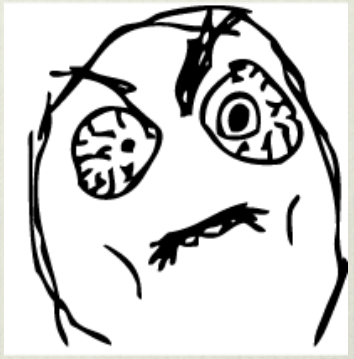
Evolving Exploits through Genetic Algorithms

By soen

Who am I

- CTF Player
- Programmer
- Virus / Worm Aficionado
- Computer Scientist
- Penetration Tester in daylight



Exploiting Web Applications

- Attack problems
  - Driven by customer
  - Small scope
  - Limited time
  - Report driven
- Attack methodology

Exploiting Web Applications

- Attack problems
- Attack methodology
  - Run as many scanning tools as possible
  - Manually poke at suspicious areas until a vulnerability is found
  - Write an exploit

Exploiting Web Applications

- Attack problems
- Attack methodology
- Problems with this
  - Manual code coverage is inherently small
  - Manual inspection of suspicious areas is time-costly
  - Manual exploit development takes time

Existing tools for exploit discovery / development

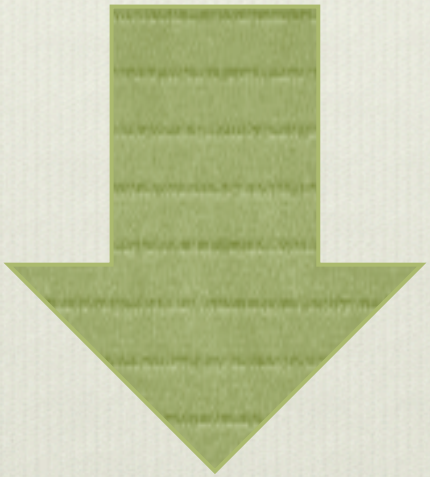
- *Nessus / nmap / blind elephant / other scanning tools don't really count because they rely upon a signature developed for a specific vulnerability / finding.*
- Acunetix
- Burp
- ZAP
- sqlmap

Foundational problems with current scanning techniques

- Systemic signature problem
  - Web Scanners == Anti-Virus
- Solution: Evolve unique exploits for web applications
  - Web Application Firewall blocks ‘or 1=1 -- ?

EVOLVE

‘ or 1=1; --



Aso1239^;’or 2=1 or 1=3 or 1=1 --asdl1ojcud//

Covered in this talk

- Genetic algorithms to create exploits
  - SQL injection (MySQL, SQL, MSSQL, Oracle)
  - Command injection (Bash, CMD, PHP, Python)
  - Attack surface is HTTP / HTTPS POST and GET parameters

- What we will not cover
  - Everything else

Genetic Exploit Development

- Forced Evolution
  - github.com/soen-vanned/forced-evolution

Evolutionary Algorithms

1. Create a large number of exploit strings
2. While solution/goal != found:
    1. Score all of the strings' performance using a fitness function
    2. Cull the weak performing
    3. Breed the strong performing
    4. Mutate the strings randomly
3. Display the exploit string that solved the solution

Forced Evolution

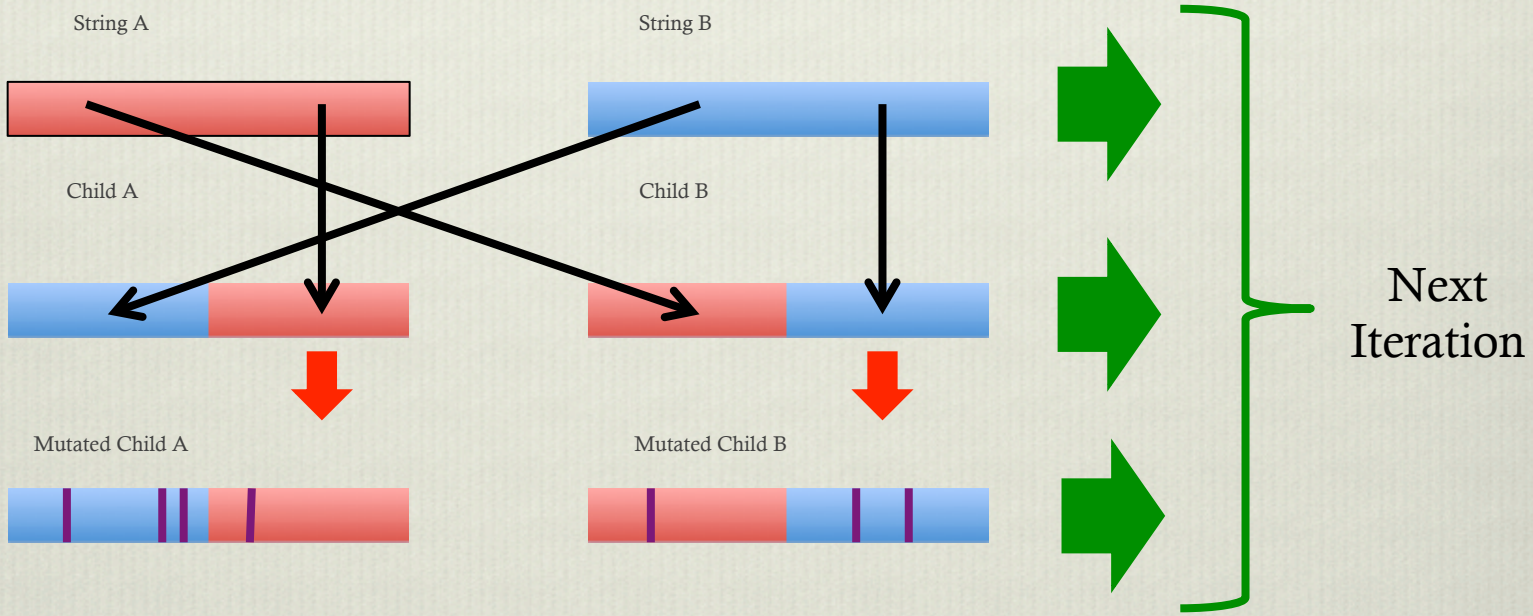
1. Create a large number of pseudo-random strings
2. While exploit != successful:
    1. Send the string as parameter value (I.E. POST, GET, etc.)
    2. Use the response from the server to determine the score (string fitness)
    3. Cull the weak performing strings
    4. Breed the strong performing strings
    5. Mutate the strong performing strings
3. Display the string that successfully exploits the app

Fitness Function

- Does the exploit string cause sensitive information to be displayed?
- Does the string cause an error (and if so, what type?)
- Is the string reflected? (XSS…)
- Other information displayed?

Breeding Strings

- Pairs of strings are bred using genome cross-over



- The amount of children and parents varies on implementation.
  - The amount of children depends on implementation
  - Parents are kept alive depending on implementation

Mutating Strings

- Mutation rate is variable
- Mutation Operations:
  - Mutate
  - Add
  - Remove a string item
- Pre-mutation String: ABCD
  - Post-mutated String: XACF

Population Dynamics

- Mutation rate vs. Search speed
- String cull rate vs. repopulation speed

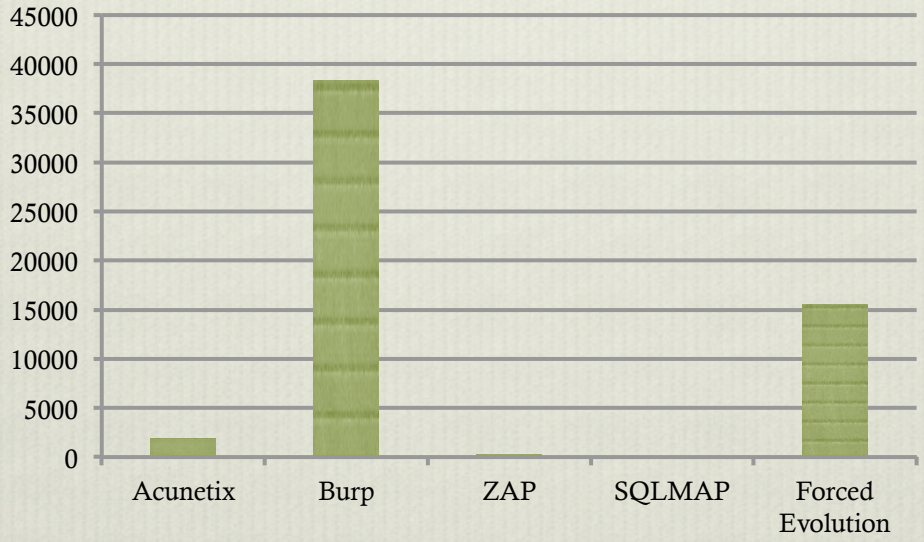
Tool Comparison

- Command Injection
  - Statistics

CMD injection	Vulnerability Found?	Exploit Developed	Auto WAF bypass	Time for Attack (seconds)	Requests
Acunetix	Yes	No	No	20	1854
Burp	Yes	No	Yes	926	38297
ZAP	Yes	No	No	118	264
SQLMAP	N/A	N/A	N/A		N/A
Forced Evolution	Yes	Yes	Yes	246	15489

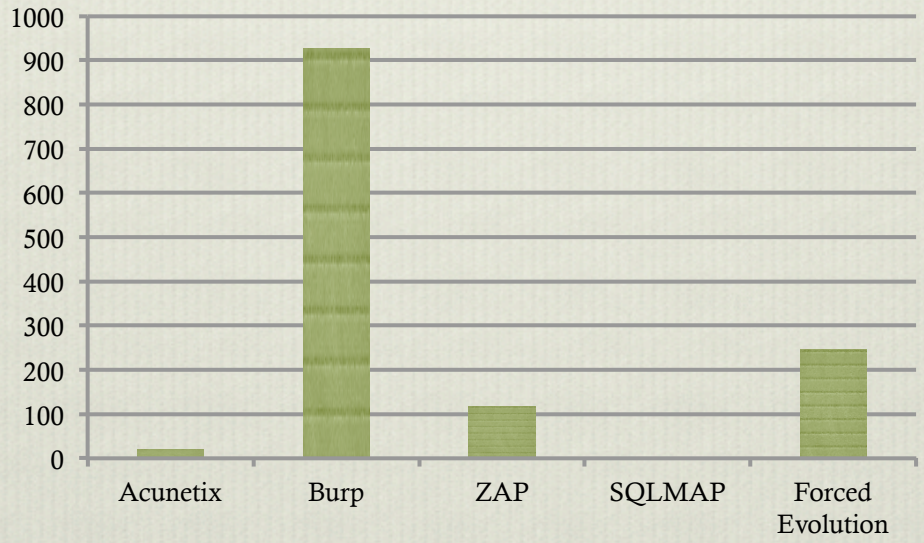
Tool Comparison

- Command Injection
  - Requests sent to server:



Tool Comparison

- Command Injection
  - Time to exploit (seconds)



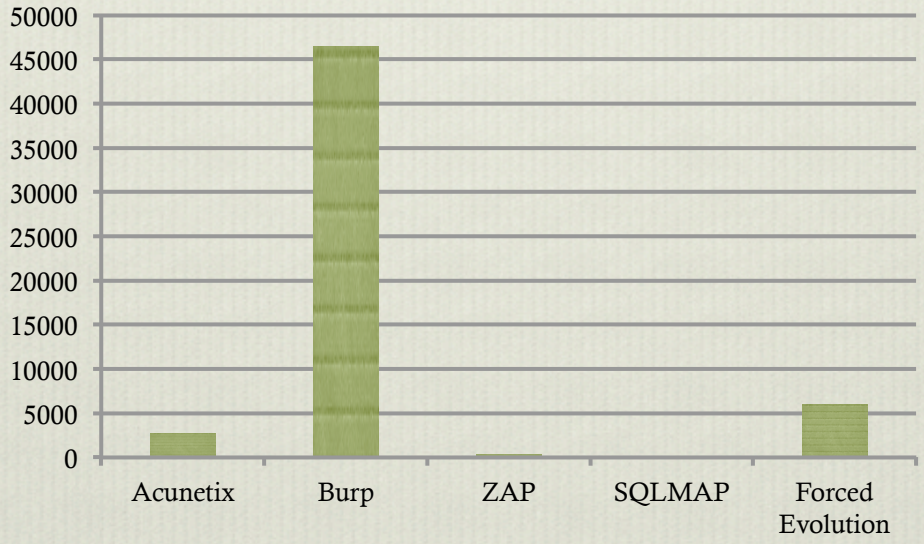
Tool Comparison

- SQL Injection
  - Statistics

SQLi	Vulnerability Found?	Exploit Developed	Auto WAF bypass	Time for Attack	Requests
Acunetix	Yes	Yes	No	53	2685
Burp	Yes	Yes	Yes	1101	46516
ZAP	Yes	No	No	157	315
SQLMAP	Yes	Yes	Yes	15	166
Forced Evolution	Yes	Yes	Yes	17	5996

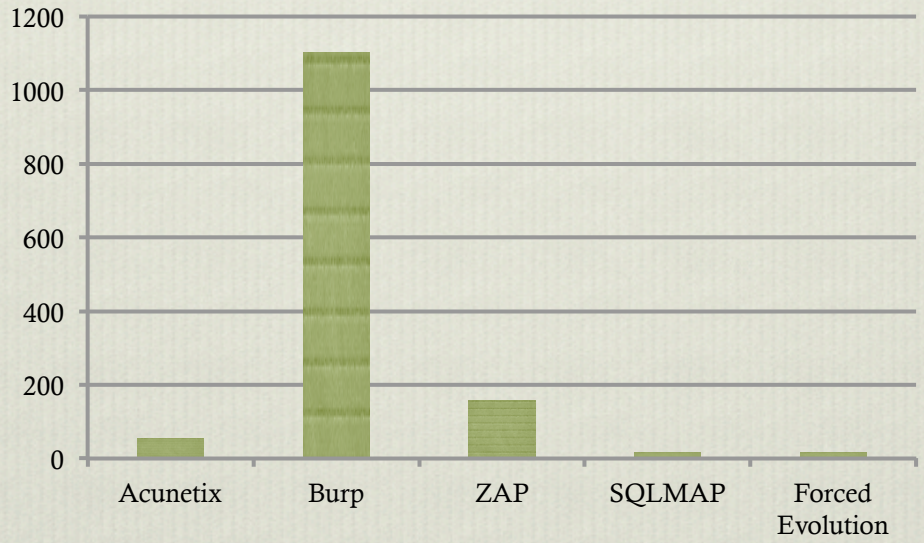
Tool Comparison

- SQL Injection
  - Requests sent to server



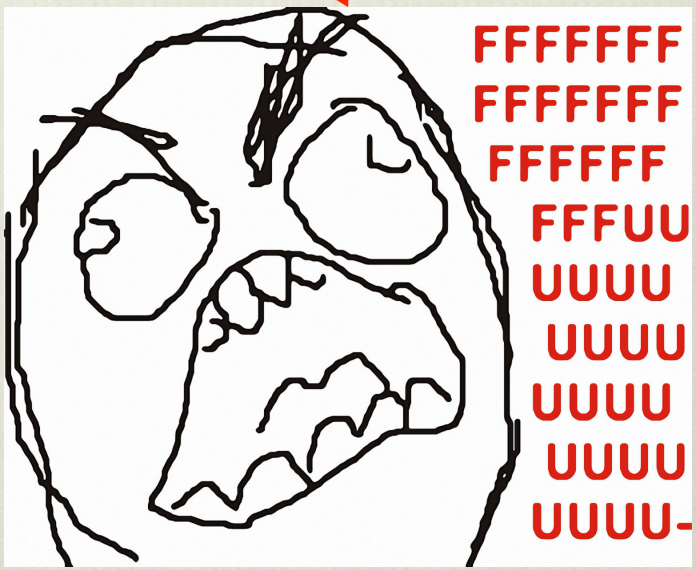
Tool Comparison

- SQL Injection
  - Time to exploit (seconds)



Demo


EVERYTHING
WENT
BETTER
THAN
EXPECTED


FFFFFFF
FFFFFFF
FFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU
UUUU-

Pro's and Con's

- Con's for genetic exploit evolution:
  - Very noisy attacks
  - Small potential to inadvertently destroy the database / OS
  - Slow process to develop and test exploits
  - Sub-optimal to source code analysis

Pro's and Con's

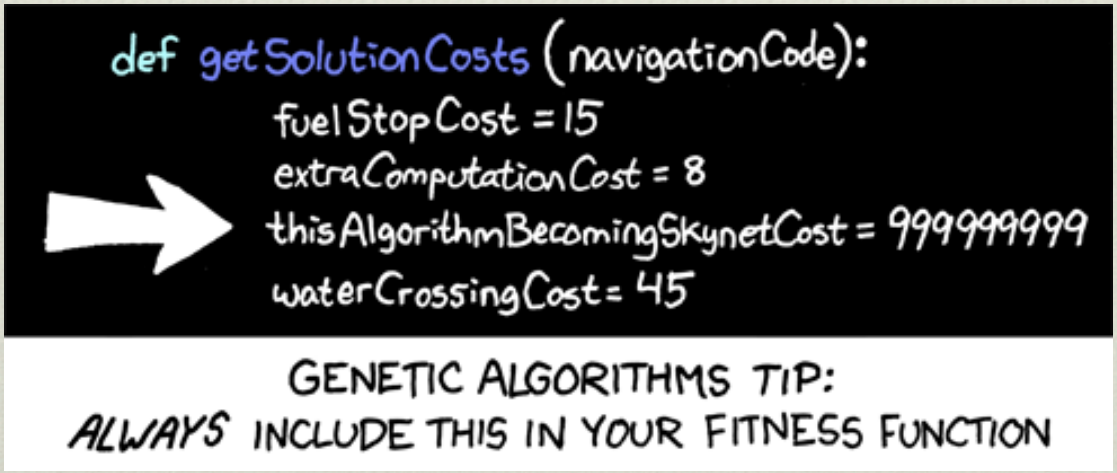
- Pro's for genetic exploit evolution
  - Cheap in CPU/RAM/HD and human time
  - More complete code coverage than other black-box approaches
  - Exploit breeding is the future, upgrades to the current approach will improve efficiency but the code *right now* will break web apps *in the future*.
  - **Automatic exploit development** – Exploits genetically bred to tailor to a specific web app
  - **Emergent exploit discovery** – New exploit methodologies and techniques will emerge

Conclusion

- Download Forced Evolution

  - github.com/soen-vanned/forced-evolution

- Contact: soen.vanned@gmail.com /@soen_vanned / http://0xSOEN.blogspot.com



References

- Fred Cohen (Computer Viruses – Theory and Experiments - 1984)
- Dr. Mark Ludwig (The little & giant black book of computer viruses, Computer Viruses, Artificial Life and Evolution)
- Herm1t's VX Heaven(http://vxheaven.org/)
- Artificial Intelligence: A Modern Approach (3rd Edition, Stuart Russell & Peter Norvig)