Evolving Exploits through Genetic Algorithms

By soen

Who am I

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

Domain Constraints

- What we will cover
  - 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

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 unless there is a signature developed for a specific vulnerability / finding.*
- Acunetix
- Burp
- ZAP
- sqlmap

Foundational problems with current scanning techniques

- Systemic signature problem
  - Anti-Virus == Web Scanners
- Solution: Evolve unique exploits for web applications
  - Web Application Firewall blocks ‘or 1=1 -- ?
    - Evolve from
      - ‘or 1=1 --
    - To:
      - Aso1239^;’or 2=1 or 1=3 or 1=1 --asdl1ojcud//\

Evolutionary Algorithms

In English:

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

Exploit Evolution

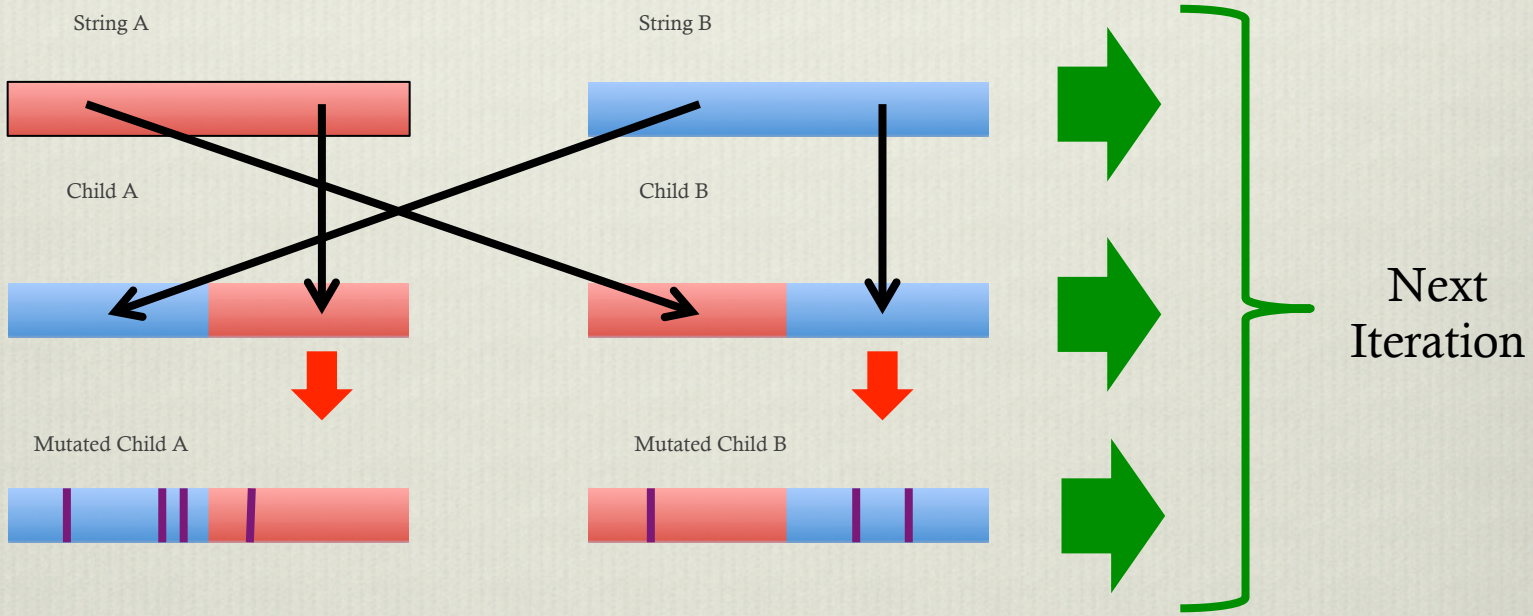
1. Create a large number of strings
2. While exploit != successful:
    1. Send the string as parameter value (I.E. POST, GET, etc.)
    2. Use the response from the server to determine the score
        1. +Error Pages (more if the string was reflected)
        2. +Blank / delayed responses
        3. +For objectives displayed (passwords displayed, sensitive DB information, etc.)
    3. Delete 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

- This is the performance /score of how well a creature performs
- Creatures that score well will live to breed
- Creatures that score poorly will be culled
- Fitness in this context is the following:
  - Does the creature cause sensitive information to be displayed?
  - Does the creature cause an error (and if so, what type?)
  - Is the creature reflected? (XSS…)
  - Is 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

- Pseudo code:
  - Mutation rate is greater than 0 and less than 1.0
  - Select an amount of string items to mutate given the length of the string (0 -> len(string)) * mutation rate
  - For each mutation, replace/add/remove a random string item with a random character
  - Example:
    - Pre-mutation String: ABCD
    - Post-mutated String: XACF
      - (Prepended X, B deleted, and D mutated to F)

Population Dynamics

- It is critical to choose a mutation rate that will allow for sufficient diversity in the pool of creatures, but at the same time allow a solution to be efficiently reached.
- Cull rate / string death rate must be high enough to maintain the population, but low enough to not drastically reduce it. (E.G. For 300% growth rate of breeding the top 33%, cull 67% of the population)

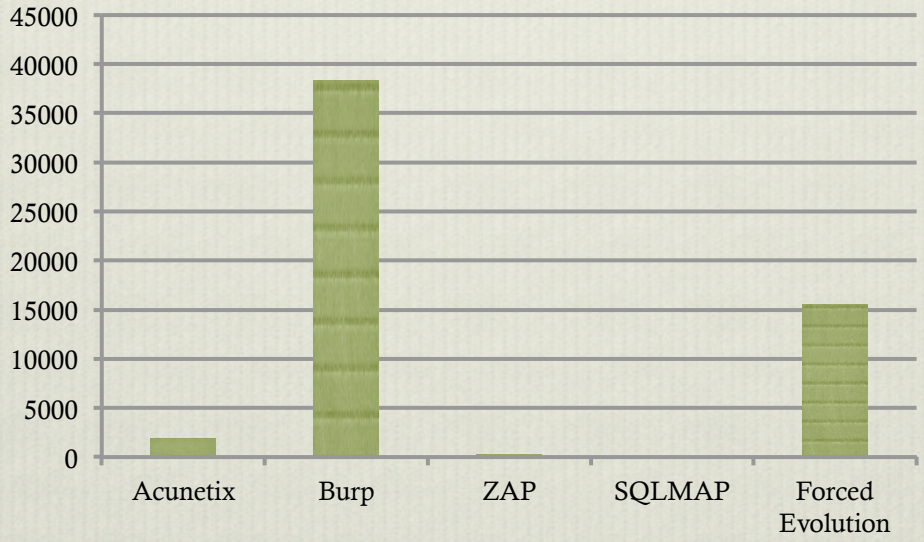
Tool Comparison

- 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

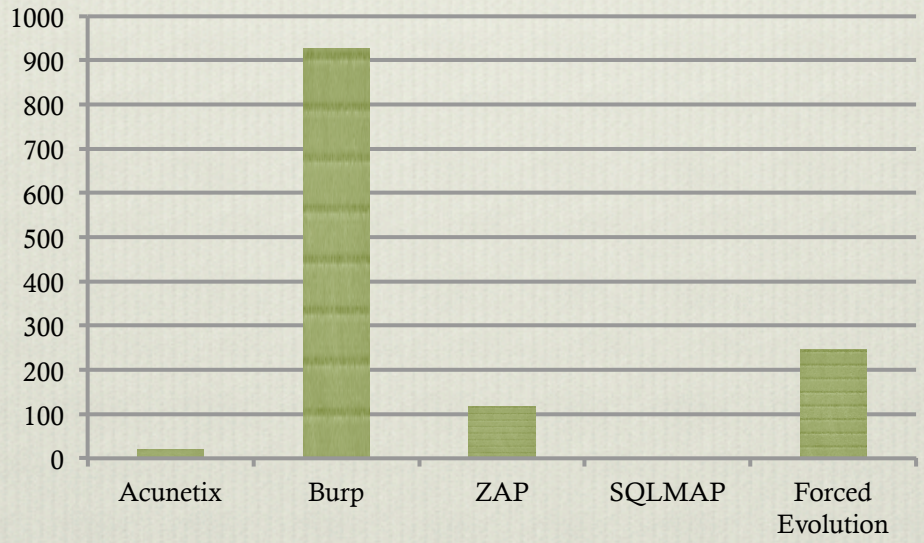
Tool Comparison

- Command Injection
  - Requests sent to server:



Tool Comparison

- Command Injection
  - Time to exploit (seconds)



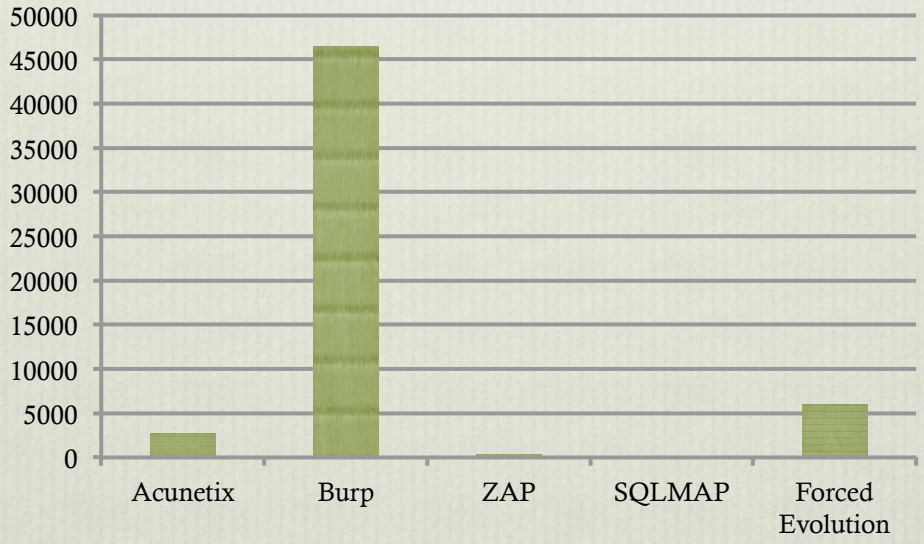
Tool Comparison

- 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

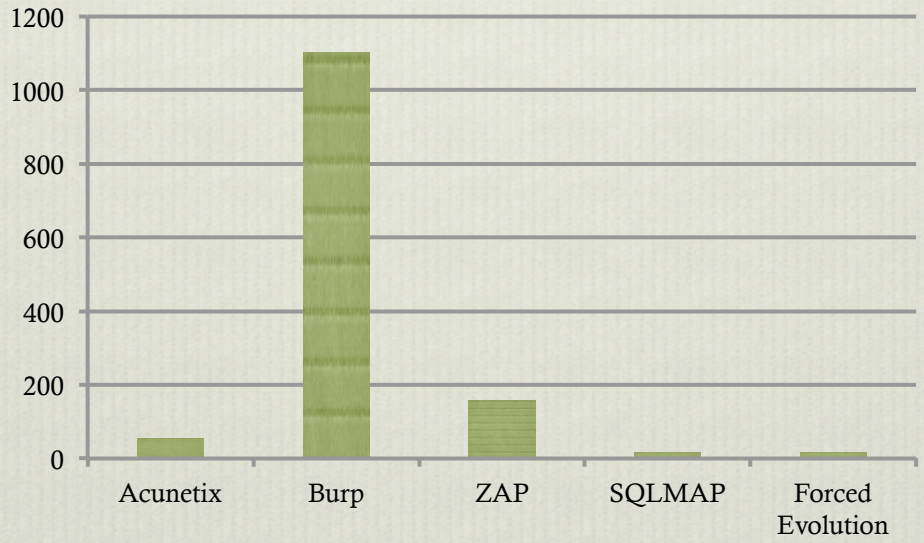
Tool Comparison

- SQL Injection
  - Requests sent to server



Tool Comparison

- SQL Injection
  - Time to exploit (seconds)



Pro's and Con's

- Con's for Exploit Evolution
  - Very noisy attacks
  - 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 Exploit Evolution
  - Cheap in CPU 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 from a system like this.

Demo

Contact

- Download Forced Evolution
  - github.com/soen-vanned/forced-evolution
- soen.vanned@gmail.com
- @soen_vanned
- http://0xSOEN.blogspot.com
- 1KVh6pWfi4tiBPxy9jQCxtcMYnpraWkzmv

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)