GRAMMATECH

Repairing COTS Router Firmware without Access to Source Code or Test Suites

A Case Study in Evolutionary Software Repair

Eric Schulte
eschulte@grammatech.com
July 12, 2015

NETGEAR Exploit


www.pcworld.com/article/2057260/vulnerabilities-in-some-netgear-router-and-nas-products-open-door-to-remo

PCWorld Macworld TechHive Subscribe

26 Like 47 Tweet 3 8+1

SECURITY netgear, routers

Vulnerabilities in some Netgear routers open door to remote attacks

Lucian Constantin, IDG News Service

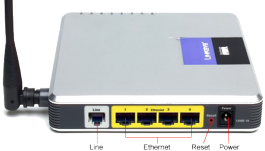
Oct 23, 2013 10:30 AM

1. URL starting with `BRS` bypasses authentication
2. URL including `unauth.cgi` or `securityquestions.cgi` bypass authentication
3. unprotected page removes authentication for every page `http://router/BRS_02_genieHelp.html`

Common Vulnerability

NETGEAR WNDR4700

D-Link DIR-100



Linksys WAG200G

GRAMMATECH

Contributions

Previous evolutionary repair Requirements

- Un-stripped ELF file
- Regression test suite
- Fault localization information

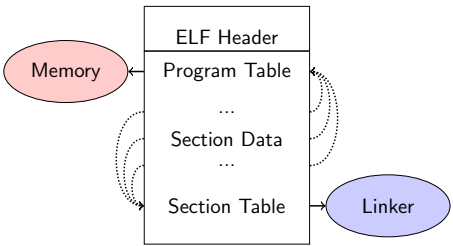
Contributions

Reduced evolutionary repair Requirements

- ~~Un-stripped~~ *arbitrary* ELF file
- ~~Regression test suite~~
- ~~Fault localization information~~

ELF Mutation

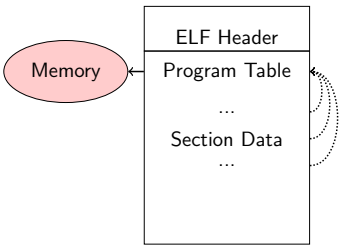
ELF File



Mutate code in `.text` section.

ELF Mutation

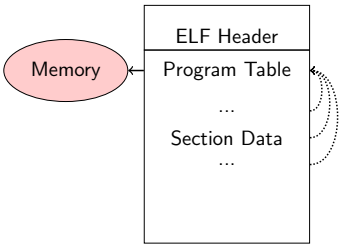
ELF File



Mutate all `PT_LOAD` type sections.

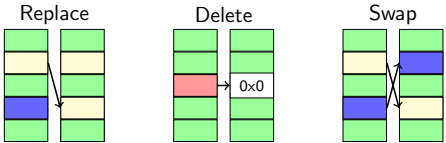
ELF Mutation

ELF File



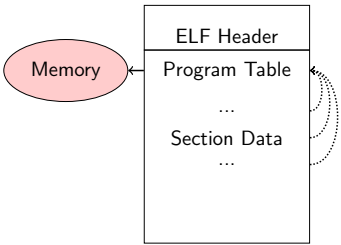
Mutate all `PT_LOAD` type sections.

MIPS Mutations



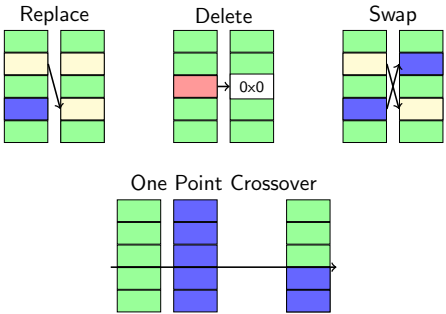
ELF Mutation

ELF File



Mutate all `PT_LOAD` type sections.

MIPS Mutations



Evolutionary Algorithm

Input: Vulnerable Program, original : *ELF*
Input: Test Suite, suite : [*ELF* → *Fitness*]
Parameters: *populationSize*, *tournamentSize*, *crossRate*
Output: Patched version of Program

1: **let** *fitness* ← evaluate(original, suite)
2: **let** *pop* ← populationSize copies of ⟨original, fitness⟩
3: **repeat**
4: **if** Random() < *CrossRate* **then**
5: **let** p_1 ← tournament(pop, tounamentSize, +)
6: **let** p_2 ← tournament(pop, tounamentSize, +)
7: **let** p ← crossover(p_1, p_2)
8: **else**
9: *p* ← tournament(pop, tounamentSize, +)
10: **end if**
11: **let** p' ← Mutate(*p*)
12: **let** *fitness* ← evaluate(suite, p')
13: incorporate(*pop*, ⟨p', Fitness(Run(p'))⟩)
14: **if** length(pop) > maxPopulationSize **then**
15: evict(pop, tournament(pop, tounamentSize, −))
16: **end if**
17: **until** fitness > length(suite)
18: **return** p'

Properties

- Parameters

pop-size	512
tourny-size	2
crossover	$\frac{2}{3}$
fitness evals	$\sim 10,000$

Evolutionary Algorithm

Input: Vulnerable Program, original : *ELF*
Input: Test Suite, suite : [*ELF* → *Fitness*]
Parameters: *populationSize*, *tournamentSize*, *crossRate*
Output: Patched version of Program

1: **let** *fitness* ← evaluate(original, suite)
2: **let** *pop* ← populationSize copies of ⟨original, fitness⟩
3: **repeat**
4: **if** Random() < *CrossRate* **then**
5: **let** p_1 ← tournament(pop, tounamentSize, +)
6: **let** p_2 ← tournament(pop, tounamentSize, +)
7: **let** p ← crossover(p_1, p_2)
8: **else**
9: p ← tournament(pop, tounamentSize, +)
10: **end if**
11: **let** p' ← Mutate(p)
12: **let** *fitness* ← evaluate(suite, p')
13: incorporate(*pop*, ⟨p', Fitness(Run(p'))⟩)
14: **if** length(pop) > maxPopulationSize **then**
15: evict(pop, tournament(pop, tounamentSize, −))
16: **end if**
17: **until** fitness > length(suite)
18: **return** p'

Properties

- Parameters

pop-size	512
tourny-size	2
crossover	$\frac{2}{3}$
fitness evals	$\sim 10,000$

- Steady State
  - reduced memory consumption
  - simplifies parallelization
  - exploitative

Evolutionary Algorithm

Input: Vulnerable Program, original : *ELF*
Input: Test Suite, suite : [*ELF* → *Fitness*]
Parameters: *populationSize*, *tournamentSize*, *crossRate*
Output: Patched version of Program

1: **let** *fitness* ← evaluate(original, suite)
2: **let** *pop* ← populationSize copies of ⟨original, fitness⟩
3: **repeat**
4: **if** Random() < *CrossRate* **then**
5: **let** p_1 ← tournament(pop, tounamentSize, +)
6: **let** p_2 ← tournament(pop, tounamentSize, +)
7: **let** p ← crossover(p_1, p_2)
8: **else**
9: p ← tournament(pop, tounamentSize, +)
10: **end if**
11: **let** p' ← Mutate(p)
12: **let** *fitness* ← evaluate(suite, p')
13: incorporate(*pop*, ⟨p', Fitness(Run(p'))⟩)
14: **if** length(pop) > maxPopulationSize **then**
15: evict(pop, tournament(pop, tounamentSize, −))
16: **end if**
17: **until** fitness > length(suite)
18: **return** p'

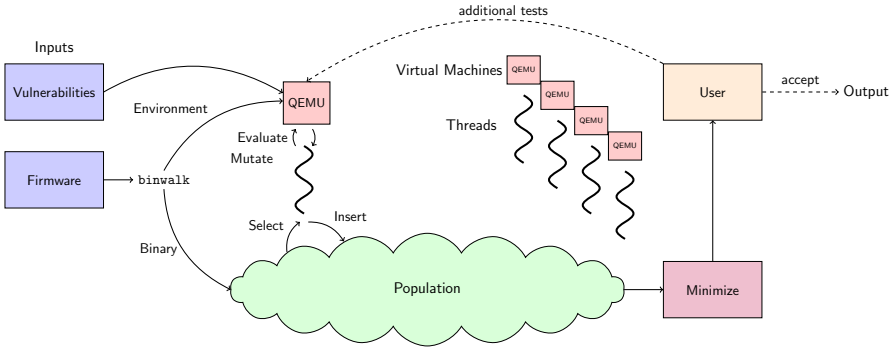
Properties

- Parameters

pop-size	512
tourny-size	2
crossover	$\frac{2}{3}$
fitness evals	$\sim 10,000$

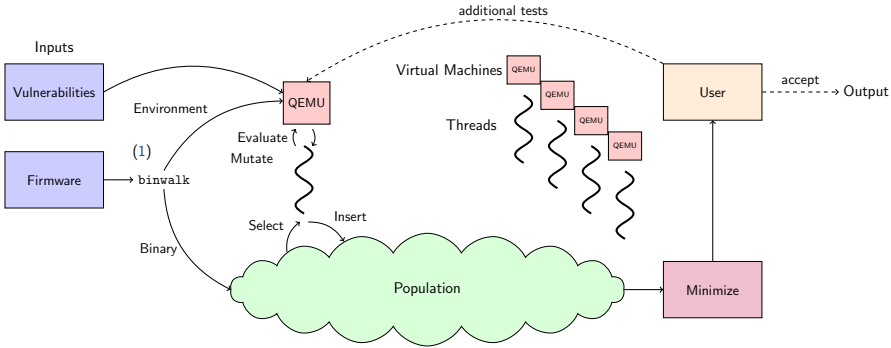
- Steady State
  - reduced memory consumption
  - simplifies parallelization
  - exploitative
- Eviction & selection use tournaments

NETGEAR Repair Overview



Repair process

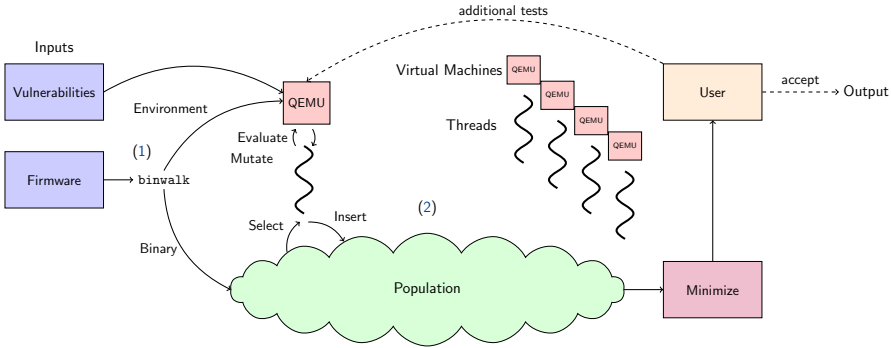
NETGEAR Repair Overview



Repair process

1. Extract binary

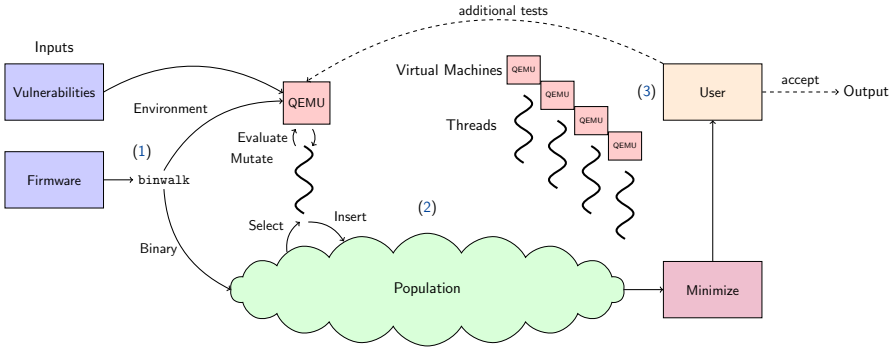
NETGEAR Repair Overview



Repair process

1. Extract binary, 2. Evolve repair

NETGEAR Repair Overview



Repair process

1. Extract binary, 2. Evolve repair, 3. Interactive review

Evolved repairs

Repair runtime, size, and impact of minimization

Run	Fit Evals	Full Diff	Min Diff	Full Fit	Min Fit
0	90405	500	2	8	22
1	17231	134	3	22	22
2	26879	205	2	21	22
3	23764	199	2	19	22
4	47906	319	2	6	6
5	13102	95	2	16	22
6	76960	556	3	17	22
7	11831	79	3	20	22
8	2846	10	1	14	14
9	25600	182	2	21	22
mean	33652.4	227.9	2.2	16.4	19.6

Evolved repairs

Repair runtime, size, and impact of minimization

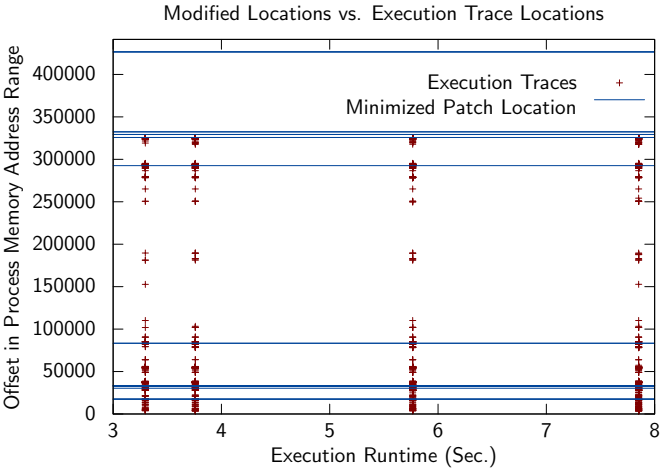
Run	Fit Evals	Full Diff	Min Diff	Full Fit	Min Fit
0	90405	500	2	8	22
1	17231	134	3	22	22
2	26879	205	2	21	22
3	23764	199	2	19	22
4	47906	319	2	6	6
5	13102	95	2	16	22
6	76960	556	3	17	22
7	11831	79	3	20	22
8	2846	10	1	14	14
9	25600	182	2	21	22
mean	33652.4	227.9	2.2	16.4	19.6

evaluations $\sim 36,000$, runtime 86.6 minutes, threads 32

GRAMMATECH

Fix locations

Code modification and fault localization



Conclusion

Contribution and impact

Contributions

EC repair

- Without a regression test suite
- Without fault localization
- Leveraging user interaction
- Of embedded firmware
- Of a real-world exploit
- Of multiple vulnerabilities

Conclusion

Contribution and impact

Contributions

EC repair

- Without a regression test suite
- Without fault localization
- Leveraging user interaction
- Of embedded firmware
- Of a real-world exploit
- Of multiple vulnerabilities

Enables

- Repair of COTS binaries
  - Users
  - Security researchers
- Repair of data sections
- Interactive improvement

Questions or Comments?

eschulte@grammatech.com