation between binaries

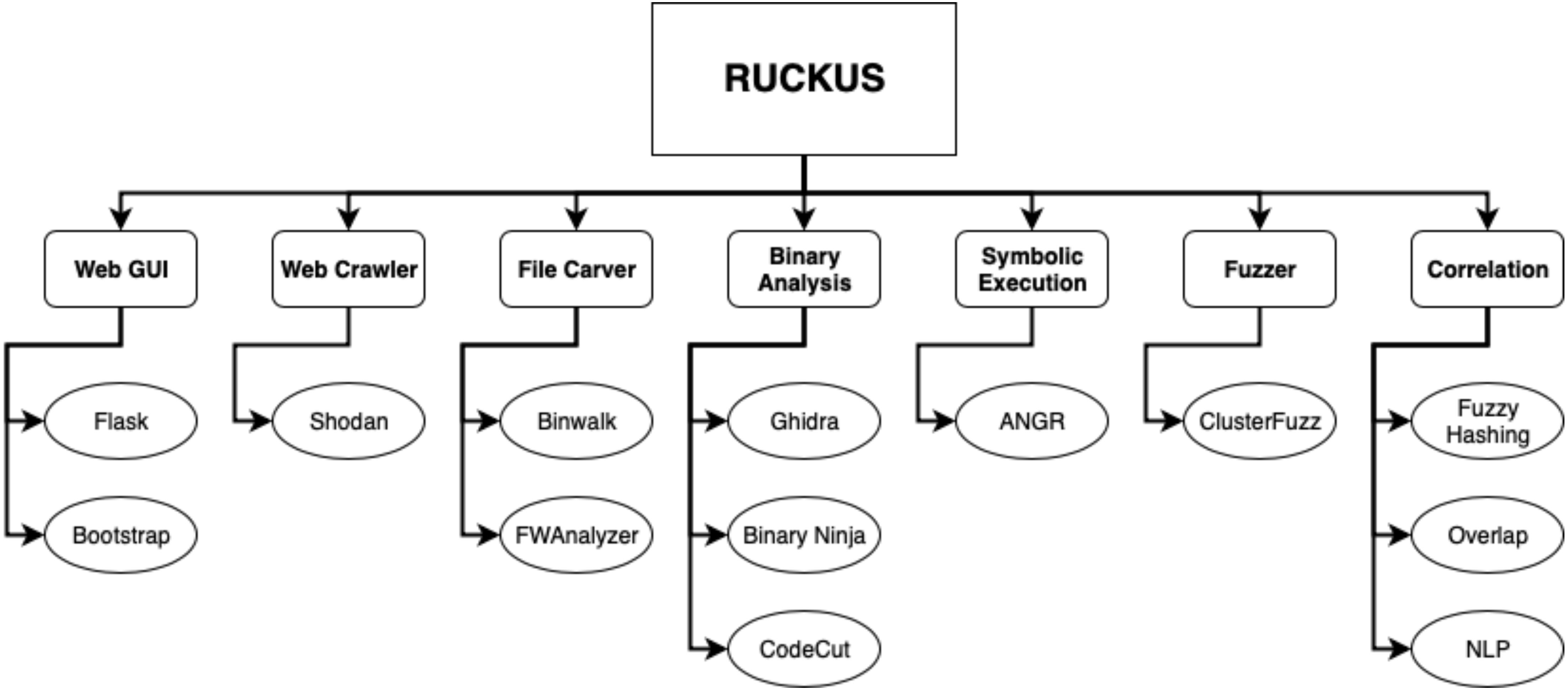
Require: Files (F) $\subseteq$ Binary Files (β) $\subseteq$ {Executable, Library}
Require: Comparators (C) $\subseteq$ {Vulns, Dependencies, Signatures, Fuzzy Hash}
Require: Target Firmware (TF) $\subseteq \beta_{TF} \subseteq C_{TF}$
Require: Dataset (D) $\subseteq Firmware_D \subseteq \beta_D \subseteq C_D$

```
Matches List ML
Binary Files BM
for all File F in TF do
    if F.Type ⊇ β then
        Vulns_F = findVulns(F)
        Deps_F = findDeps(F)
        Sigs_F = findSigs(F)
        Hash_F = computeHash(F)
        F.comps= {Vulns_F , Deps_F , Sigs_F , Hash_F}
        BM.append(F)
    end if
end for
for all Firmware Firm in D do
    MatchScore score_ba , score_sigs, score_hash, totalscore
    counter=0
    for all File Fcur in Firm do
        if F.Type ⊇ β then
            counter+=1
            Vulns_Fcur = findVulns(Fcur)
            Deps_Fcur = findDeps(Fcur)
            Sigs_Fcur = findSigs(Fcur)
            Hash_Fcur = computeHash(Fcur)
            score_ba = findOverlap(BM, Vulns_Fcur, Deps_Fcur)
            score_sigs = findOverlap(BM, Sigs_Fcur)
            score_hash = findOverlap(BM,Hash_Fcur)
            filescore = (score_ba + score_sigs + score_hash) / 3
            totalscore += filescore
        end if
    end for
    Match Score firmMatchScore = totalscore / counter
    Match m = {Firm_TF , Firm, firmMatchScore}
    ML.append(m)
end for
```

Database

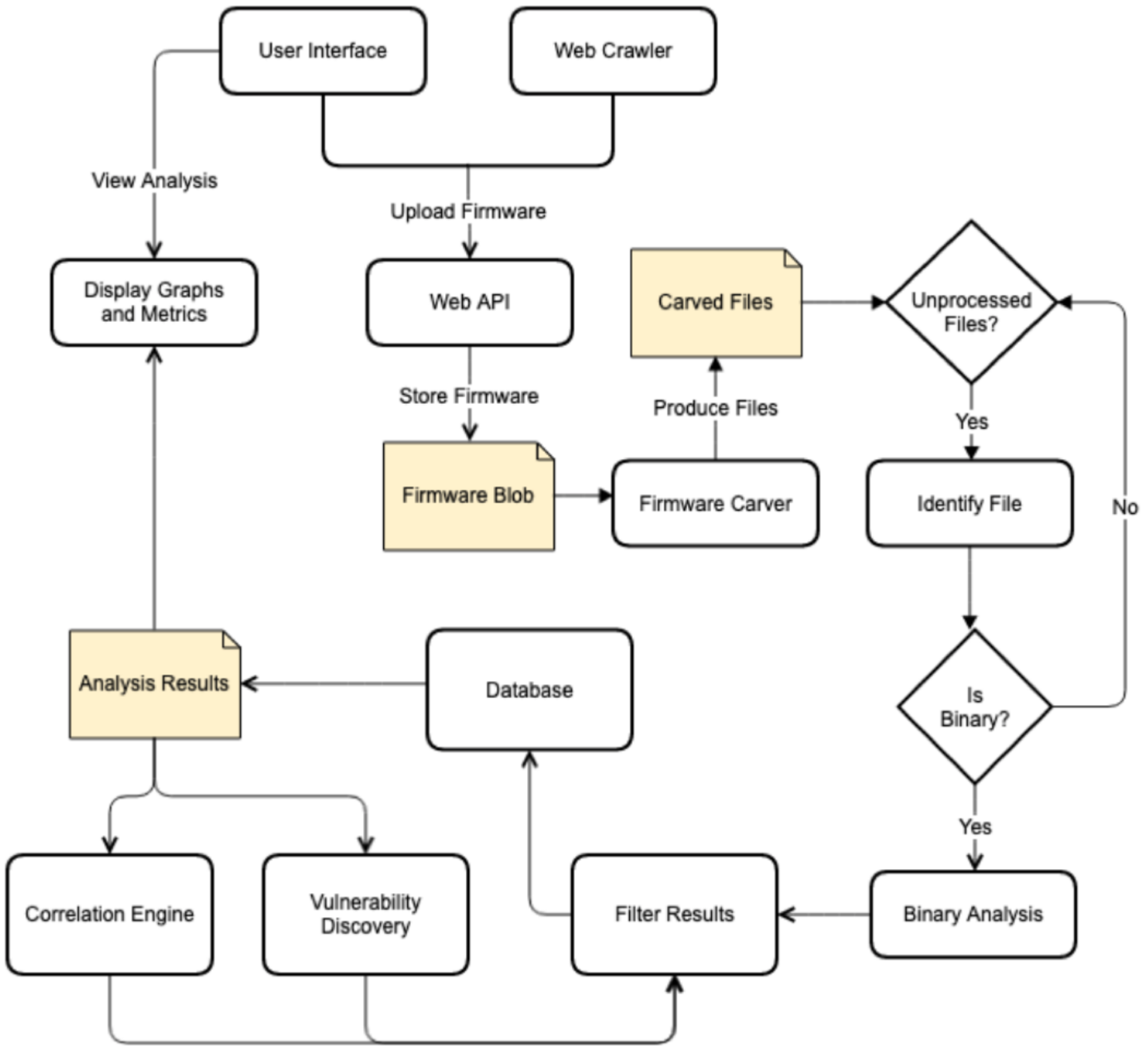
- Hybrid Graph and Relational
  - Graph – Stores high level relations
    - Firmware similarity
    - File dependencies
  - Relational – Stores binary blobs and content
    - Vulnerabilities
    - Signatures
- Speeds up lookup time

Implementation



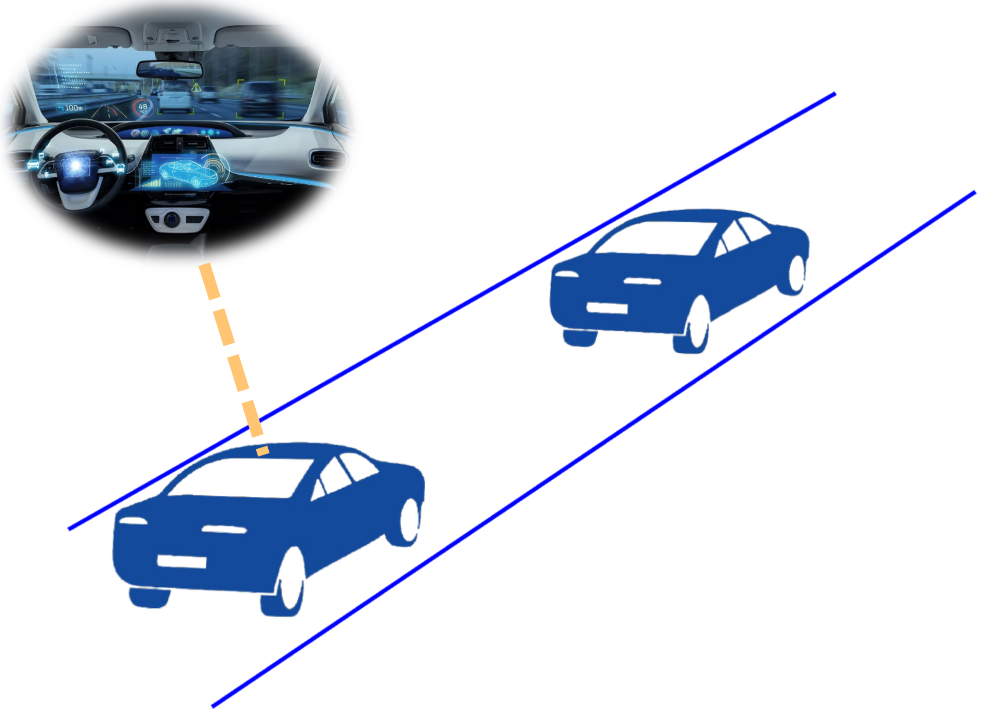
Process Flow

- Collect firmware images and carve binary files of interest
- Perform binary analysis to find relevant symbols, properties, and dependent libraries
- Store binary analysis results in hybrid graph-relational database
- Fetch vulnerability and correlation information to identify most likely vulnerabilities to search for
- Perform a more thorough manual vulnerability discovery process and update database



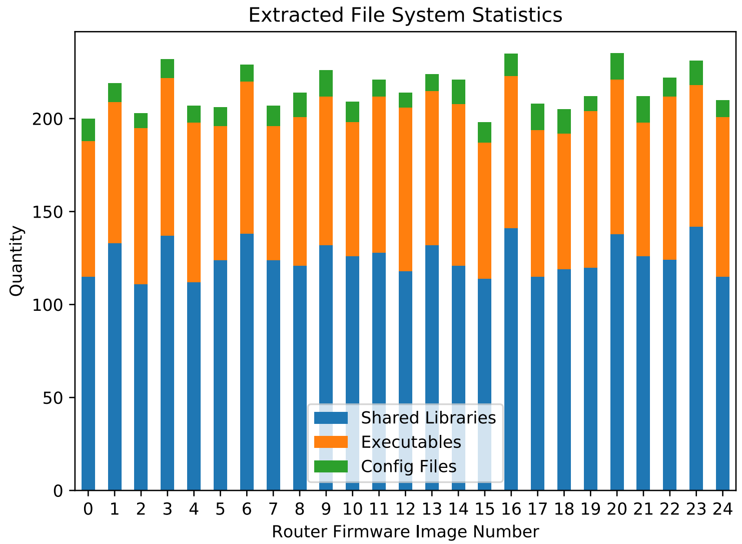
Evaluation

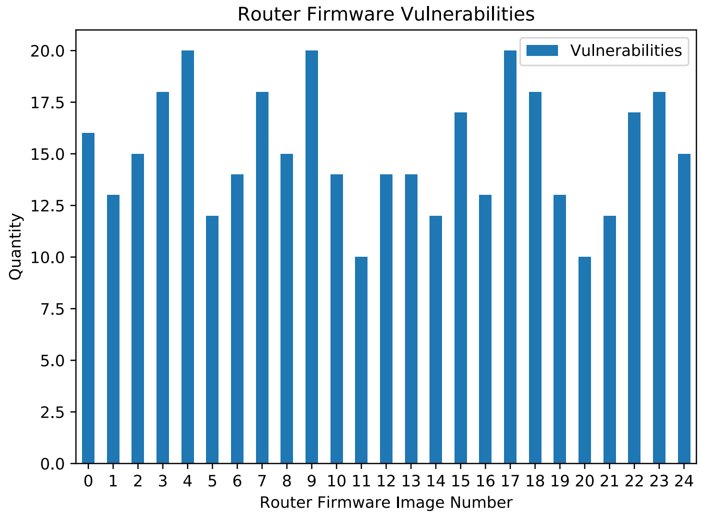
- Mission
    - Rapidly reverse engineer adversary automobiles
    - Discover potentially exploitable vulnerabilities for war fighter mission
    - Deliverables must be done within a day
- Firmware Dataset
    - 5 commercial automotive firmware images
    - 20 open source firmware images
- Scenario
    - Assume no knowledge of automotive firmware
    - Starting with knowledge of vulnerabilities in open source router firmware



Router Firmware Descriptive Statistics

- 5 brands of routers
  - Cisco
  - Belkin
  - Liksys
  - DD-WRT
  - Netgear
- 3 types of vulnerability locations
  - Shared libraries
  - Configuration files
  - Executables

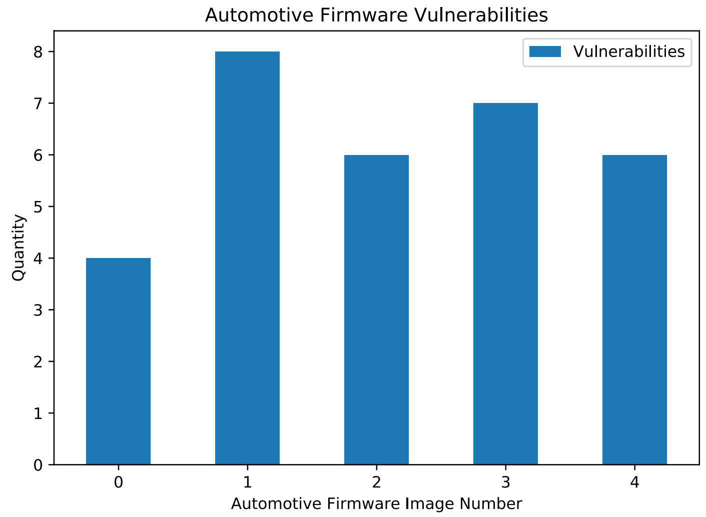




Automotive Correlation Statistics

- 5 Automobile Vendors
  - Millions of vehicles globally
- Correlation Metric
  - Fuzzy Hashing
  - Similar file names
  - Similar symbol names
- Discovered Vulnerabilities
  - Memory corruption
  - Web App
- Time to Discovery
  - Human only – 8 days
  - Ruckus – 1.5 hours

Firmware Matching Scores (x100)					
Router ID	**Auto1**	**Auto2**	**Auto3**	**Auto4**	**Auto5**
Cisco '17	49	32	31	29	21
Belkin '16	51	38	29	36	24
Linksys '17	46	43	38	39	22
DD-WRT '19	52	32	36	44	29
Cisco '16	48	48	41	45	48
Belkin '15	58	60	46	45	41
Linksys '16	59	51	54	58	39
DD-WRT '06	62	58	53	49	51
DD-WRT '08	55	53	56	53	53
Belkin '14	49	51	47	51	50
DD-WRT '13	50	53	51	48	49
DD-WRT '17	48	48	47	48	48
Linksys '18	47	51	44	45	43
DD-WRT '18	42	44	43	42	44
Netgear '10	41	40	41	42	41
Netgear '12	36	35	38	41	30
Netger '14	42	37	36	32	31
DD-WRT '20	31	34	39	31	31
Linksys '19	29	31	30	29	29
Netgear '16	23	22	26	27	23
Belkin '18	25	20	26	23	21
Cisco '18	12	21	24	18	20
Netgear '18	20	16	15	14	12
Netgear '19	23	21	20	11	14
Netgear '20	18	14	15	16	17



Conclusion

- Human fine grained inspection + autonomous correlation and vulnerability discovery provides a comprehensive first pass to rapidly discovery vulnerabilities in proprietary
- Ruckus significantly decreases time to vulnerability discovery versus a traditional human only approach
- There is a significant correlation between proprietary automotive firmware and open source router firmware
  - Security through obscurity is no longer effective
  - More active and dynamic defenses are necessary
  - Software needs to be more unique

Questions?