

---

# Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing

---

Jared DeMott

Dr. Richard Enbody @msu.edu

Dr. William Punch

DEFCON 2007

---

# Agenda

- Goals and previous works
  - (1) Background
    - Software, fuzzing, and evolutionary testing
  - (2) Describe EFS in detail
    - GPF & PaiMei & development++ == EFS
  - (3) Initial benchmarking results
  - (4) Initial results on a real world application
  - Conclusion and future works
-

---

# Goals and Previous Works

- Research is focused on building a better fuzzer
    - EFS is a new breed of fuzzer
      - No definitive proof (yet) that it's better than current approaches
        - Need to compare to Full RFC type, GPF, Autodafe, Sulley, etc
      - As of 6/21/07 there are no (available) other fuzzers that learn the protocol via a grey-box evolutionary approach
        - Embleton, Sparks, and Cunningham's *Sidewinder* research
          - Code has not been released
        - Hوجلund claims to have recreated something like Sidewinder, but also didn't release details
        - Autodafe and Sulley are grey-box but require a capture (like GPF), or definition file (like Spike), respectively, and do not evolve
-

---

# Section 1: Background

- Software Testing
  - Fuzz Testing
    - Read Sutton/Greene/Amini
    - And then read DeMott/Takanen
  - Evolutionary Testing
-

---

# Software Testing

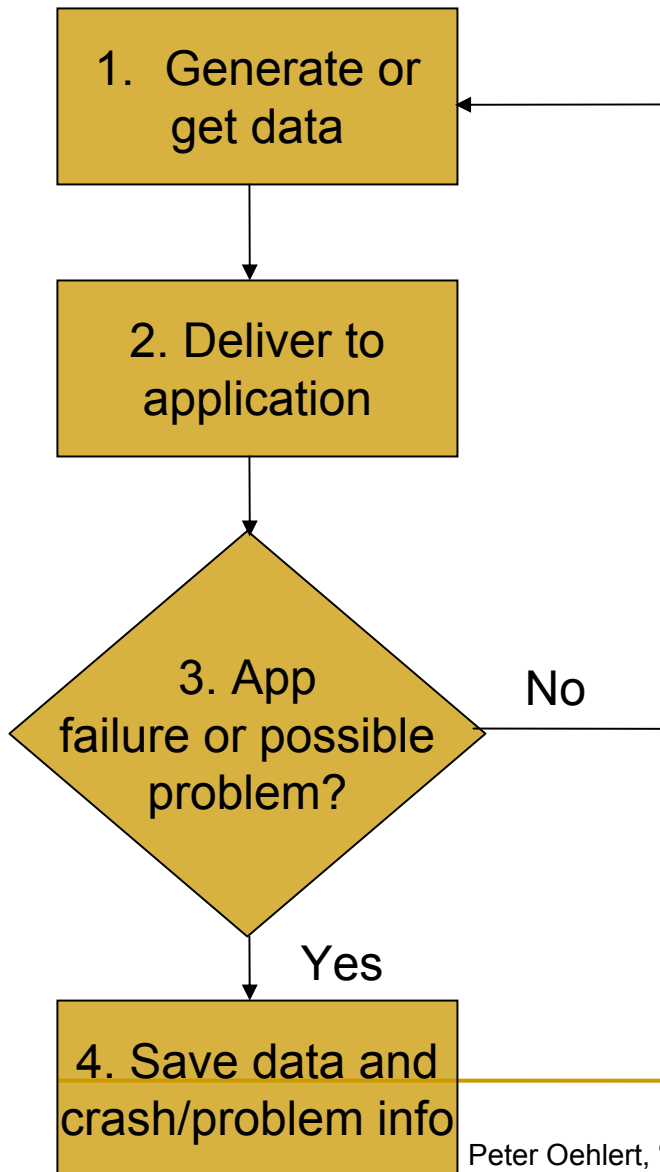
- Software testing can be
    - Difficult, tedious, and labor intensive
      - Cannot “prove” anything other than existence of bugs
    - Poorly integrated into the development process
    - Abused and/or misunderstood
    - Has a stigma as being, “easier” than engineering
  - Software testing is expensive and time-consuming
    - About 50% of initial development costs
  - However, primary method for gaining confidence in the correctness of software (pre-release)
    - Done right, does increase usability, reliability, and security
      - Example, Microsoft’s new security push: SDL
  - In Short, testing is a (NP) hard problem
    - New methods to better test software are important and in constant research
-

---

# Fuzzing, Testing, QC, and QA

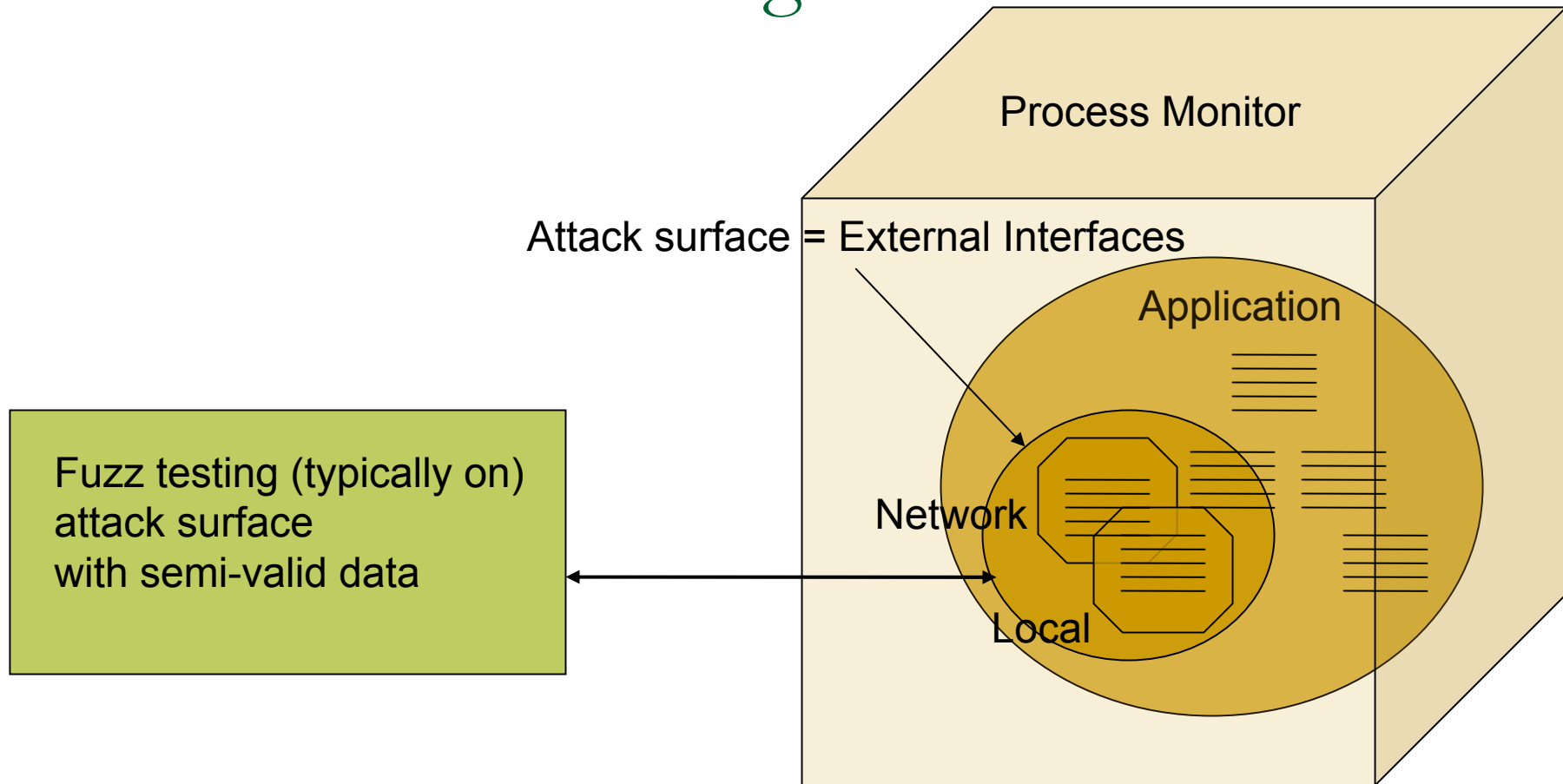
- How does fuzzing fit into the development life cycle?
    - Formal Methods of Development
    - Quality Assurance
      - Quality Control
        - Testing
          - Fuzzing
          - Many other types of testing!
  - Fuzzing is one small piece of the bigger puzzle, but one that has been shown useful to ensure better security
-

# Fuzzing



- Fuzzing is simply another term for interface robustness testing
  - Focuses on:
    - Input validation errors
    - Actual applications - dynamic testing of the finished product
    - Interfaces that have security implications
      - Known as an attack surface
        - Portion of code that is externally exercisable in the finished product
        - Changes of privilege may occur

# Attack Surface Testing



---

# Evolutionary Testing

- Uses evolutionary algorithms (GAs) to discover better test data
    - A GA is a computer science search technique inspired by evolutionary biology
      - Evaluating a granular fitness function is the key
    - ET requires structural (white-box) information (source code)
      - Couldn't find others doing grey-box ET
  - Brief look at ET:
    - Standard approach, typical uses, problems
-

# Current ET Method for Deriving Fitness

- Approach\_level + norm(branch distance)
  - Example: a=10, b=20, c=30, d=40
    - Answer: fitness = 2 + norm(10). (Zero == we've found test data.)

```
(s) void example(int a, int b, int c, int d)
{
(1)     if (a >= b)
        {
(2)         if (b <= c)
            {
(3)                 if (c == d)
                    {
                        //target
                    }
            }
        }
    }
```

---

# Typical ET uses

- Structural software testing
    - Instrument discovered test cases for initial and regression testing
  - Wegener et al. of DaimlerChrysler [2001] are working on ET for safety critical systems
  - Boden and Martino [1996] concentrate on error treatment routines of operating system calls
  - Schultz et al. [1993] test error tolerance mechanisms of an autonomous vehicle
-

# ET Problems

- Flag problem == flat landscape. Resort to random search

```
void flag_example(int a, int b)
{
    int flag = 0;
    if (a == 0)
        flag = 1;
    if (b != 0)
        flag = 0;
    if (flag)
        //target
}
```

- Deceptive problems

```
double function_under_test
(double x)
{
    if (inverse(x) == 0)
        //target
}
double inverse (double d)
{
    if (d == 0)
        return 0;
    else
        return 1 / d;
}
```

---

# Evolutionary Fuzzing System

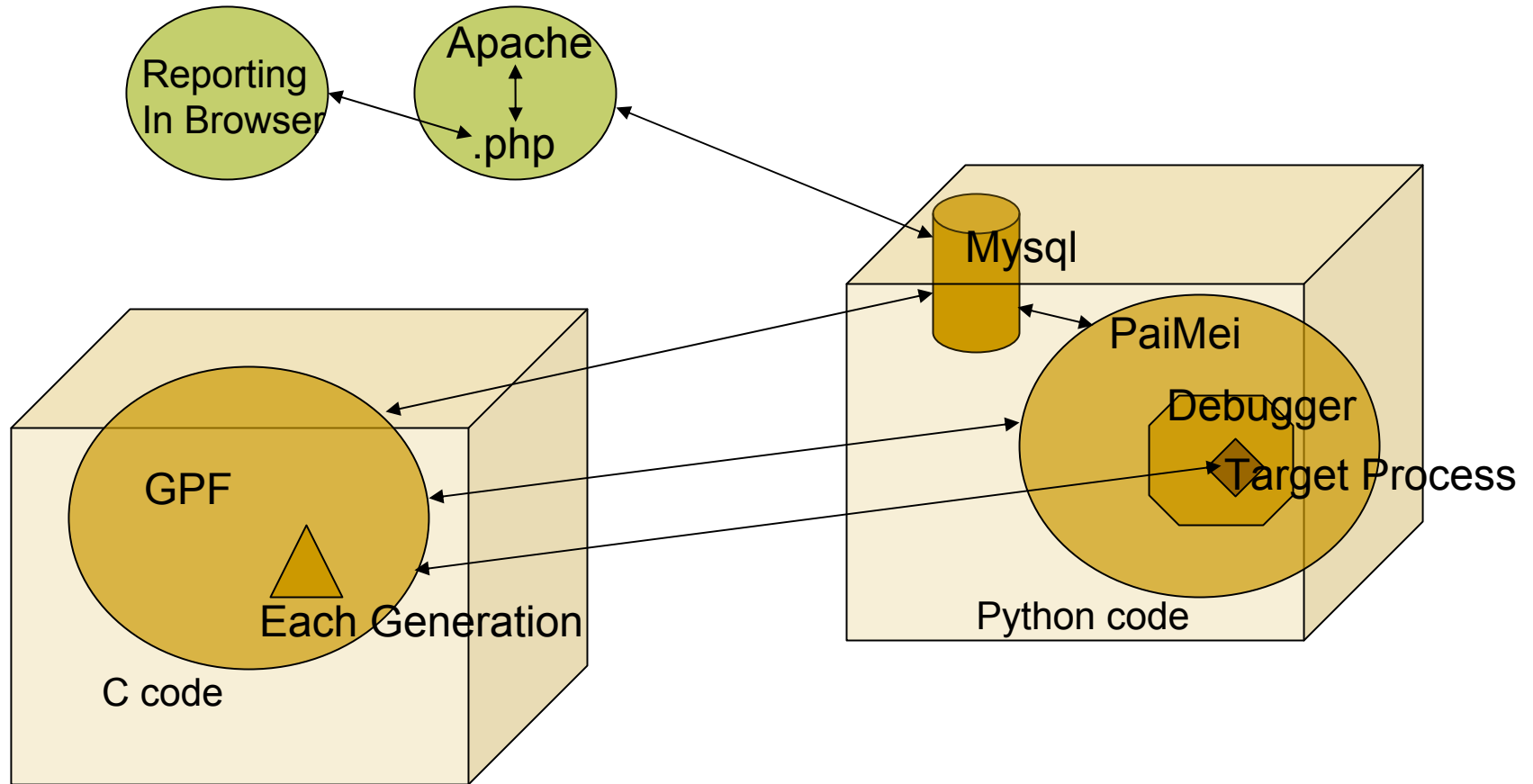
- McMinn and Holcombe (U.o.Sheffield) are working on solving ET problems [2]
    - 2006 paper on Extended Chaining Approach
  - Our approach is different for two reasons:
    - Grey-box, so no source code needed
    - Application is being monitored while test cases are being discovered. Fuzzing heuristics are used in mutations. This equals real-time testing. Crash files are written while evolution continues. Also includes reporting capability. Seed file helps with some of the traditional ET problems, though still rough fitness landscape.
-

---

# Section 2: A Novel Approach

- Evolutionary Fuzzing System
    - Evolutionary Testing
      - EFS uses GA's, but does not require source code
    - Fuzzing
      - EFS uses GPF for fuzzing
    - PaiMei
      - EFS uses a modified version of pstalker for code coverage
-

# EFS: A System View



---

# EFS: GPF - Stalker (PaiMei) Protocol

- GPF initialization/setup data → PaiMei
  - Ready ← PaiMei
  - <GPF carries out communication session with target>
  - GPF {OK|ERR} → PaiMei
  - <PaiMei stores all of the hit and crash information to the database>
-

---

# EFS: How the Evolution works

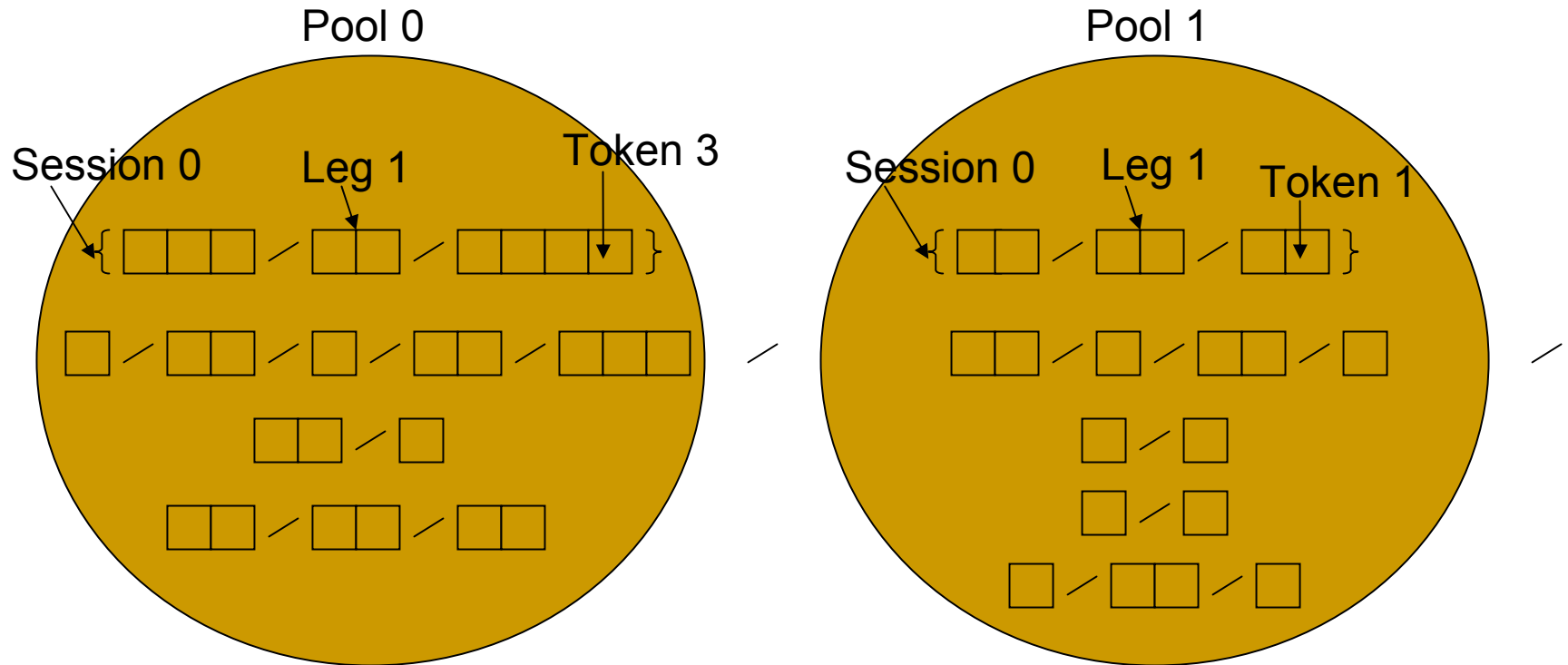
- GA or GP?
    - Variable length GA. Not working to find code snippets as in GP. We're working with data (GA).
  - Code coverage + diversity = fitness function
    - The niching or speciation used for diversity is defined later
    - Corollary 1:
      - Code coverage != security, but < 100% attack surface coverage == even less security
    - Corollary 2:
      - 100% attack surface coverage + diverse test cases that follow and break the protocol with attack/fuzzing heuristics throughout == the best I know how to do
-

---

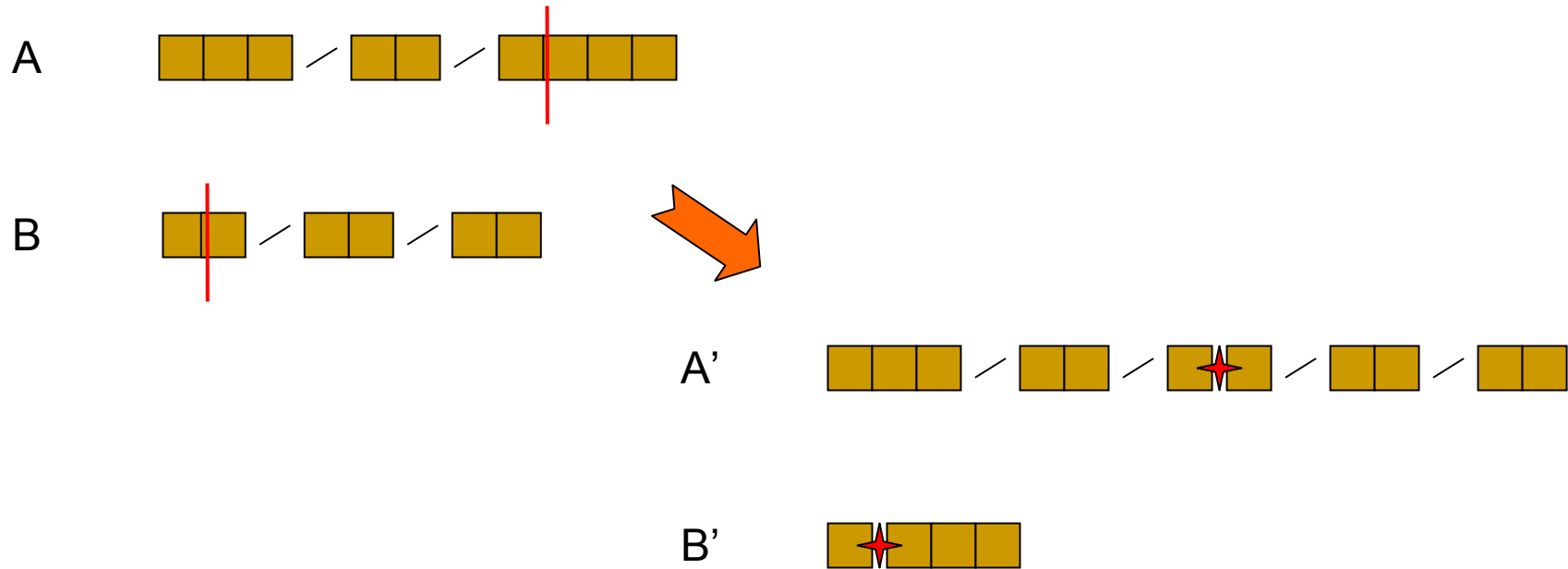
# EFS: How the Evolution works (cont.)

- Any portion of the data structures can be reorganized or modified in various ways
    - But not the best pool or the best session/pool
      - Elitism of 1
  - All evolutionary code is 100% custom code
    - Session Crossover
    - Session Mutation
    - Pool Crossover
    - Pool Mutation
-

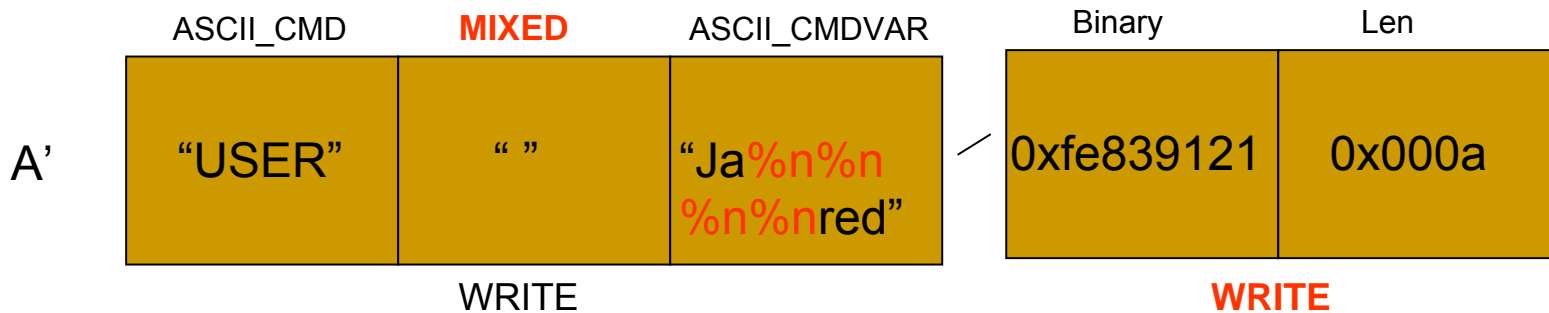
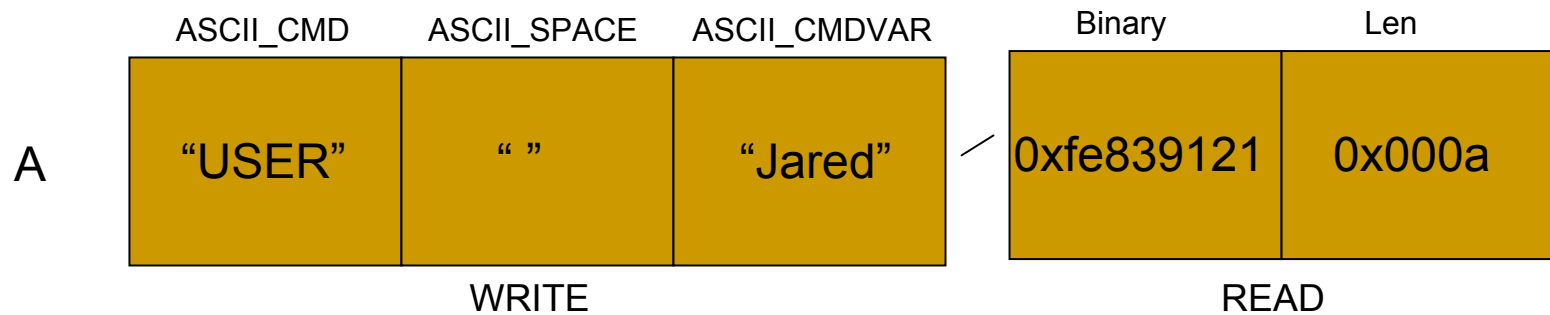
# EFS: Data Structures



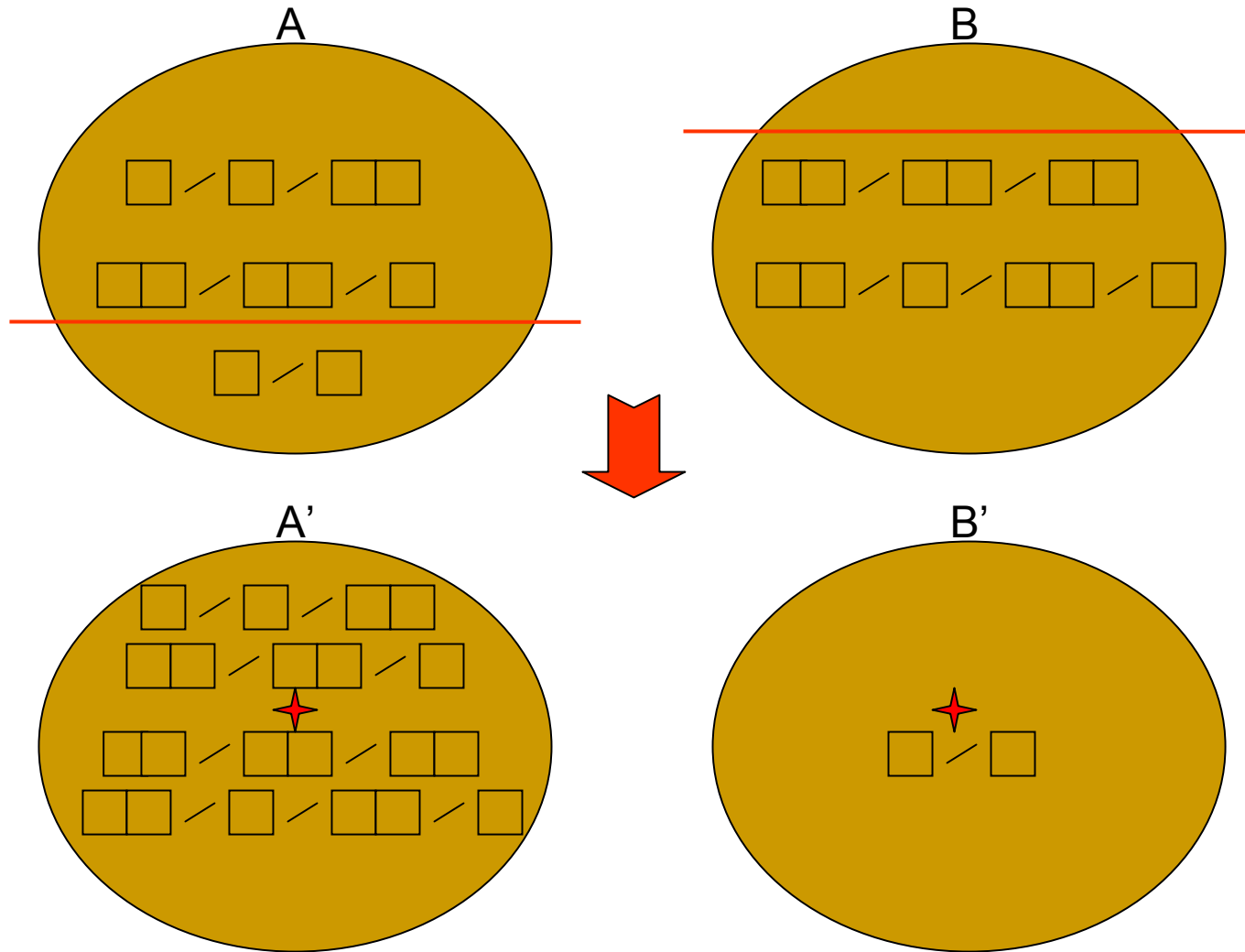
# EFS: Session Crossover



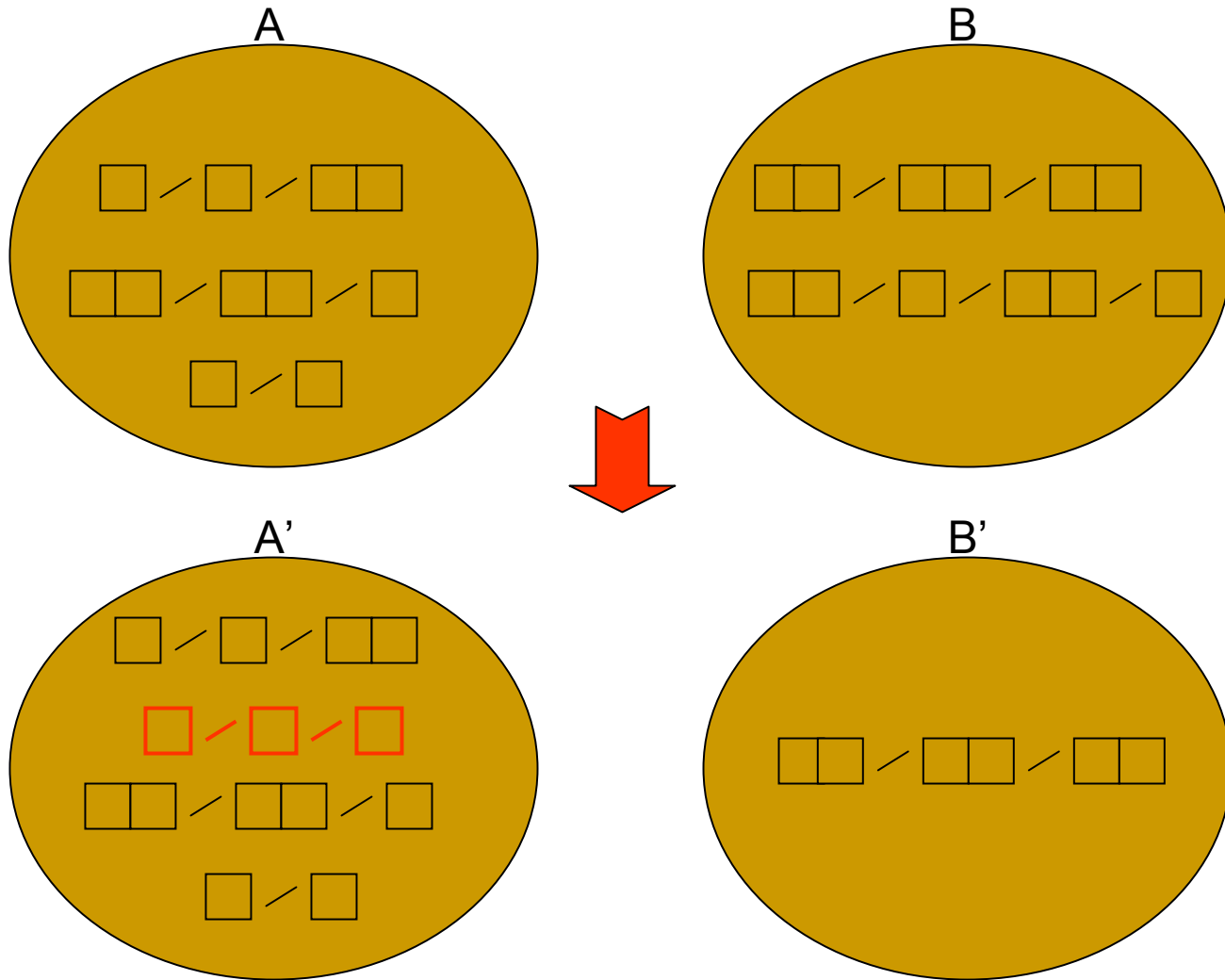
# EFS: Session Mutation



# EFS: Pool Crossover



# EFS: Pool Mutation



---

# Simple Example of Maturing EFS Data

- GENERATION 1

- S1: "USER #\$\$%^&\*Aflkdsjflk"

- S2: "ksdfjkj\nPASS %n%n%n%n"

- S3: "\r\njksd Jared9338498\d\d\xfeffe"

- ...

- GENERATION 15

- S1: "USER #\$\$%\n PASS %n%n%n%n\r\njksd"

- S2: "PASS\nQUIT NNNNNNNNNNNN\r\n"

- S3: "RETR\r\nUSER ;asidf;asifh; kldsif;kdfj"

- ...

---

---

# EFS: GPF -E Parameters

- Mysql Host, mysql user, mysql passwd
  - ID, generation
  - PaiMei host, PaiMei port, stalk type
  - Playmode, host, port, sport, proto, delay, wait
  - Display level, print choice
  - Pools, MaxSessions, MaxLegs, MaxToks, MaxGenerations, SessionMutationRate, PoolCrossoverRate, PoolMutationRate
  - UserFunc, SeedFile, Proxy
-

# Seed File

## ■ SMTP

- ❑ HELO
- ❑ Mail from: [me@you.com](mailto:me@you.com)
- ❑ Rcpt to: root
- ❑ Data
- ❑ “Hello there”
- ❑ \r\n.\r\n
- ❑ EHLO
- ❑ RSET
- ❑ QUIT
- ❑ HELP
- ❑ AUTH
- ❑ BDAT
- ❑ VRFY
- ❑ EXPN
- ❑ NOOP
- ❑ STARTTLS
- ❑ etc.

## ■ FTP

- ❑ USER anonymous
- ❑ PASS [me@you.com](mailto:me@you.com)
- ❑ CMD
- ❑ PASV
- ❑ RETR
- ❑ STOR
- ❑ PORT
- ❑ APPE
- ❑ FEAT
- ❑ OPTS
- ❑ PWD
- ❑ LIST
- ❑ NLST
- ❑ TYPE
- ❑ SYST
- ❑ DELE
- ❑ etc.

---

# EFS: Stalker Start-up Sequence

- Create and PIDA file using IDApro
    - Load the PIDA file in PaiMei
  - Configure/start test target
  - Stalk by functions or basic blocks
  - Filter common break points
    - Start-up, connect, send junk, disconnect, GUI
      - Allows EFS to run faster
  - Connect to mysql
    - Listen for incoming GPF connection
  - Start GPF in the -E (evolutionary) mode
-

# EFS GUI (the PaiMei portion)

The screenshot displays the EFS GUI interface with several panels:

- Data Sources:** A tree view under 'Available Targets' showing folders like 'filter' and 'GPF', with sub-items such as 'gftp\_start\_gui\_conn\_junk\_discon' and 'TextServer\_startup\_con\_junk\_d'.
- Data Exploration:** A table with columns '#', 'EIP', 'TID', 'Module', 'Func?', and 'Tag'. Below it, statistics show 'Functions: 0 / 309' and 'Basic Blocks: 0 / 671'.
- Data Capture:** Controls for 'Refresh Process List', 'Target' (set to 'c:\my dir\a.exe' args), 'StartScript', 'Load/Attach' (set to 'c:\textserver.'), 'Coverage Depth' (Basic Blocks selected), 'Database to Save Hits' (GPF selected), and 'Start Stalking' options (None selected).
- Log Console:** A text area at the bottom containing the following log output:

```
[*] EFS (Evolving Fuzzer System), by Jared DeMott
[*] Based on the PaiMei Process Stalker module, by Pedram Amini
[*] Loaded PIDA module 'textserver.exe' in 0.25 seconds.
[*] Function coverage at 0.000000%. Basic block coverage at 0.000000%.
[*] Using 'gpf' as stalking tag.
```

**Listen for Fuzzer Command & Control**

Host:	0.0.0.0
Port:	31338
General Wait:	.3
Dump Directory:	mps\TextServer\0
GPF_ID:	0

Listen

---

# Section 3: Research Evaluation

- Benchmarking EFS
    - Attack surface coverage
    - Text and Binary protocols
    - Functions (funcs) vs. basic blocks (bbs)
    - Pool vs. Diversity (also called niching)
  - See benchmarking paper for more details [3]
    - Will be up on vdalabs.com when complete
-

---

# Benchmarking: An investigation into the properties of EFS

- Develop a tool kit that can be used to test various products
  - Currently the toolkit is simply two network programs used to test EFS's ability to discover a protocol
    - Clear text (TextServer)
    - Binary (BinaryServer)
  - Intend to insert easy and hard to find bugs, to test 0day hunting ability
-

---

# TextServer

- Three settings, low (1 path), med (9 paths), high (19 paths)
  - Protocol
    - ← “Welcome.\r\n Your IP is 192.168.31.103”
    - “cmd x\r\n” →
    - ← “Cmd x ready. Proceed.\r\n”
    - “y\r\n” →
    - ← “Sub Cmd y ok.\r\n”
    - “calculate\r\n” →
    - ← “= x + y\r\n”
-

---

# Aside: Measuring the Attack Surface

- One example, TextServer on Medium:
    - Startup and shutdown = 137 BBs or  $137/597 = \underline{23\%}$  of code.
    - Network code = 15 BBs or  $15/597 = \underline{3\%}$  of code
    - **Parsing** = 94 BBs or  $\underline{16\%}$  of code. *This is the portion of code likely to contain bugs!*
    - Total Attack surface = network code + parsing. 109bb or  $\underline{18\%}$  of code.
    - Code accounted for:  $137+94\text{bb}$  or  $\underline{39\%}$ .  
( $68+22\text{funcs}$  or  $31\%$ )
-

---

# The seed file for TextServer

- ❑ “\r\n”
  - ❑ “calculate”
  - ❑ “cmd “
  - ❑ “1”
  - ❑ “2”
  - ❑ “3”
  - ❑ “4”
  - ❑ “5”
  - ❑ “6”
  - ❑ “7”
  - ❑ “8”
  - ❑ “9”
-

---

# Clear Text Results

- EFS had no trouble learning the language of *TextServer.exe*
  - Best session was found quickly
  - But the entire attack surface was not completely covered
    - Why? Think “error” or “corner cases”
    - Used pools to increase session diversity. Had some success, but still not 100%
    - In a few slides we see that niching was used as well, and did better than pools, but still not 100%
-

# BinaryServer

- Will be similar to TextProtocol, but binary format

Client Request Message Structure →:

Total LEN 4 bytes	Session ID 4 bytes	CMD LEN 2 bytes	CMD Str Var bytes
----------------------	-----------------------	--------------------	----------------------

← Server Response Message Structure:

Total LEN 4 bytes	Session ID 4 bytes	RSP LEN 2 bytes	RSP Str Var bytes
----------------------	-----------------------	--------------------	----------------------

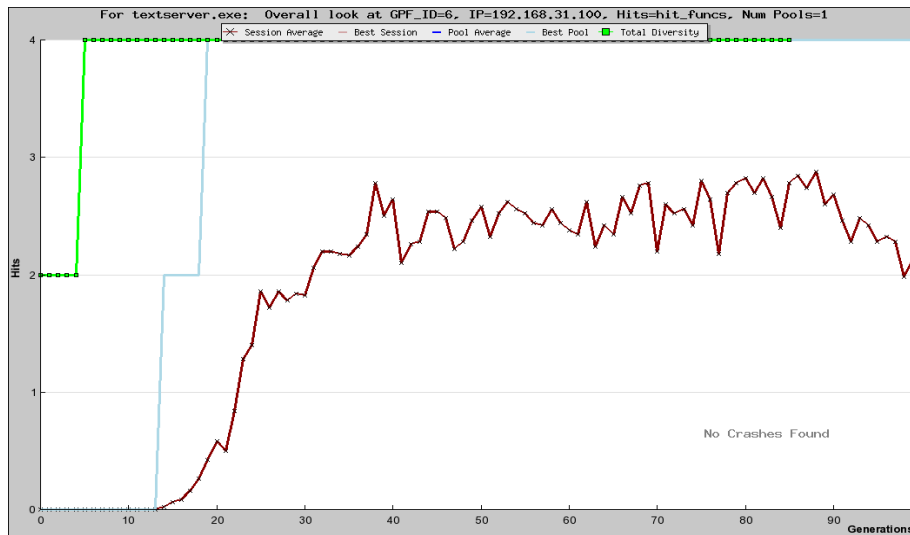
---

# Binary Protocol Results

- Lengths shouldn't be too much trouble as EFS/GPF has a tok type for lengths
    - Initial tests support this
    - Hashes are not yet implemented in GPF
    - Binary protocol not yet implemented/tested
-

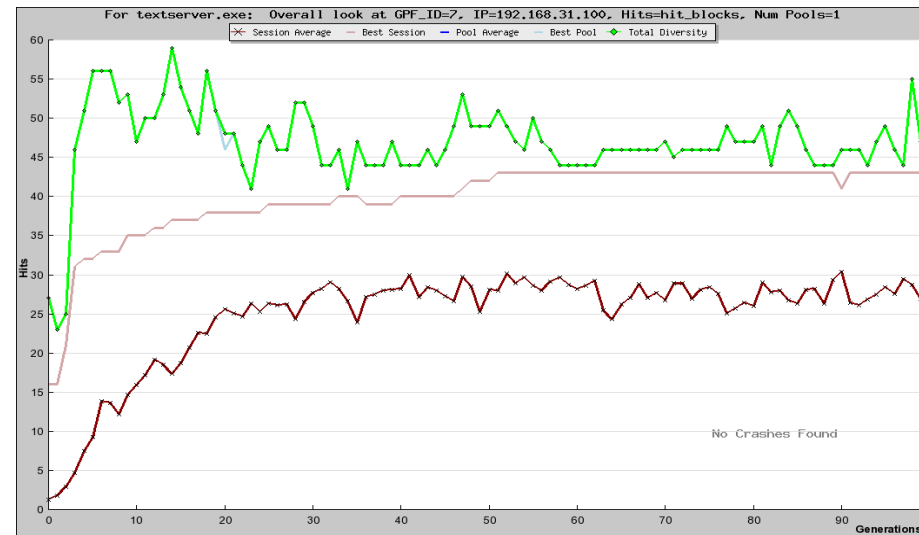
# Functions vs. Basic Blocks

- For applications with few functions, basic blocks should be used
- For more complex protocols, functions suffice and increase run speed



Low, Funcs, 1 Pool:

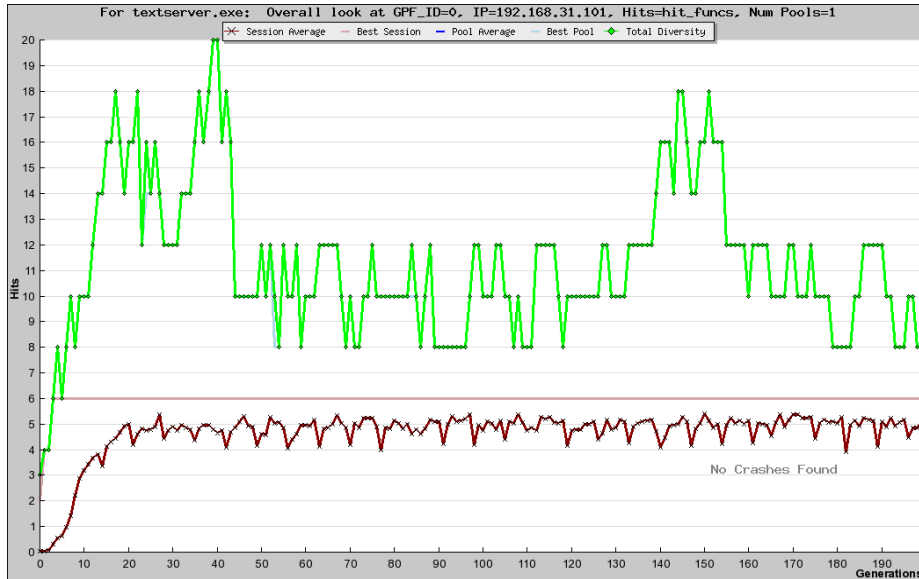
Best Session: 4/6 or 66%



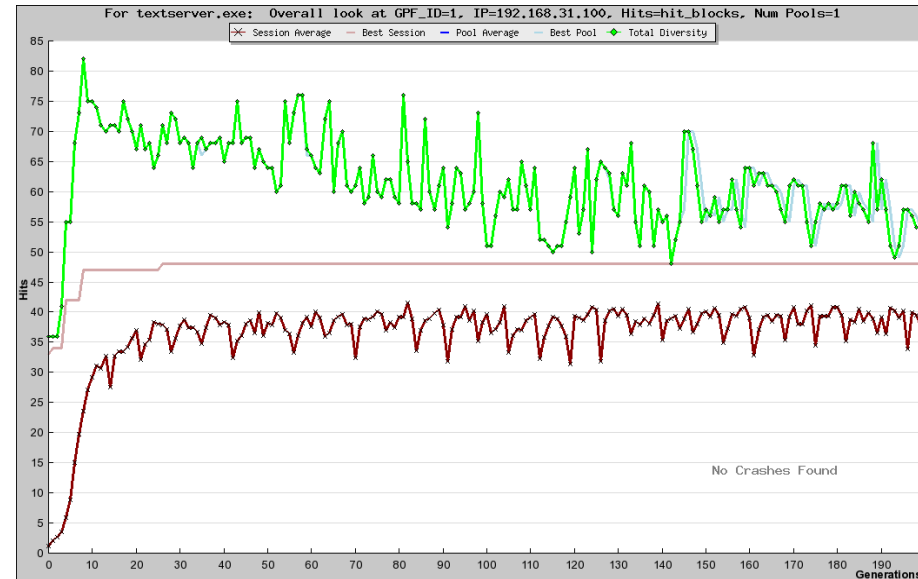
Low, BBs, 1 Pool:

Best Session: 40/37 or 100%+

# Funcs vs. BBs (cont.)



Med, Funcs, 1 Pool:  
Best Session: 6/6 or 100%  
Diversity Peak: 20/22 or 90%



Med, BBs, 1 Pool:  
Best Session: 47/37 or 100%+  
Diversity Peak: 83/94 or 88%

---

# Testing the effects of Pools

- Pools work to achieve better session diversity
    - Also achieved better crash diversity in gftp
  - Didn't achieve 100% coverage of attack surface
  - Case study at the end will show the positive affects of pools
  - Comparing and adding to niching
-

---

# Niching (or Speciation) to Foster Diversity

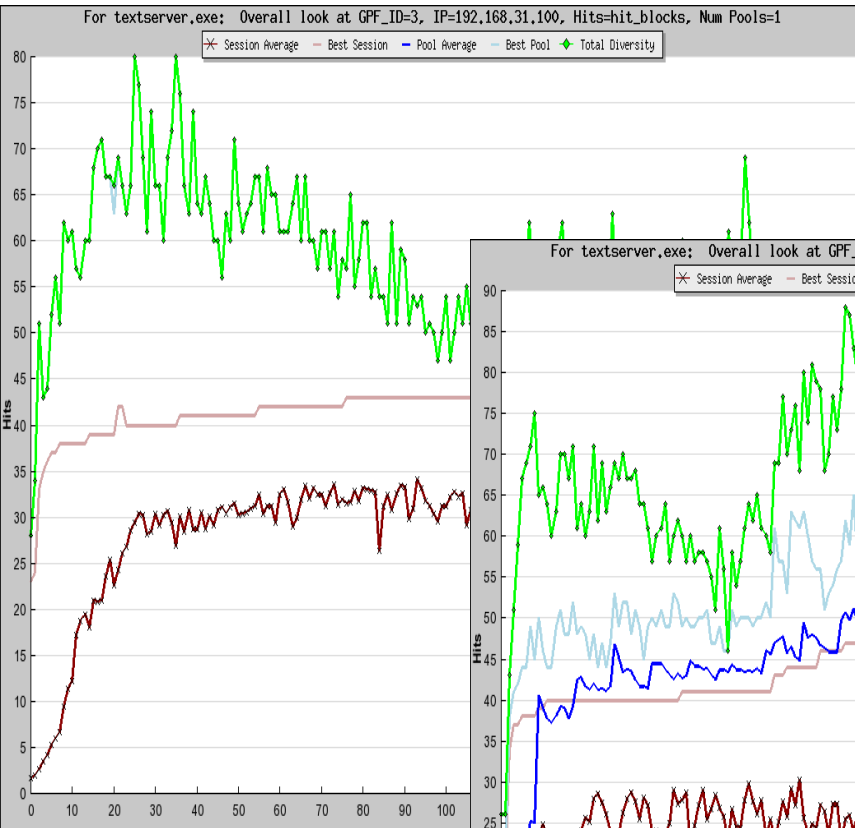
- Recently implemented so grab the new stuff off [vdalabs.com](http://vdalabs.com)
  - Provides a fitness boost for sessions and pools that are diverse when compared to the best
    - ***Fitness = Hits + ( (UNIQUE/BEST) \* (BEST-1) )***
      - ***Hits: code coverage, funcs or bbs***
      - ***UNIQUE: number of hits not found in the best session***
      - ***BEST: Session or Pool with the best CC fitness***
-

---

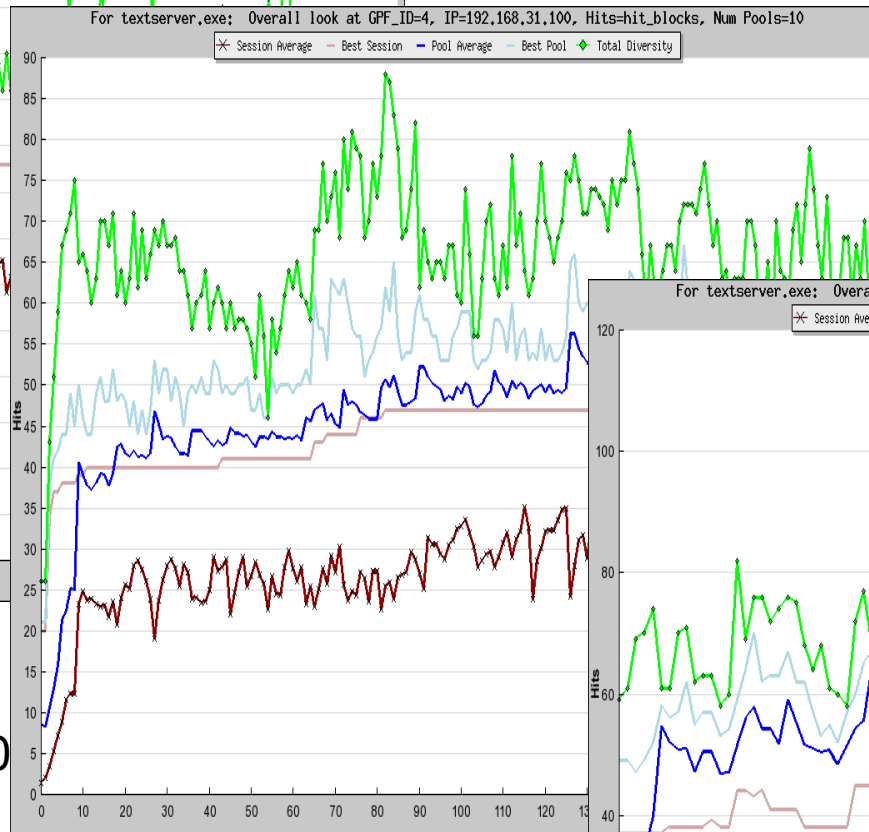
# Diversity in Action

- S1: 10 hits - (a, b, c, d, e, f, g, h, i, j)
  - S2: 7 hits - (a, b, d, e, f, g, h)
  - S3: 5 hits - (v, w, x, y, z)
- 
- Final fitnesses:
    - S1:  $10 + (0/10) * 9 = 10$
    - S2:  $7 + (0/10) * 9 = 7$
    - S3:  $5 + (5/10) * 9 = 9.5$
  - Same for pools
-

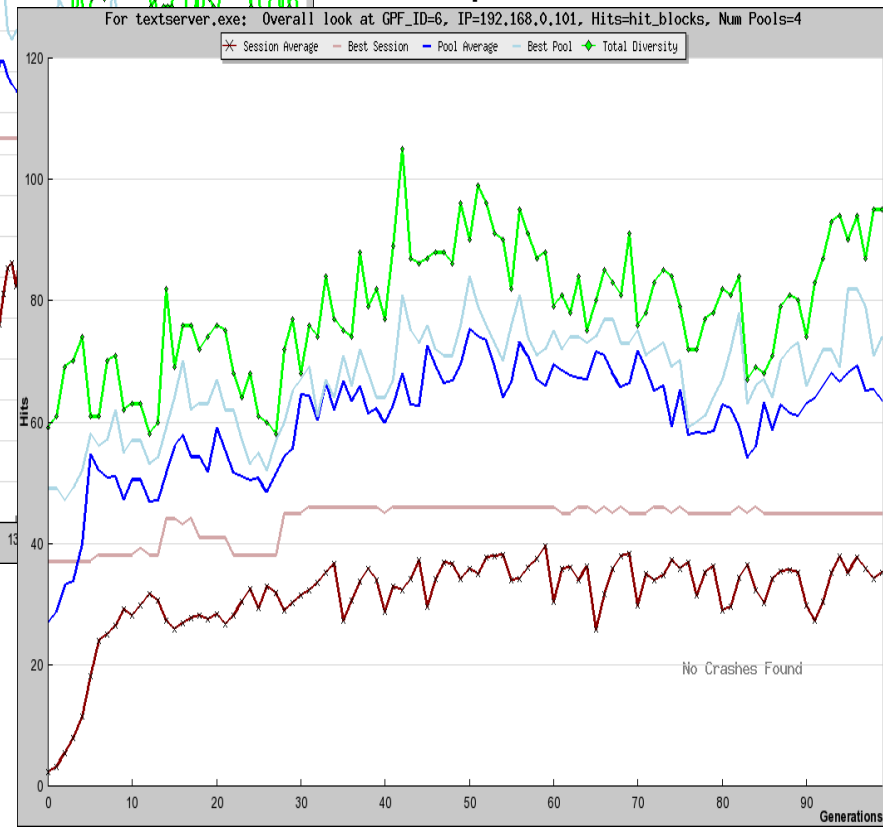
# Pools and Diversity



High, BBs, 1 Pool  
Best Session: 43  
Diversity Peak: 80  
Downward trend



High, BBs, Multi-Pool  
Best Session: 47  
Diversity Peak: 87  
Up and down trend



High, BBs, Multi-Pool  
DIVERSITY ON  
AVG: 46  
Total Peak: 107  
Up and down trend

---

# Section 4: Results

- Initial Results
  - Golden FTP
  - IIS FTP/SMTP



---

# Testing on Real World Code

- Golden FTP
    - Found lots of bugs
  - IIS FTP and SMTP
    - Found no bugs, but did seem to show some instability in FTP
      - Would lock or die once and a while
  - Plan to test many more
    - Haven't tried any with diversity on yet
-

# EFS: Found user & password (outdated picture)

**PAIMEIconsole**

Connections Advanced Help

**Data Sources**

Refresh Target List

- Available Targets
  - filter\_hits
  - startup\_gui\_connect\_junk\_disco
  - GPF 30\_1

**Data Exploration**

#	EIP	TID	Module	Func?	Tag
Functions: 0 / 5145					
Basic Blocks: 0 / 11716					

Dereferenced Data

**Data Capture**

Refresh Process List

PID	Process
-----	---------

Loading

Use/Load:

Close After Stalk  Use if Running

Coverage Depth

Functions  Basic Blocks

Restore BPs  Heavy  Unhandled Or

**PAIME Modules**

# Func	# BB	PIDA ...
5145	11716	gftp.exe

```
[*] Setting 1229 breakpoints on functions in main module
[*] Loading 0x7c900000 \WINDOWS\system32\ntdll.d11
[*] Loading 0x7c800000 \WINDOWS\system32\kernel32.d11
[*] Loading 0x77dd0000 \WINDOWS\system32\advapi32.d11
[*] Loading 0x77e70000 \WINDOWS\system32\rpcrt4.d11
[*] Loading 0x77c00000 \WINDOWS\system32\version.d11
[*] Loading 0x71ad0000 \WINDOWS\system32\wsock32.d11
[*] Loading 0x71ab0000 \WINDOWS\system32\ws2_32.d11
[*] Loading 0x77c10000 \WINDOWS\system32\msvcrt.d11
[*] Loading 0x71aa0000 \WINDOWS\system32\ws2hel.p.d11
[*] Loading 0x773d0000 \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.d11
[*] Loading 0x77f10000 \WINDOWS\system32\gd132.d11
[*] Loading 0x77d40000 \WINDOWS\system32\user32.d11
[*] Loading 0x77f60000 \WINDOWS\system32\shlwapi.d11
[*] Loading 0x7c9c0000 \WINDOWS\system32\shell32.d11
[*] Loading 0x774e0000 \WINDOWS\system32\ole32.d11
[*] Loading 0x77120000 \WINDOWS\system32\oleaut32.d11
[*] Loading 0x76390000 \WINDOWS\system32\imm32.d11
[*] Loading 0x5ad70000 \WINDOWS\system32\uxtheme.d11
[*] Loading 0x74720000 \WINDOWS\system32\MSCTF.d11
[*] Loading 0x10000000 \Program Files\McAfee\SpamKiller\MSK0EP1g.d11
[*] Loading 0x755c0000 \WINDOWS\system32\MSCTFIME.IME
[*] Loading 0x71a50000 \WINDOWS\system32\mswsock.d11
[*] Loading 0x662b0000 \WINDOWS\system32\hnetcfg.d11
[*] Loading 0x71a90000 \WINDOWS\system32\wshtcpip.d11
[*] Loading 0x00e90000 \Program Files\McAfee.com\VS0\MCVSSkt.D11
[*] debugger hit 00423f0c cc #1
[*] debugger hit 0041af78 cc #2
[*] debugger hit 00423fb0 cc #3
[*] debugger hit 00418a20 cc #4
[*] debugger hit 00415efc cc #5
[*] debugger hit 0041b3f4 cc #6
[*] debugger hit 00423fc8 cc #7
[*] debugger hit 00423f70 cc #8
[*] debugger hit 00423cc1 cc #9
[*] debugger hit 00417ccc cc #10
[*] debugger hit 004197c0 cc #11
[*] debugger hit 00418668 cc #12
```

Connected to GPF!

**Golden FTP Server**

Incoming connection:  
IP - 192.168.31.103  
Username - anonymous

start C:\jared\school\cse8... PAIMEIconsole Inbox - Thunderbird runningPaimei.PNG - ... 8:22 AM

# EFS: Crash Example (outdated picture)

The screenshot displays the PAIMEIconsole application interface. On the left sidebar, there are logos for PAIMEI docs, PAIMEI explore, PAIMEI filefuzz, and PAIMEI stalker. The main window is divided into several panes:

- Data Sources:** Shows a tree view of available targets, including 'filter\_hits' and 'startup\_gui\_connect\_junk\_disco'.
- Data Exploration:** Contains a table of running processes and a 'Dereferenced Data' section.
- Data Capture:** Includes options for loading, coverage depth, and stopping stalking.
- Log Console:** Displays the following crash message:

```
[*] debugger hit 0040e2dc cc #29
[*] 0x00402a0d rep movsd from thread 2824 caused access violation
when attempting to write to 0x004174c4
```
- CONTEXT DUMP:** Shows register values and memory addresses for the crashed thread.

```
EIP: 00402a0d rep movsd
EAX: 004174c4 ( 4289732) -> N/A
EBX: 007bfe40 ( 16514624) -> TA?B ;B';B1<B<BTB(+P ,B(-B1-Bhc,8B=8B,8B,4.9B,hDB@DB (stack)
ECX: 00000001 ( 1) -> N/A
EDX: 00000004 ( 4) -> N/A
EDI: 004174c4 ( 4289732) -> N/A
ESI: 004174c4 ( 4289732) -> N/A
EBP: 007bfd44 ( 16514468) -> N/A
ESP: 007bfd90 ( 16514448) -> TA@@(taCctyHcH$K8$zGxG_ADA5ATTADTB(tA?B ;B';B1<B<BTB(+ (stack)
+00: 004174c4 ( 4289732) -> N/A
+04: 00000000 ( 0) -> N/A
+08: 00a694ac ( 10917036) -> 1TA (heap)
+0c: 007bfe40 ( 16514624) -> TA?B ;B';B1<B<BTB(+P ,B(-B1-Bhc,8B=8B,8B,4.9B,hDB@DB (stack)
+10: ffffffff (4294967295) -> N/A
+14: 007bfe00 ( 16514560) -> DASATTADTB(tA?B ;B';B1<B<BTB(+P ,B(-B1-Bhc,8B=8B, (stack)
```
- disasm around:** Shows assembly instructions for the crash point.

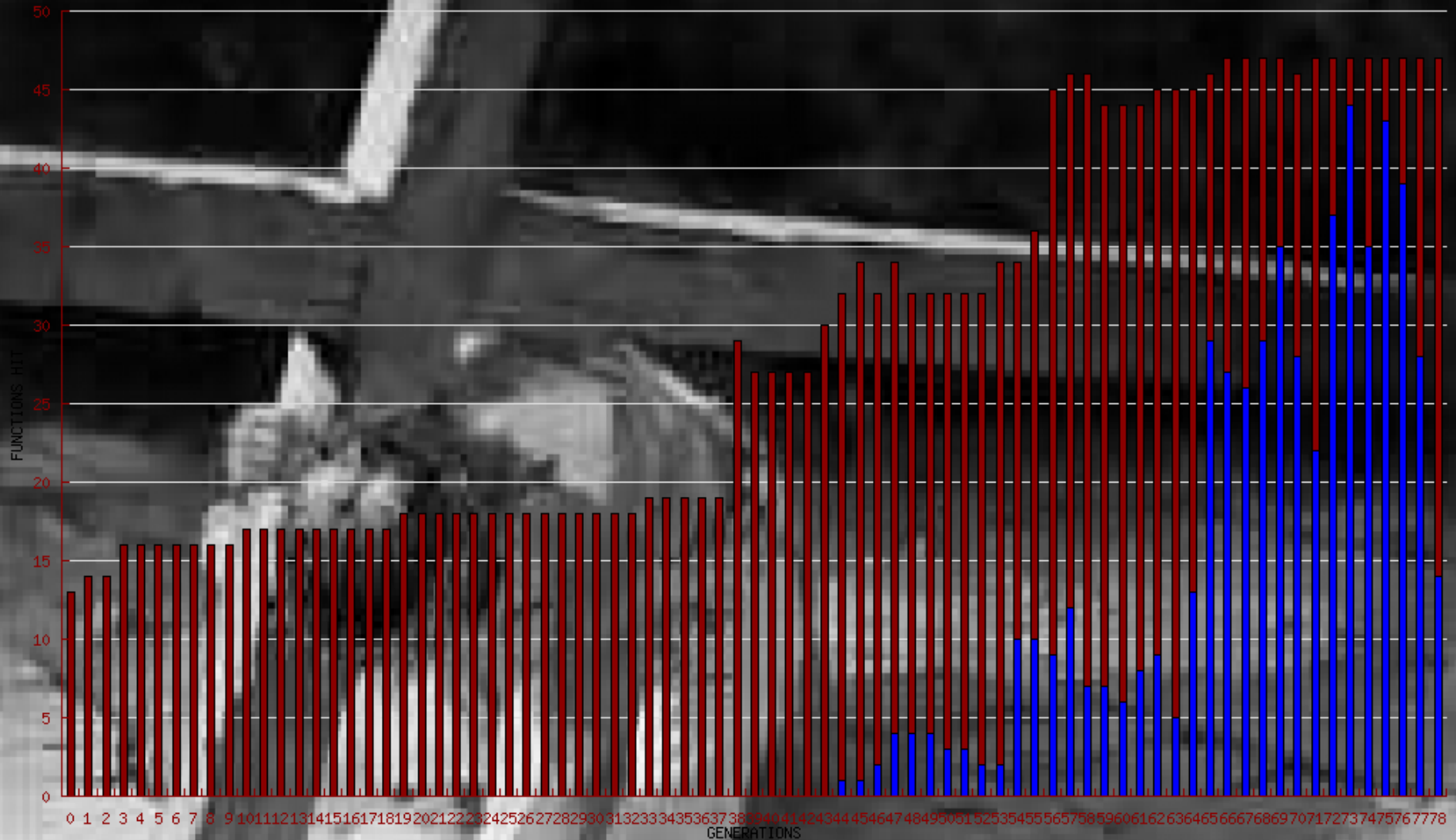
```
0x00402a02 sub edi,ecx
0x00402a04 mov edx,ecx
0x00402a06 xchg edi,esi
0x00402a08 shr ecx,0x2
0x00402a0b mov eax,edi
0x00402a0d rep movsd
0x00402a0f mov ecx,edx
0x00402a11 and ecx,0x3
0x00402a14 rep movsb
0x00402a16 pop esi
0x00402a17 xor edi,edi
```
- stack unwind:** Lists stack addresses.

```
0040f010
00418e95
00423fc5
00422cf4
```

The bottom of the window shows the Windows taskbar with the Start button, open applications (C:\jared\school\cse8..., PAIMEIconsole, Inbox - Thunderbird, runningPaimel2.PNG -...), and the system tray with the time 8:24 AM.

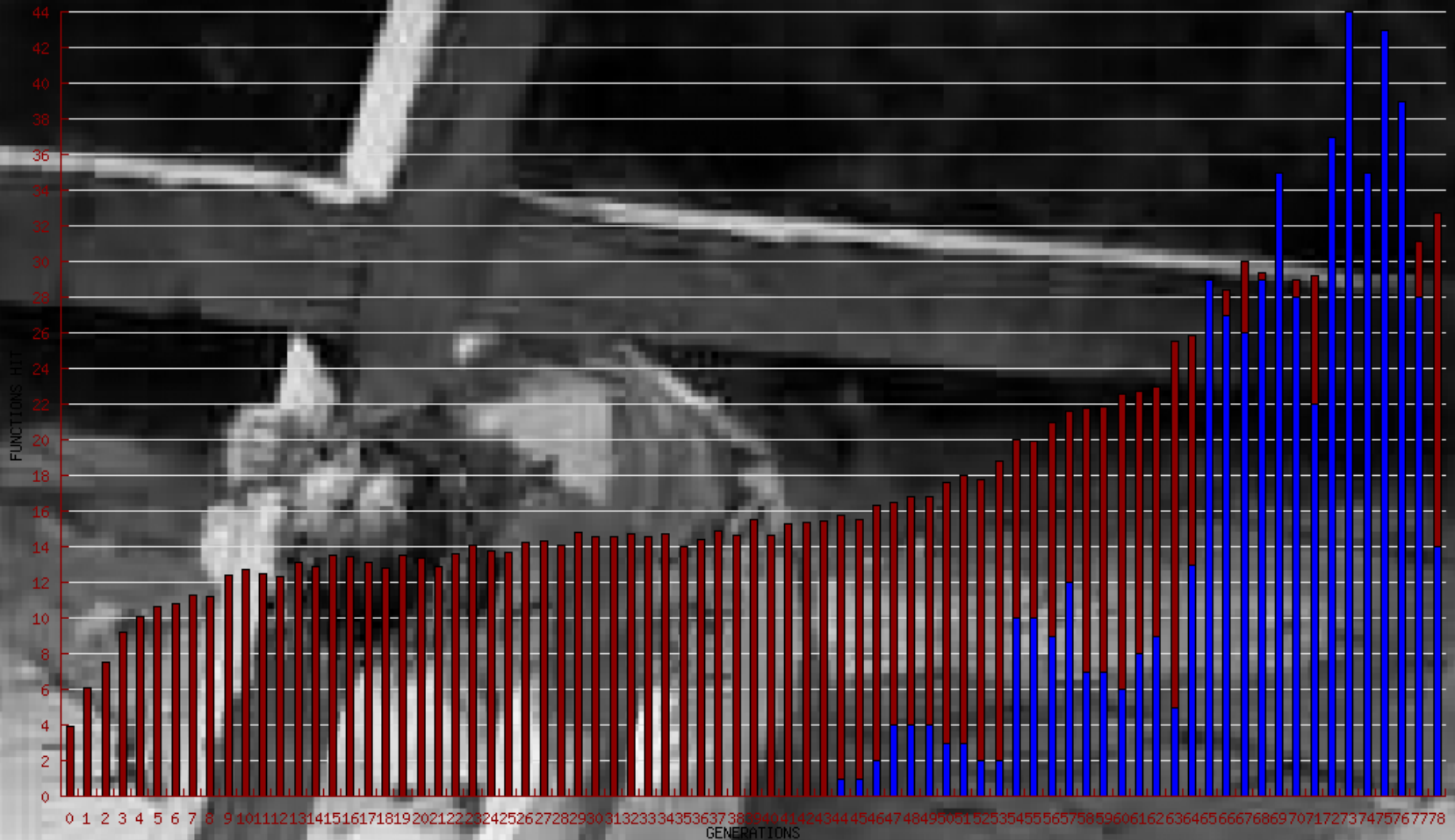
# EFS: gftp.exe Results (max) (outdated picture)

Best Session Fitness/Generation and Total Sessions that Crashed the Target

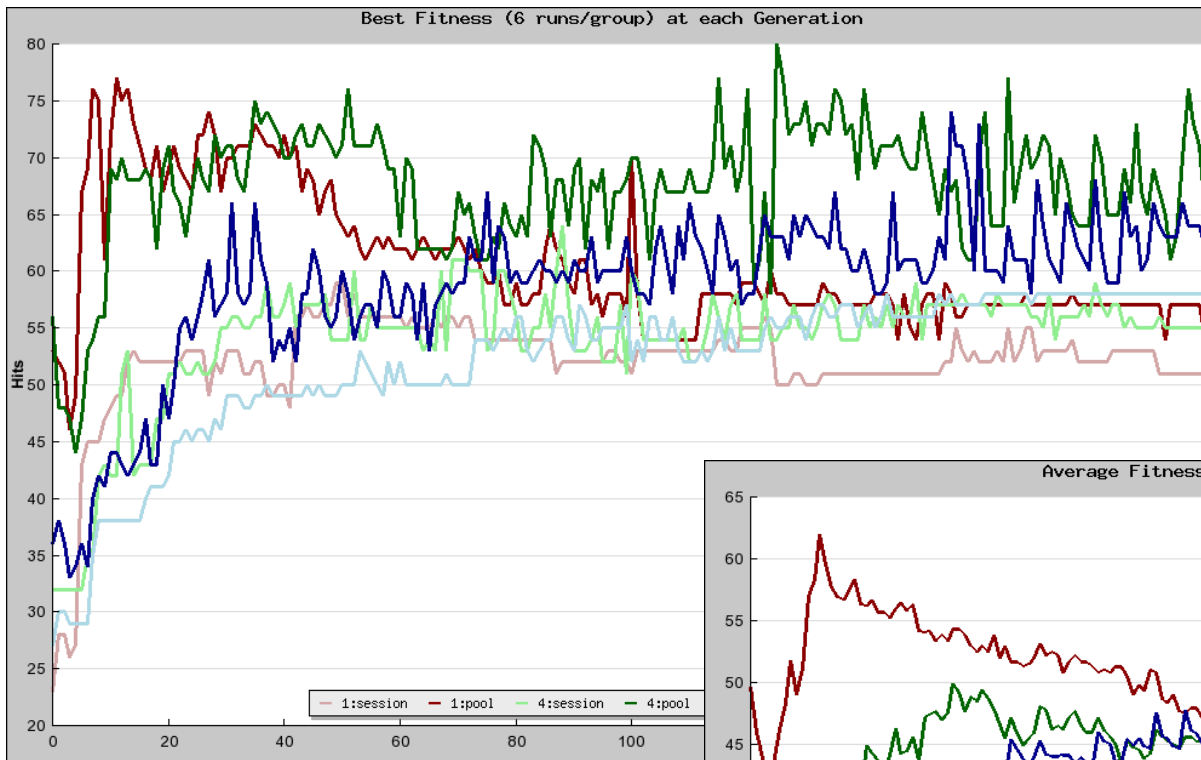


# EFS: gftp.exe Results (avg) (outdated picture)

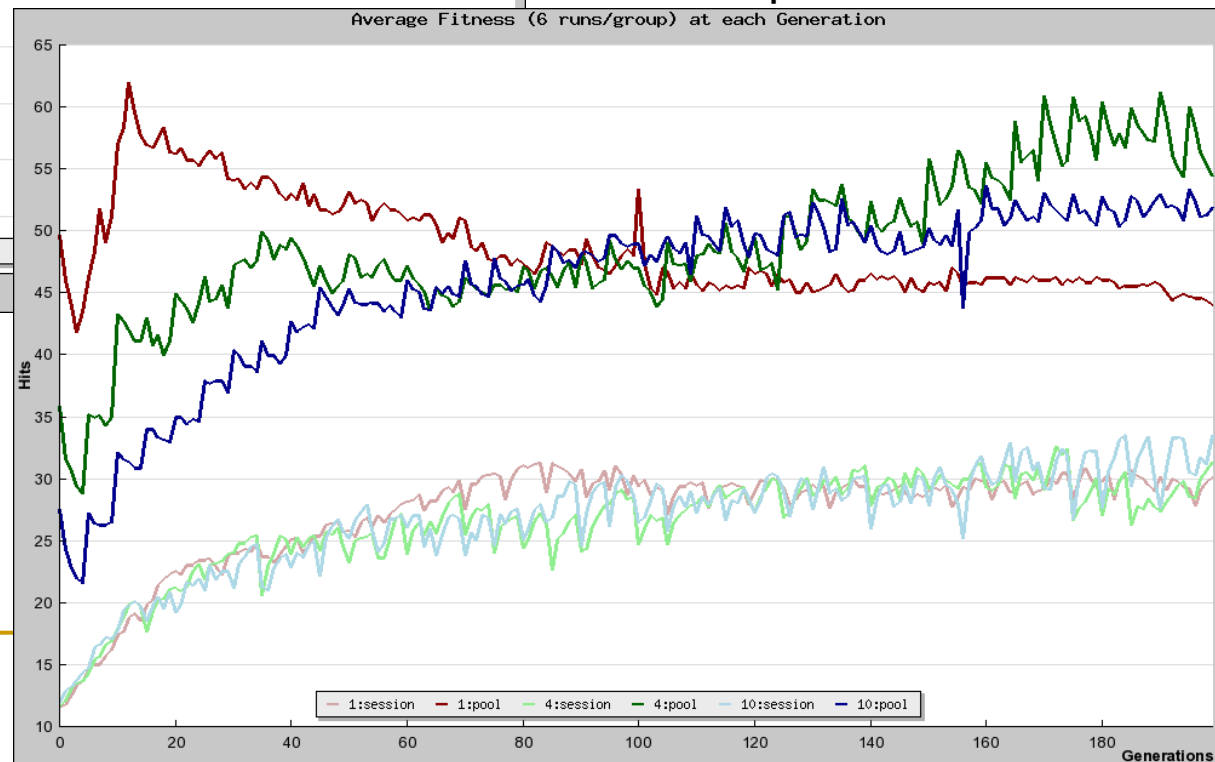
Average Session Fitness/Generation and Total Sessions that Crashed the Target



# GFTP Pool Effects – Avg over 6 runs

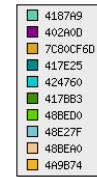
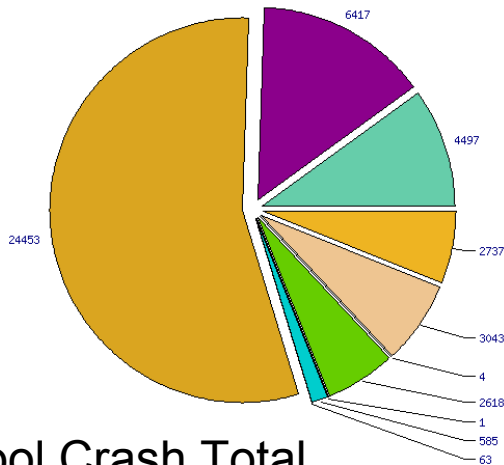
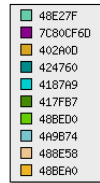
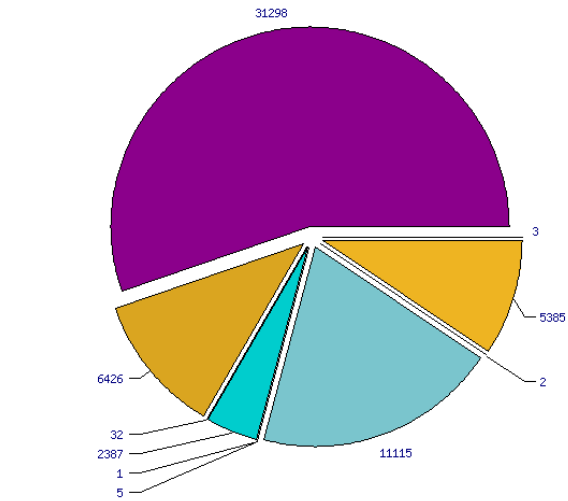


Average fitness of pool and session

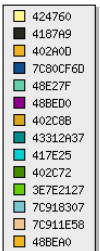
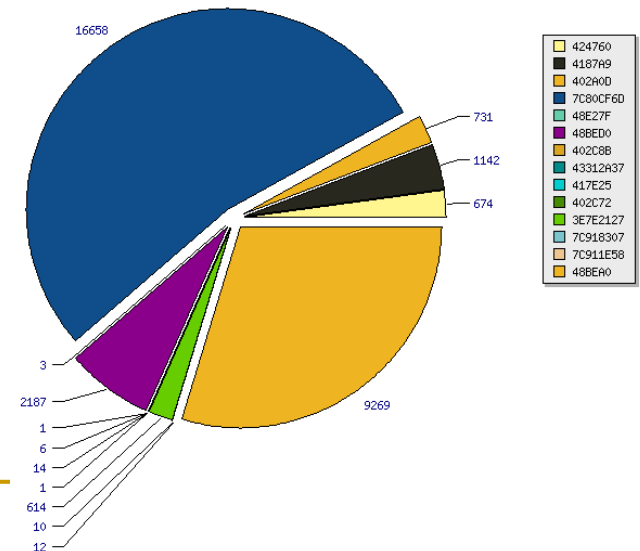


Best of Pool and Session

# Crash Results – For all Runs



10-pool Crash Total



---

# Challenges and Future Work

- Modifying EFS to work on files as well
  - How does its performance compare with existing fuzzing technologies?
    - What is the probability to find various bug types as this is the final goal of this research
      - What bugs can be found and in what software?
  - The fuzzing technology to use seems to depend on the application and general domain robustness (i.e. min work to get a bug)
    - File fuzzing == dumb fuzzing
    - Network apps == Intelligent (RFC aware) fuzzing
-

# Challenges and Future Work (cont.)

- PIDA files are great but a pain
  - Binary could be obfuscated, encrypted, or IDA just doesn't do well with it. Considered MSR, that there are issues there as well.
- Speed
  - Auto-detecting the optimal session-wait to determine if funcs or BBs is more parcticle
- Binary Protocols
  - Need more testing here
- Normal testing challenges
  - Monitoring, Instrumentation, logging, statistics, etc.

---

# References:

1. J. DeMott, R. Enbody, W. Punch, “Revolutionizing the Field of Grey-box Attack Surface Testing with Evolutionary Fuzzing”, BlackHat and Defcon 2007
  2. P. McMinn and M. Holcombe, “Evolutionary Testing Using an Extended Chaining Approach”, ACM Evolutionary Computation, Pgs 41-64, Volume 14, Issue 1 (March 2006)
  3. J. DeMott, “Benchmarking Grey-box Robustness Testing Tools with an Analysis of the Evolutionary Fuzzing System (EFS)”, continuing PhD research
-

---

# Thanks to so many!

- God
  - Family (Wonderful wife and two boys that think I'm the coolest.)
  - Friends
  - BH and DEFCON
  - Applied Security, Inc.
  - Michigan State University
  - JS -- my hacker bug from VDA Labs
  - Arun K. from Infosecwriters.com
  - L@stplace for letting me do CTF with them
-