

A series of seven horizontal blue bars of varying lengths and shapes, stacked vertically on the left side of the slide. The bars have a stepped, architectural appearance.

Fuzz Testing: Vulnerabilities and Exploit mitigation

Will Dormann [wd@cert.org]



NO WARRANTY

THIS MATERIAL OF CARNEGIE MELLON UNIVERSITY AND ITS SOFTWARE ENGINEERING INSTITUTE IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

Outline

- Vulnerability Analysis
- Fuzz Testing
 - BFF
 - FOE
- Real-world fuzzing example
- Exploitation protection
 - Microsoft EMET
- Future plans



Vulnerability Analysis at CERT

CERT Vulnerability Analysis

Mission: Reducing the birth rate and increasing the death rate of software vulnerabilities



Discovery

Disclosure

Remediation

Vulnerability Discovery

Software systems continue to be plagued by security vulnerabilities caused by underlying software defects

Goals:

- Help vendors and developers discover vulnerabilities before software is fielded
- Reduce the cost of improving software assurance

Vulnerability Discovery

Develop and improve practical tools and techniques to find software vulnerabilities

- Static analysis
- Dynamic analysis
 - Current focus is on fuzz testing

Software security quality assurance

- Feeds back into the vulnerability remediation process



Fuzz Testing

Fuzz Testing

Providing unexpected, invalid, or random data to an application with the intention of finding bugs.

- Unexpected behavior
- Crashes
 - Buffer overflows
 - Integer overflows
 - Format string

Vulnerabilities

Types of Fuzzing

Mutation (“dumb”)

- Semantics-less modification of input – “flip random bits”

Generational

- Semantics-aware modifications of input – “protocol and format aware”

Concolic – concrete and symbolic

- Using symbolic representation for code coverage

While the least sophisticated, CERT continues to focus on mutation fuzzing due to a continued high success rate

Mutation Fuzzing Challenges

Much of the research into black-box negative input software testing (i.e., fuzz testing) has focused on making tools more aware of the protocol or data structure they are targeting

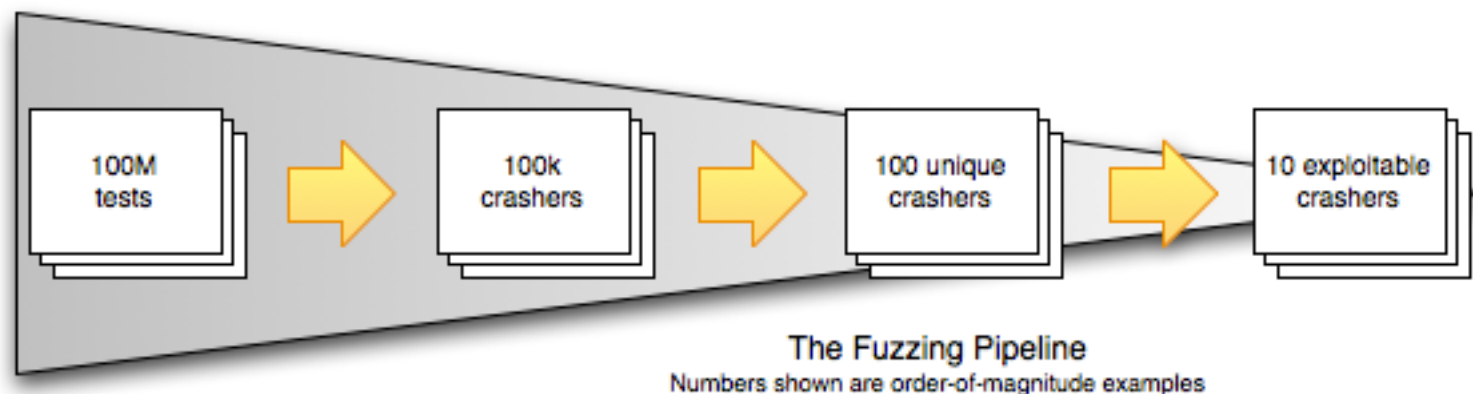
- Incurs high up-front costs to model input/protocol
- Easy to omit large branches of test cases

Developers require very generic fuzz testing tools that can apply to lots of software

Mutation Fuzzing Challenges (2)

Mutational fuzz testing produces thousands or even millions of crashing test cases that need to be identified

- A majority of the results are duplicates resulting from the same underlying software defect
- Developers and researchers need a metric of exploitability



CERT's Approach

Create very generic fuzz testing tools that can apply to lots of software

Be entirely blind to context and underlying protocol

Apply core principles of fuzz testing to a broader range of software and improve their overall efficacy

Use feedback from the cumulative performance of a testing campaign as input to the mutation algorithm and seed file selection

Fuzzing Basics

1. Mangle input (mutate or generate)
 1. Choose input file to mangle
 2. Decide how much to mangle it
2. Run target application
3. Detect exceptions (did it crash?)
4. Filter out non-unique crashes (is it new?)
5. Triage severity (how exploitable is it?)



Fuzzing on Linux and OS X: The CERT BFF

Fuzz Testing

Problem:

Fuzzing isn't rocket science, but it does require work to set up a fuzzing environment.

Solution:

The CERT® BFF

<https://www.cert.org/vuls/discovery/bff.html>

Basic Fuzzing Framework



* It's not you, it's me

BFF Components

Debian Linux virtual machine (VMware)

- Optimized for fuzzing
- zzuf, valgrind, gdb
- Software watchdog

Fuzzing scripts

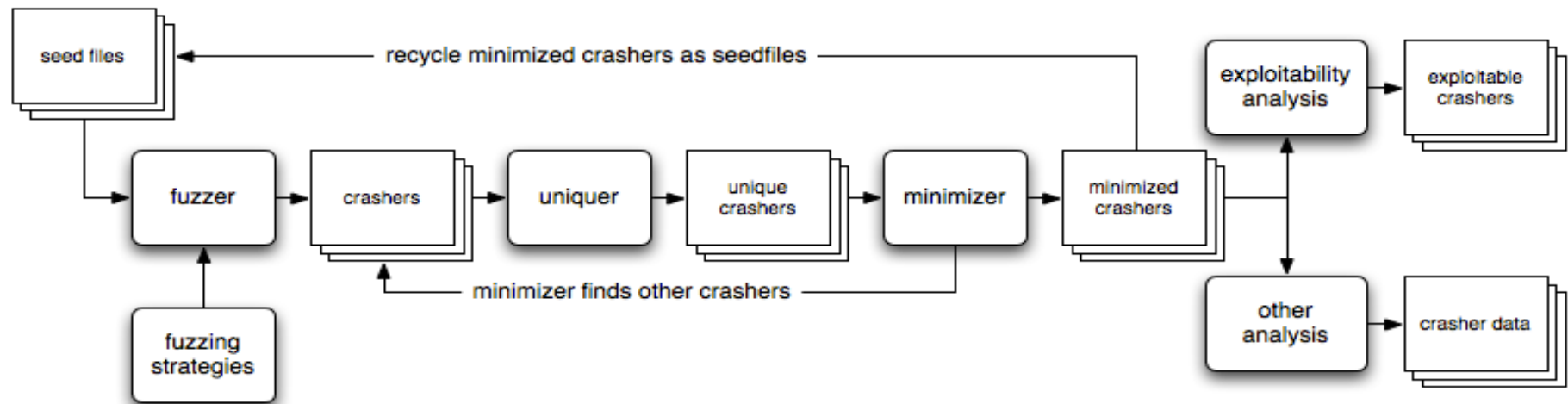
- Testcase generation
- Process killer
- Crash verification
- Crash deduplication
- Crash minimization

BFF Architecture

Perform multiple levels of results reduction

- Normalize results and remove duplicates
- Minimize crashing input to the minimum bytes to reproduce the crash

Sort final unique results by exploitability and clusters of crashes – “hot spots”

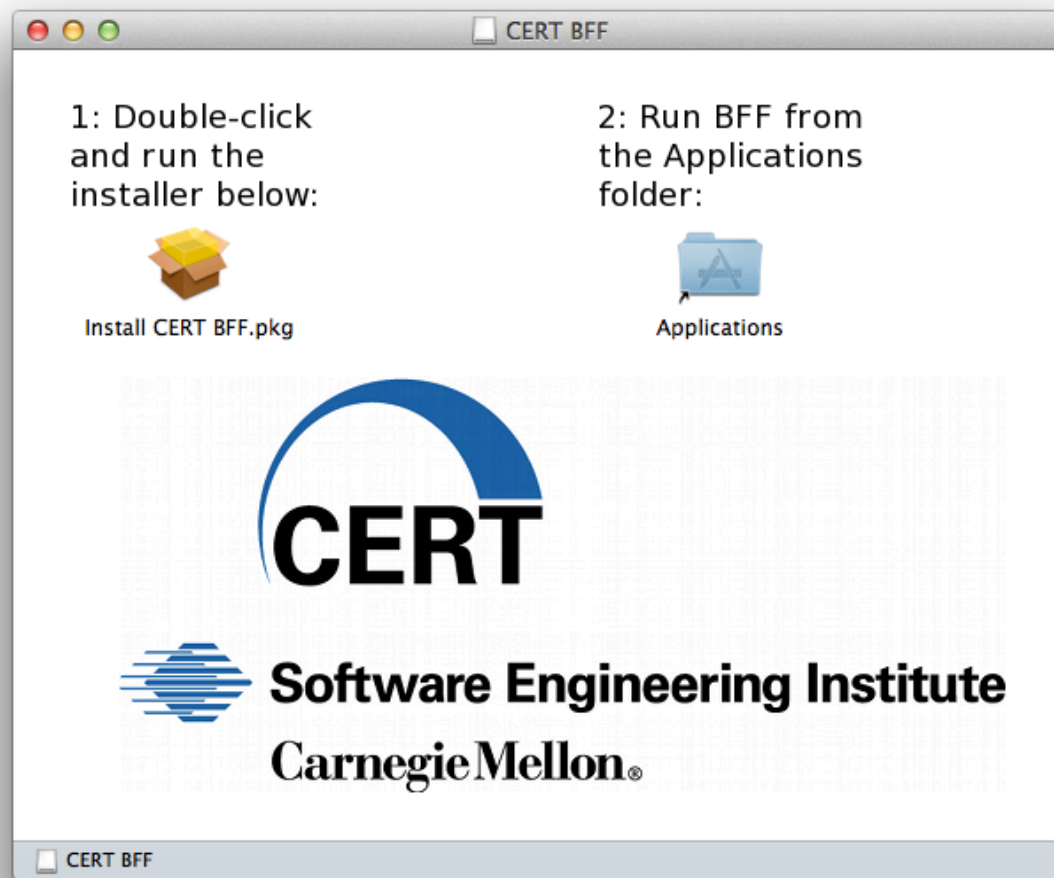


BFF Requirements

Prerequisites:

- Ability to unzip a file
- Ability to power on a VMware virtual machine

BFF on OS X



Flash Fuzzing VM

```
wd@wd-desktop: ~  
***  
output file is: /home/wd/fuzzing/flashplayer/2e668cf5eca695fc5f10838232f767f8swf  
/2e668cf5eca695fc5f10838232f767f8.swf,400141549  
checking crash count in /mnt/xcat/fuzzing/crashers/flashplayer/fuzz/d57287b3d5a9  
518cf1e30f0a157da668  
Fuzzing: 2e668cf5eca695fc5f10838232f767f8.swf, Range: 400141550-400142050, Total  
crashes: 431  
generating testcase for zzuf[s=400141613,r=1e-05;0.05]: signal 11 (SIGSEGV)  
***  
output file is: /home/wd/fuzzing/flashplayer/2e668cf5eca695fc5f10838232f767f8swf  
/2e668cf5eca695fc5f10838232f767f8.swf,400141613  
checking crash count in /mnt/xcat/fuzzing/crashers/flashplayer/fuzz/d57287b3d5a9  
518cf1e30f0a157da668  
Fuzzing: 2e668cf5eca695fc5f10838232f767f8.swf, Range: 400141614-400142114, Total  
crashes: 432  
generating testcase for zzuf[s=400142015,r=1e-05;0.05]: signal 11 (SIGSEGV)  
***  
output file is: /home/wd/fuzzing/flashplayer/2e668cf5eca695fc5f10838232f767f8swf  
/2e668cf5eca695fc5f10838232f767f8.swf,400142015  
checking crash count in /mnt/xcat/fuzzing/crashers/flashplayer/fuzz/d57287b3d5a9  
518cf1e30f0a157da668  
Fuzzing: 2e668cf5eca695fc5f10838232f767f8.swf, Range: 400142016-400142516, Total  
crashes: 433  
2926 root -2 0 1856 1852 1580 S 0,3 0,4 0:01.14 watchdog  
3344 wd 20 0 10748 2044 1792 R 0,3 0,4 0:53.73 xterm  
3345 wd 20 0 10748 2844 1760 S 0,3 0,6 0:38.07 xterm  
31538 wd 20 0 1996 612 532 R 0,3 0,1 0:00.03 zzuf  
1 root 20 0 1908 564 508 S 0,0 0,1 0:03.46 init  
2 root 15 -5 0 0 0 S 0,0 0,0 0:00.00 kthreadd  
3 root RT -5 0 0 0 S 0,0 0,0 0:00.00 migration/0  
5 root RT -5 0 0 0 S 0,0 0,0 0:00.00 watchdog/0  
6 root 15 -5 0 0 0 S 0,0 0,0 0:00.16 events/0  
7 root 15 -5 0 0 0 S 0,0 0,0 0:00.00 khelper  
8 root RT -5 0 0 0 S 0,0 0,0 0:00.00 kstop/0  
9 root 15 -5 0 0 0 S 0,0 0,0 0:00.00 kintegrityd/0  
10 root 15 -5 0 0 0 S 0,0 0,0 0:01.02 kblockd/0  
Workspace 1 10 Feb, Wed 09:29:22 wd@wd-desktop: ~ top
```



Fuzz Testing Variables and Solutions

Fuzzing Variables

Fuzzing effectiveness depends on many variables:

- Fuzzer
- Mutation strategy
- Seed File
 - Program used to generate
 - Options used for generation
 - Size

Seed file selection

Some input files reveal more unique crashes under fuzzing than others

- Different files induce different code coverage

Objective: Focus attention on the files that are more productive

Seed file selection method

Model fuzzing as Bernoulli trials and unique crashes as Poisson-distributed random events

For each seed file, maintain a confidence interval on the expected crash density based on empirical measurement during the course of a fuzz campaign

Choose seed files with likelihood in proportion to their expected crash density

Result: Seed files that yield more crashes get more attention

How much to mangle?

Too much:

- 'breaks the file' → missing code coverage
- Some bugs won't be found

Too little:

- Results take too long
- Some bugs won't be found

Solution: Rangefinder

Segment proportion of file to be fuzzed into ranges

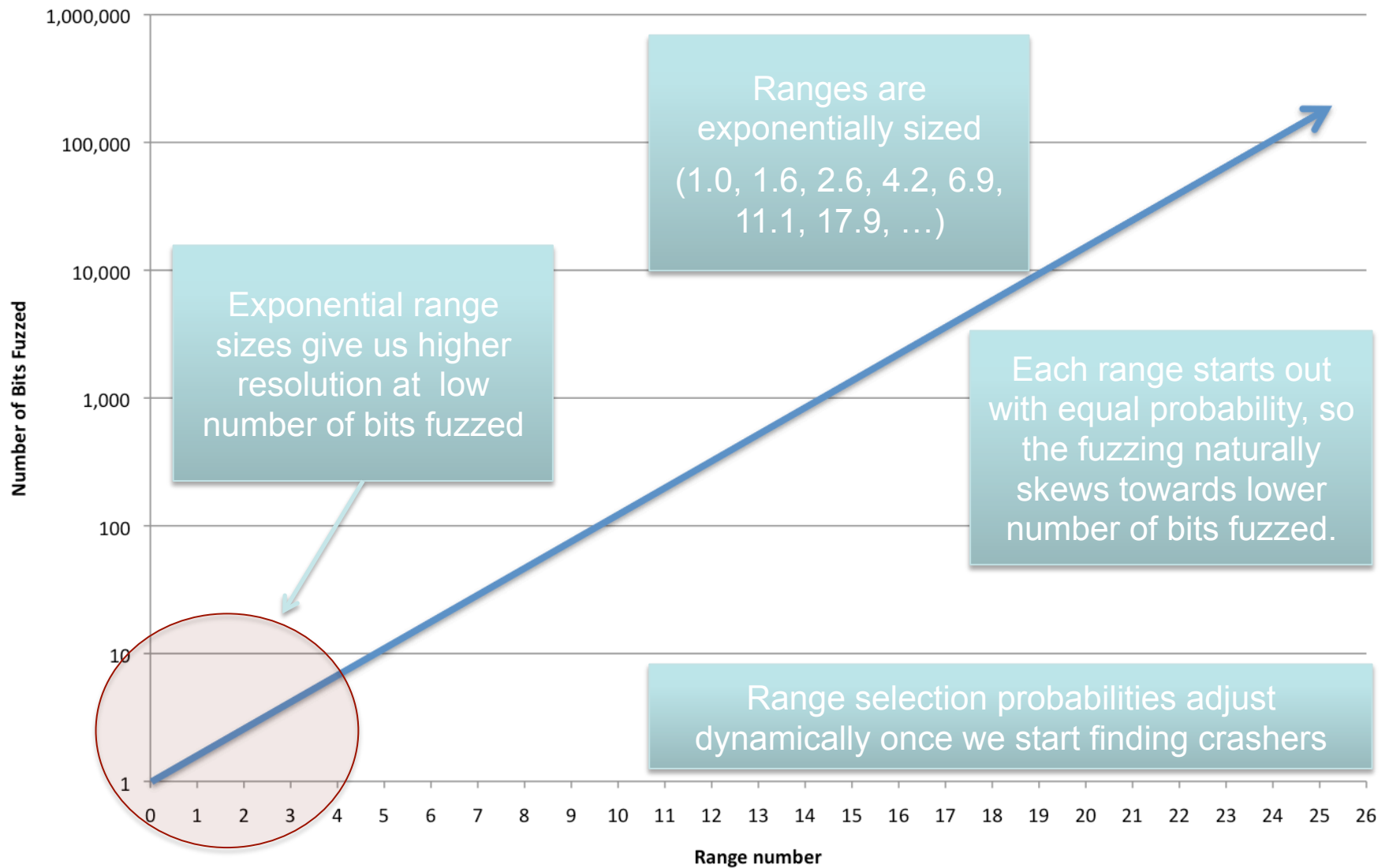
- fuzz 1 bit all the way up to ~100% of the bits
- range widths grow exponentially

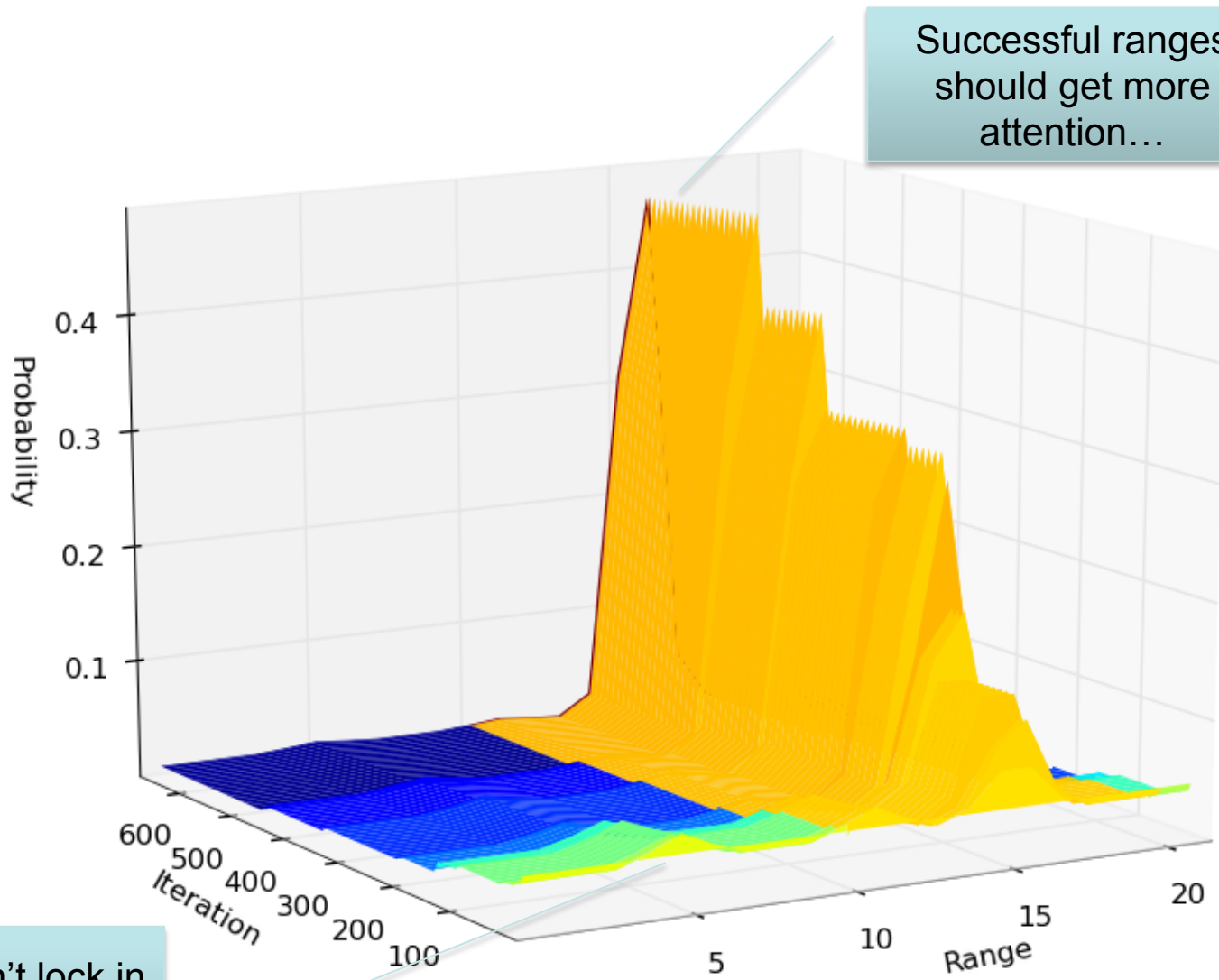
Prefer higher granularity at lower proportion of mangled bits

Each unique crash encountered increases range score

Pick next range based on probability distribution derived from the range's score

Avg Bits Fuzzed by Range





Problem: Volume of crashing test cases

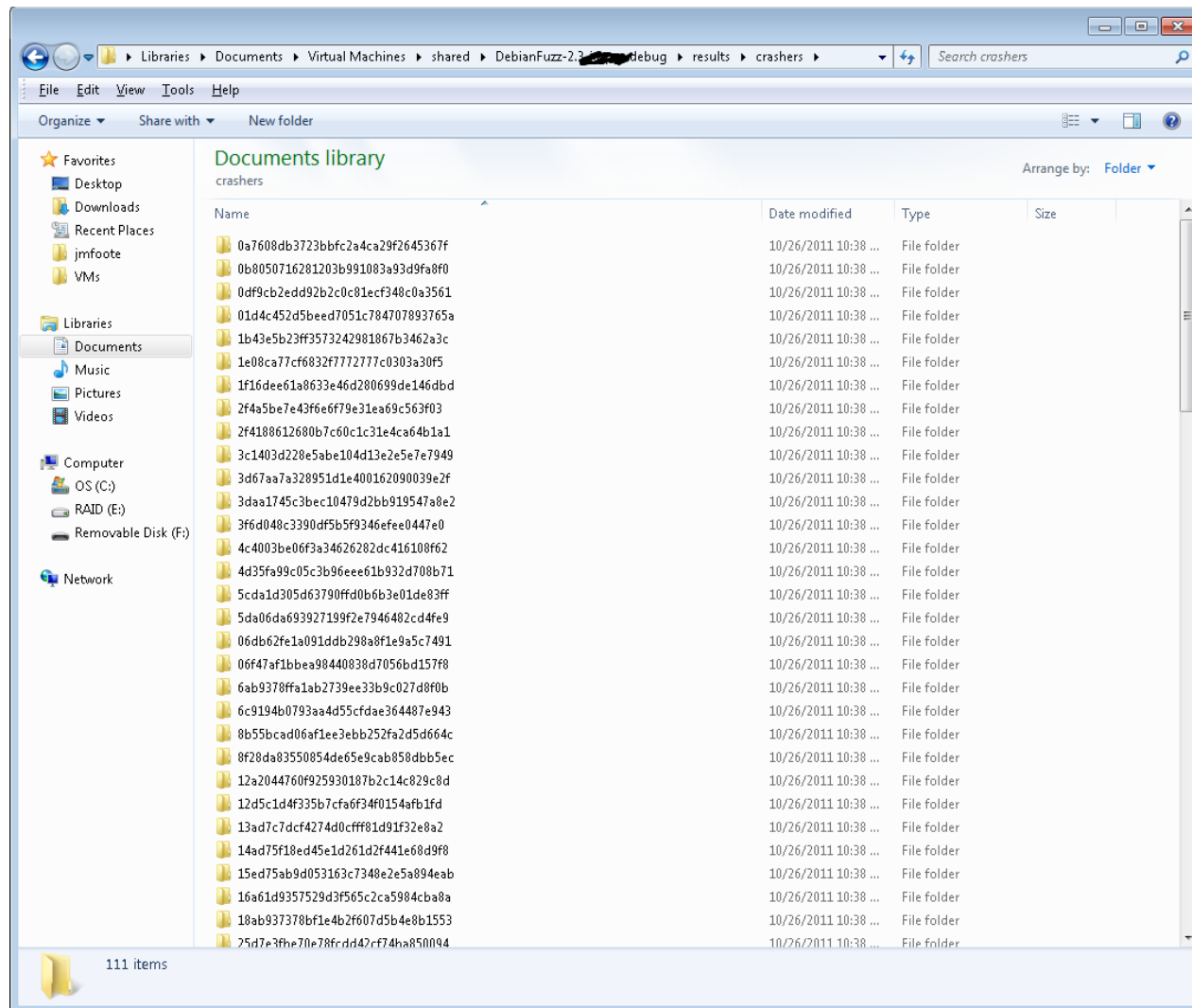
File fuzzing can yield a large number of crashing test cases

Improvements to BFF have dramatically increased the number of crashes available for analysis

- BFF run on widely-used open-source J2K codec yielded 111 unique crashers in a few days

Our capacity to find crashes outstrips our ability to analyze them using traditional human-oriented techniques

Where to start?



Solution: lightweight automated analysis

Perform a quick automated analysis to find test cases that present security vulnerabilities

For each test case

1. Run crashing test case under a debugger
2. Examine application state
3. Determine “exploitability”

Existing solutions for Windows and OSX

Windows

- WinDbg + MSEC !exploitable extension
- Used by CERT FOE

OSX

- Apple CrashWrangler
- Used by CERT BFF on OSX

Linux

- Couldn't find anything that does this exactly
- Valgrind memcheck, (rumored) private debuggers

Solution for Linux: CERT triage tools

“exploitable” extension for GDB

- GDB is the most widely available debugger for Linux
- Implemented on nascent GDB Python API available in versions > 7.1
- Determines exploitability of a single test case

“triage” example batch script

- Python script that wraps multiple calls to GDB + exploitable
- Determines exploitability of a corpus of crashing test cases

“exploitable” output

```
Program received signal SIGSEGV, Segmentation fault.
memcpy () at ../sysdeps/i386/i686/memcpy.S:75
75      ../sysdeps/i386/i686/memcpy.S: No such file or directory.
      in ../sysdeps/i386/i686/memcpy.S
(gdb) source exploitable.py
(gdb) exploitable
Description: Possible stack corruption
Short description: PossibleStackCorruption (6/20)
Hash: dc64d713b1eb2f213638e3aa329f27fa.dc64d713b1eb2f213638e3aa329f27fa
Exploitability Classification: EXPLOITABLE
Explanation: GDB generated an error while unwinding the stack and/or the stack contained return addresses that were not mapped in the inferior's process address space and/or the stack pointer is pointing to a location outside the default stack region. These conditions likely indicate stack corruption, which is generally considered exploitable.
Other tags: BlockMoveAv (13/20), SourceAv (15/20)
(gdb) █
```

“triage” output

```
EXPLOITABLE: SegFaultOnPc
/mnt/hgfs/fuzz/results/crashers/048aeba70a426bb162db81befc3240c9/sf_19d59a74213e2410ce6b9f86b1b57e46-15594305.j2k
/mnt/hgfs/fuzz/results/crashers/0a7608db3723bbfc2a4ca29f2645367f/sf_9db558c6149b7f52e24030fc177138e5-10068443.j2k
/mnt/hgfs/fuzz/results/crashers/0df9cb2edd92b2c0c81ecf348c0a3561/sf_9db558c6149b7f52e24030fc177138e5-12790010.j2k
/mnt/hgfs/fuzz/results/crashers/3083553e92e188fbc5f04b3208b281b4/sf_9db558c6149b7f52e24030fc177138e5-155534.j2k
/mnt/hgfs/fuzz/results/crashers/3f6d048c3390df5b5f9346fee0447e0/sf_19d59a74213e2410ce6b9f86b1b57e46-26302406.j2k
/mnt/hgfs/fuzz/results/crashers/55e22c6a6e2193a99f1739a843d6201c/sf_9db558c6149b7f52e24030fc177138e5-8279699.j2k
/mnt/hgfs/fuzz/results/crashers/6933ea7f9529662eeec8c90514ae7a5/sf_19d59a74213e2410ce6b9f86b1b57e46-13415758.j2k
/mnt/hgfs/fuzz/results/crashers/912b7000029428daa2f0f6a4c1ecfb35/sf_19d59a74213e2410ce6b9f86b1b57e46-27405371.j2k
/mnt/hgfs/fuzz/results/crashers/99e7c2a0f5c59b4d01e1204e31e20264/sf_19d59a74213e2410ce6b9f86b1b57e46-4360719.j2k
/mnt/hgfs/fuzz/results/crashers/aedc3ad8713a9c3a38f3bfa552605b4d/sf_9db558c6149b7f52e24030fc177138e5-10694045.j2k
/mnt/hgfs/fuzz/results/crashers/c43374d9b9daa3e65f31336eb347efcc/sf_19d59a74213e2410ce6b9f86b1b57e46-24030138.j2k
/mnt/hgfs/fuzz/results/crashers/fbf811a892ae4b4a110220bbf0d1fb65/sf_9db558c6149b7f52e24030fc177138e5-16455967.j2k

EXPLOITABLE: PossibleStackCorruption
/mnt/hgfs/fuzz/results/crashers/8f28da83550854de65e9cab858dbb5ec/sf_c4f93705986f7bb49103ee1788f0785d-14884466.j2k (BlockMoveAv) (SourceAv)

EXPLOITABLE: DestAv
/mnt/hgfs/fuzz/results/crashers/794b5908263368b01ada6548755d9576/sf_dea42617b9d8a286c6c0768050668df2-13060382.j2k
/mnt/hgfs/fuzz/results/crashers/97e588d2fe17bf5b3fa29b762687adba/sf_9291b984ed7a03169246ab5cda1fa301-1468817.jp2
/mnt/hgfs/fuzz/results/crashers/bfff89809721cdd64f6e176d978c889a/sf_422b371ea60d1f8766e86f914fda134f-742911.jp2

EXPLOITABLE: HeapError
/mnt/hgfs/fuzz/results/crashers/12a2044760f925930187b2c14c829c8d/sf_65cc567685f54ddf23e7d13e3b034d30-5141.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/13857d2feef6227ee8cddb43c01be59ce/sf_76905a851f5ee1fc4ad828431bcce5b5-27561820.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/16a61d9357529d3f565c2ca5984cba8a/sf_3fba9442b5afd7329e708852c52d3451-768216.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/1f16dee61a8633e46d280699bd146dbd/sf_19d59a74213e2410ce6b9f86b1b57e46-6877704.j2k (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/2f4188612680b7c60c1c31e4ca64b1a1/sf_f65284e16ba712fadad41d1ffd18242f-25562005.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/2f4a5be7e43f6e6f79e31ea69c563f03/sf_cf221b76e7875304939b585523c40fb5-1147.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/3c1403d228e5abe104d13e2e5e7e7949/sf_f65284e16ba712fadad41d1ffd18242f-3109640.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/4082d74b46f2b0239128e59825d4a787/sf_dbd80193e081acf111e4744356922b1a-8331.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/40ead2f79d2198bcaebbd94a46ba6026/sf_dbd80193e081acf111e4744356922b1a-1955414.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/5143610306886a6b763851949de7080c/sf_512e53a782f4cfb4054f06bc259cfe0f-764294.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/5cda1d305d63790ffd0b6b3e01de83ff/sf_76905a851f5ee1fc4ad828431bcce5b5-534132.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/6009d9ea20d063c77804b4cc4fa1c264/sf_65cc567685f54ddf23e7d13e3b034d30-1639163.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/629dfc07884ea43f2c44bc46aa66842/sf_4d55b040cfc6748007ee53a6c0e4528e-12087.j2k (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/65841a85bd9dd36c1df59c997ff38040/sf_7a09938f8e57cef8ca2f7bc7cc9da978-1954601.j2k (AbortSignal)
```

Test case minimization

Why minimize?

- Fuzzed test cases can significantly alter the code coverage through the executable
- Many of those differences may not be relevant to the crash

Goal: Find the test case that

- (1) is minimally different from the known good seed file
 - (2) still causes the same crash
- 'same crash' = match the last N entries in back trace

(we typically choose N=5)

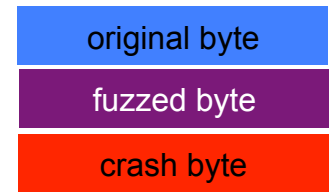
Steps to a solution

Figure out how much to attempt to revert based on what we know (or can guess)

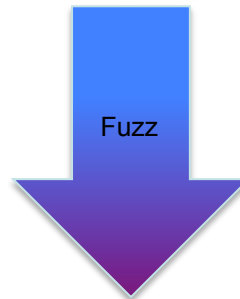
Test to see if we still see the same crash

Iterate and update strategy based on what we learn

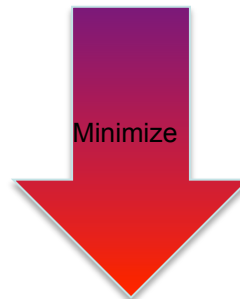
What Minimizer Does



Known good seedfile — does not cause crash

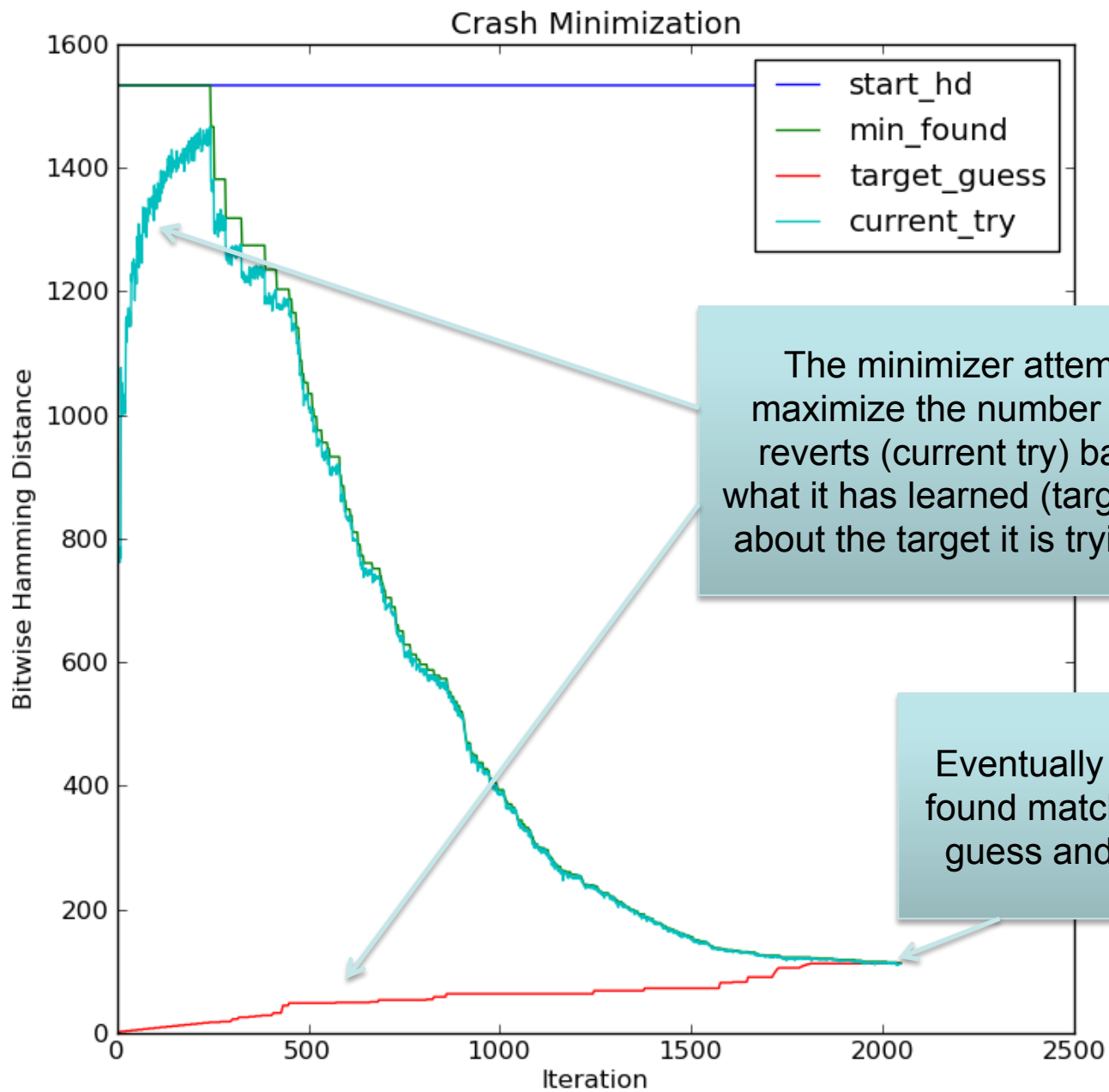


Fuzzed file — causes crash, many changed bytes are not involved in the crash



Minimized fuzzed file — causes same crash, all changed bytes are involved in the crash





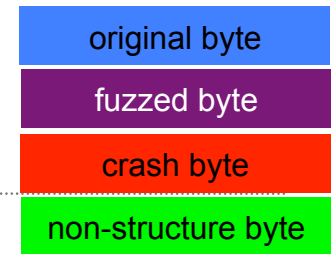
Minimize to string

Standard minimization gives the minimally-different-from-seed-file test case. But which of those bytes are irrelevant to the crash?

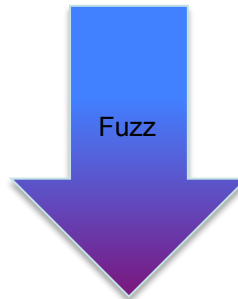
We want to know:

- Bytes required for processing (Structure)
- Bytes required to trigger crash (Vulnerability)

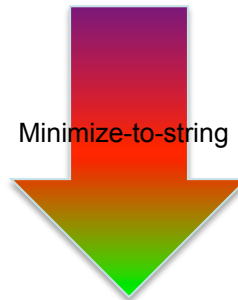
What Minimize-to-String Does



Known good seedfile — does not cause crash



Fuzzed file — causes crash, many changed bytes are not involved in the crash

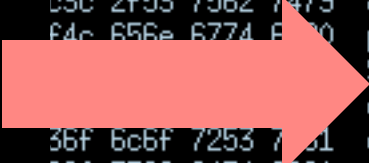


Minimized-to-string file — causes same crash, replaces non-structure bytes



Minimize to string example

```
0004bb0: 1248 6f8d 1061 6e89 6e74 2030 2f48 6f8c 7878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004bc0: 6f72 5370 6163 6520 3337 2030 2052 2f57 orSpace 37 0 R/W 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004bd0: 6964 7468 2035 3030 2f48 6569 6768 7420 idth 500/Height 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004be0: 322f 5479 7065 2f58 4f62 6a65 6374 3e3e 2/Type/XObject>> 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004bf0: 7374 7265 616d 0d0a 68de 6260 1805 a360 stream,.h,b'...' 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004c00: 140c 7700 1060 0003 e800 010a 0d0a 656e ..w.. .....en 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004c10: 6473 7472 6561 6d0d 656e 646f 626a 0d35 dstream.endobj.5 878 7878 7878 0d35 xxxxxxxxxxxxxxxx.5
0004c20: 3820 3020 6f62 6a0d 3c3c 2f53 7562 7479 8 0 obj.<</Subty c3c 2f53 7562 7479 8 0 obj.<</Subty
0004c30: 7065 2f49 6d61 6765 2f4c 656e 6774 6820 pe/Image/Length f4c 656e 6774 6820 pe/Image/Length
0004c40: 3939 382f 4669 6c74 6572 2f44 4354 4465 998/Filter/DCTDe 9xx/Filter/DCTDe
0004c50: 636f 6465 2f42 6974 7350 6572 436f 6d70 code/BitsPerComp code/BitsPerComp
0004c60: 6f6e 656e 7420 382f 436f 6c6f 7253 7061 onent 8/ColorSpa 36f 6c6f 7253 7061 onent 8/ColorSpa
0004c70: 6365 2032 3920 3020 522f 5769 6474 6820 ce 29 0 R/Width 22f 5769 6474 6820 ce 29 0 R/Width
0004c80: 3134 352f 4865 6967 6874 2031 392f 5479 145/Height 19/Ty 874 2031 782f 7878 1xx/Height 1x/xx
0004c90: 7065 2f58 4f62 6a65 6374 3e3e 7374 7265 pe/XObject>>stre 878 3e3e 7374 7265 xx/xxxxxxx>>stre
0004ca0: 616d 0d0a ffd8 ffee 000e 4164 6f62 6500 am.....Adobe. 878 7878 7878 7878 am,x.,xxxxxxxxxx
0004cb0: 6480 0000 0001 ffdb 0084 000c 0808 0809 d..... 084 0078 7878 7878 xxxxxx.....xxxxx
0004cc0: 080c 0909 0c11 0b0a 0b11 150f 0c0c 0f15 ..... 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004cd0: 1813 1315 1313 1817 1214 1414 1412 1717 ..... 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004ce0: 1b1c 1e1c 1b17 2424 2727 2424 3533 3333 .....$#' '$5333 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004cf0: 353b 3b3b 3b3b 3b3b 3b3b 3b01 0d0b 0b0d 5;::::::::::..... 878 7801 7878 7878 xxxxxxxxxxxxxx,xxxxx
0004d00: 0e0d 100e 0e10 140e 0f0e 1414 1011 1110 ..... 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004d10: 141d 1414 1514 141d 251a 1717 1717 1a25 .....%.....% 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004d20: 2023 1e1e 1e23 2028 2825 2528 2832 3230 #...# ((%((220 878 7878 7878 7878 xxxxxxxxxxxxxxxx
0004d30: 3232 3b3b 3b3b 3b3b 3b3b 3b3b ffc0 0011 22;::::::::::..... 878 7878 ffc0 0011 xxxxxxxxxxxxxx....
0004d40: 0800 1300 9103 0122 0002 1101 0311 01ff ..... " ..... 002 1101 0311 01ff .xxxxx." .....
0004d50: c401 3f00 0001 0501 0101 0101 0100 0000 ..?..... 101 0101 0100 0000 ..?.....
0004d60: 0000 0000 0300 0102 0405 0607 0809 0a0b 878 7878 7878 7878 xxxxxxxxxxxxxxxx
```



Minimize to string downside

It's a more complex problem to solve.

- It's slow!

Mitigation:

Only run it for cases that you want to write a PoC for.

Writing a PoC

Achieving code execution with a memory corruption vulnerability requires two pieces of knowledge:

1. What bytes are under my control?
2. How do I get there?

The original crash

The screenshot shows the Immunity Debugger interface for EnCase.exe. The CPU window displays the following assembly code and registers:

Address	Hex dump	ASCII
00CB4000	56 65 72 73 69 6F 6E 3A	Version:
00CB4008	20 32 2E 35 30 00 44 61	2.50.Da
00CB4010	74 65 39 20 41 75 67 20	te: Aug
00CB4018	20 38 20 32 30 30 37 00	8 2007.
00CB4020	00 00 00 00 68 61 73 70	...hasp
00CB4028	66 6F 72 60 61 74 00 00	format..
00CB4030	66 6F 72 60 61 74 00 00	format..
00CB4038	73 65 73 73 69 6F 6E 69	sessioni
00CB4040	6E 66 6F 00 3C 68 61 73	nfo.<has
00CB4048	70 66 6F 72 60 61 74 20	pformat
00CB4050	66 6F 73 60 61 74 3D 22	format="
00CB4058	73 65 73 73 69 6F 6E 69	sessioni
00CB4060	6E 66 6F 22 2F 3E 00 00	nfo"/>..
00CB4068	75 70 64 61 74 65 69 6E	updatein
00CB4070	66 6F 00 00 3C 68 61 73	fo.<has
00CB4078	70 66 6F 72 60 61 74 20	pformat
00CB4080	66 6F 72 60 61 74 3D 22	format="
00CB4088	75 70 64 61 74 65 69 6E	updatein
00CB4090	66 6F 22 2F 3E 00 00 00	fo"/>... fastupda
00CB4098	66 61 73 74 75 70 64 61	teInfo..
00CB40A0	74 65 69 6E 66 6F 00 00	<haspfor
00CB40A8	3C 68 61 73 70 66 6F 72	mat form
00CB40B0	6D 61 74 20 66 6F 72 60	at="fast
00CB40B8	61 74 3D 22 66 61 73 74	updatein
00CB40C0	75 70 64 61 74 65 69 6E	fo"/>... F... 00610000
00CB40C8	66 6F 22 2F 3E 00 00 00	

The Registers (FPU) window shows the following values:

Register	Value
EAX	00000000
ECX	69660000
EDX	7C9032BC ntdll.7C9032BC
EBX	00000000
ESP	0216C14C
EBP	0216C16C
ESI	00000000
EDI	00000000
EIP	69660000

The status bar at the bottom indicates: [10:33:45] Access violation when executing [69660000] - use Shift+F7/F8/F9 to pa Paused

Which 0x78787878 ?

Minimization to x shows:

- Which bytes are under my control ('xxxx...')
- How to get there (JMP ECX)

The problem:

Which 'x' is which?

The solution:

Metasploit string pattern.

The minimized-to-Metasploit crash

Immunity Debugger - EnCase.exe

File View Debug Plugins ImmLib Options Window Help Jobs

l e m t w h c P k b z r ... s ? Immunity: Consulting Services Ma

CPU - main thread, module vsgdsf

```
03B4744B 88041F MOV BYTE PTR DS:[EDI+EBX],AL
03B4744E 83C7 01 ADD EDI,1
03B47451 3BFD CMP EDI,EBP
03B47453 ^72 DB JB SHORT vsgdsf.03B47430
03B47455 8B4424 14 MOV EAX,DWORD PTR SS:[ESP+14]
03B47459 8B6E 04 MOV EBP,DWORD PTR DS:[ESI+4]
03B4745C 2B2E SUB EBP,DWORD PTR DS:[ESI]
03B4745E 5F POP EDI
03B4745F 2BE8 SUB EBP,EAX
03B47461 036E 08 ADD EBP,DWORD PTR DS:[ESI+8]
03B47464 5E POP ESI
03B47465 8BC5 MOV EAX,EBP
03B47467 5D POP EBP
03B47468 5B POP EBX
03B47469 C3 RETN
03B4746A 84C9 TEST CL,CL
03B4746C 74 23 JE SHORT vsgdsf.03B47491
03B4746E 8B87 28000000 MOV EAX,DWORD PTR DS:[EDI+8028]
03B47474 C3 RETN
```

Registers (FPU)

```
EAX 00000064
ECX 04903F7B ASCII "9We0We1We2We3We4We5We6We7We8We
EDX 04903CE9 ASCII "i0U1U2U3U4U5U6U7U8U9U
EBX 0216C5AC ASCII "Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1
ESP 0216C51C
EBP 03B4744B
ESI 000612A0
EDI 00003A54
EIP 03B4744B vsgdsf.03B4744B
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
D 0 DS 0023 32bit 0(FFFFFFFF)
Z 0 SS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

Address Hex dump ASCII

```
00CB4000 56 65 72 73 69 6F 6E 3A Version:
00CB4008 20 32 2E 50 35 30 00 44 61 2.50.Da
00CB4010 74 65 3A 20 41 75 67 20 te: Aug
00CB4018 20 38 20 32 30 30 37 00 8 2007.
00CB4020 00 00 00 00 68 61 73 70 ...hasp
00CB4028 66 6F 72 60 61 74 00 00 format..
00CB4030 66 6F 72 60 61 74 00 00 format..
00CB4038 73 65 73 73 69 6F 6E 69 session i
00CB4040 6E 66 6F 00 3C 68 61 73 nfo.<has
00CB4048 70 66 6F 72 60 61 74 20 pformat
00CB4050 66 6F 73 60 61 74 3D 22 format="
00CB4058 73 65 73 73 69 6F 6E 69 session i
00CB4060 6E 66 6F 22 2F 3E 00 00 nfo"/>..
00CB4068 75 70 64 61 74 65 69 6E update in
00CB4070 66 6F 00 00 3C 68 61 73 fo.<has
00CB4078 70 66 6F 72 60 61 74 20 pformat
00CB4080 66 6F 72 60 61 74 3D 22 format="
00CB4088 75 70 64 61 74 65 69 6E update in
00CB4090 66 6F 22 2F 3E 00 00 00 fo"/>...
00CB4098 66 61 73 74 75 70 64 61 fastupda
00CB40A0 74 65 69 6E 66 6F 00 00 teinfo..
00CB40A8 3C 68 61 73 70 66 6F 72 <haspfor
00CB40B0 6D 61 74 20 66 6F 72 6D mat form
00CB40B8 61 74 3D 22 66 61 73 74 at="fast
00CB40C0 75 70 64 61 74 65 69 6E update in
00CB40C8 66 6F 22 2F 3E 00 00 00 fo"/>...
```

0216CDE8 35734634 4Fs5 Pointer to next SEH rec
0216CDEC 46367346 Fs6F SE handler
0216CDF0 73463773 s7Fs
0216CDF4 39734638 8Fs9
0216CDF8 73463773 s7Fs
0216CDFC 74463174 t1Ft
0216CE00 33744632 2Ft3
0216CE04 46347446 Ft4F
0216CE08 74463574 t5Ft
0216CE0C 37744636 6Ft7
0216CE10 46387446 Ft8F
0216CE14 75463974 t9Fu
0216CE18 31754630 0Fu1
0216CE1C 46327546 Fu2F
0216CE20 75463975 u3Fu
0216CE24 35754634 4Fu5
0216CE28 46367546 Fu6F
0216CE2C 75463775 u7Fu
0216CE30 39754638 8Fu9
0216CE34 46307646 Fv0F
0216CE38 76463176 v1Fu
0216CE3C 33764632 2Fu3
0216CE40 46347646 Fu4F
0216CE44 76463576 v5Fu
0216CE48 37764636 6Fu7
0216CE4C 46387646 Fv8F
0216CE50 77463976 v9Fu COMCTL32.77463976
0216CE54 31734638 8Fs1

Trace into <Ctrl+F11> Paused

start 2 Windows ... 2 Windows ... EnCase Fore... Immunity De... 10:37 AM

The minimized-to-Metasploit crash

The screenshot displays the Immunity Debugger interface for the process EnCase.exe. The CPU window shows the main thread's execution state. The registers window shows the current register values, with EIP pointing to 46367346. The assembly window shows the following code:

Address	Hex dump	ASCII
00CB4000	56 65 72 73 69 6F 6E 3A	Version:
00CB4008	20 32 2E 35 30 00 44 61	2.50.Da
00CB4010	74 65 30 20 41 75 67 20	te: Aug
00CB4018	20 38 20 32 30 30 37 00	8 2007.
00CB4020	00 00 00 00 68 61 73 70	...hasp
00CB4028	66 6F 72 60 61 74 00 00	format..
00CB4030	66 6F 72 60 61 74 00 00	format..
00CB4038	73 65 73 73 69 6F 6E 69	sessioni
00CB4040	6E 66 6F 00 3C 68 61 73	nfo.<has
00CB4048	70 66 6F 72 60 61 74 20	pformat
00CB4050	66 6F 72 60 61 74 3D 22	format="
00CB4058	73 65 73 73 69 6F 6E 69	sessioni
00CB4060	6E 66 6F 22 2F 3E 00 00	nfo"/>..
00CB4068	75 70 64 61 74 65 69 6E	updatein
00CB4070	66 6F 00 00 3C 68 61 73	fo.<has
00CB4078	70 66 6F 72 60 61 74 20	pformat
00CB4080	66 6F 72 60 61 74 3D 22	format="
00CB4088	75 70 64 61 74 65 69 6E	updatein
00CB4090	66 6F 22 2F 3E 00 00 00	fo"/>..
00CB4098	66 61 73 73 74 75 70 64	fastupda
00CB40A0	74 65 69 6E 66 6F 00 00	teinfo..
00CB40A8	3C 68 61 73 70 66 6F 72	<haspfor
00CB40B0	6D 61 74 20 66 6F 72 6D	mat form
00CB40B8	61 74 3D 22 66 61 73 74	at="fast
00CB40C0	75 70 64 61 74 65 69 6E	updatein
00CB40C8	66 6F 22 2F 3E 00 00 00	fo"/>..

The assembly window shows the following instructions:

```
0216C140 7C903298 22E1 RETURN to ntdll.7C903298
0216C150 0216C234 4T.0
0216C154 0216CDE8 3=-.0 ASCII "4Fs5Fs6Fs7Fs8Fs9"
0216C158 0216C250 PT.0
0216C15C 0216C208 4T.0
0216C160 0216CDE8 3=-.0 Pointer to next SEH rec
0216C164 7C9032BC 42E1 SE handler
0216C168 0216CDE8 3=-.0 ASCII "4Fs5Fs6Fs7Fs8Fs9"
0216C16C 0216C21C LT.0
0216C170 7C90327A 22E1 RETURN to ntdll.7C90327A
0216C174 0216C234 4T.0
0216C178 0216CDE8 3=-.0 ASCII "4Fs5Fs6Fs7Fs8Fs9"
0216C17C 0216C250 PT.0
0216C180 0216C208 4T.0
0216C184 46367346 Fs6F
0216C188 00003A54 T..
0216C18C 0216C234 4T.0
0216C190 0216CDE8 3=-.0 ASCII "4Fs5Fs6Fs7Fs8Fs9"
0216C194 7C92A8C3 12E1 RETURN to ntdll.7C92A8C3
0216C198 0216C234 4T.0
0216C19C 0216CDE8 3=-.0 ASCII "4Fs5Fs6Fs7Fs8Fs9"
0216C1A0 0216C250 PT.0
0216C1A4 0216C208 4T.0
0216C1A8 46367346 Fs6F
0216C1AC 00003A54 T..
0216C1B0 0216C234 4T.0
0216C1B4 000612A0 3+=.0
0216C1B8 00003A54 T..
```

The status bar shows the error message: "Access violation when executing [46367346] - use Shift+F7/F8/F9 to pa". The debugger is in a "Paused" state.



Fuzzing on Windows: The CERT FOE

Enter the FOE

Failure Observation Engine (FOE)

<https://www.cert.org/vuls/discovery/foe.html>

- Windows-compatible
- Functional decomposition of BFF (and zzuf)
- Python (and a bit of C)
- Easy to use
 1. Pick seed files to mutate
 2. Enter target app command line
 3. Go!

Exception Detection

Debuggers

- Slow (sometimes)
- Foiled by anti-RE tricks
- Heisenbugs

Exception handler hooks

- Fast
- Less likely for anti-RE to detect
- Not very informative (yet)

Exception Detection - Hook

KiUserExceptionDispatcher()

- Called in userland before process exception handling

Installation

1. Use *Applnit_DLLs* registry value to load hook DLL
2. Overwrite first few instructions to jmp to our trampoline code

Trampoline: Do we care about the exception?

- Yes: Kill the process (group) with the exception code
- No: Pass exception to target application

Uniqueness Determination

MS !exploitable debugger extension

- <http://msecdbg.codeplex.com/>

!exploitable hash

- Based on current instruction pointer, other state
- Form: Major.Minor
- E.g. 0x2472222b.0x134c461c

Cannot unique true heisenbugs

Exploitability

!exploitable “Exploitability Classification”

- Based on exception type and properties
 - Read A/V on eax near NULL = PROBABLY_NOT_EXPLOITABLE
 - Write A/V not near NULL = PROBABLY_EXPLOITABLE
 - Read A/V on instruction pointer not near NULL = EXPLOITABLE
- Assumes all inputs to faulting instruction are attacker controlled (tainted)
- Errs on false positive side

Interesting crashes

Problem: Even with !exploitable crash categorization, you may have too many results to sift through.

Solution: drillresults.py

- Select interesting exceptions
- Look for byte patterns that match fuzzed file
- Rank interesting crashes

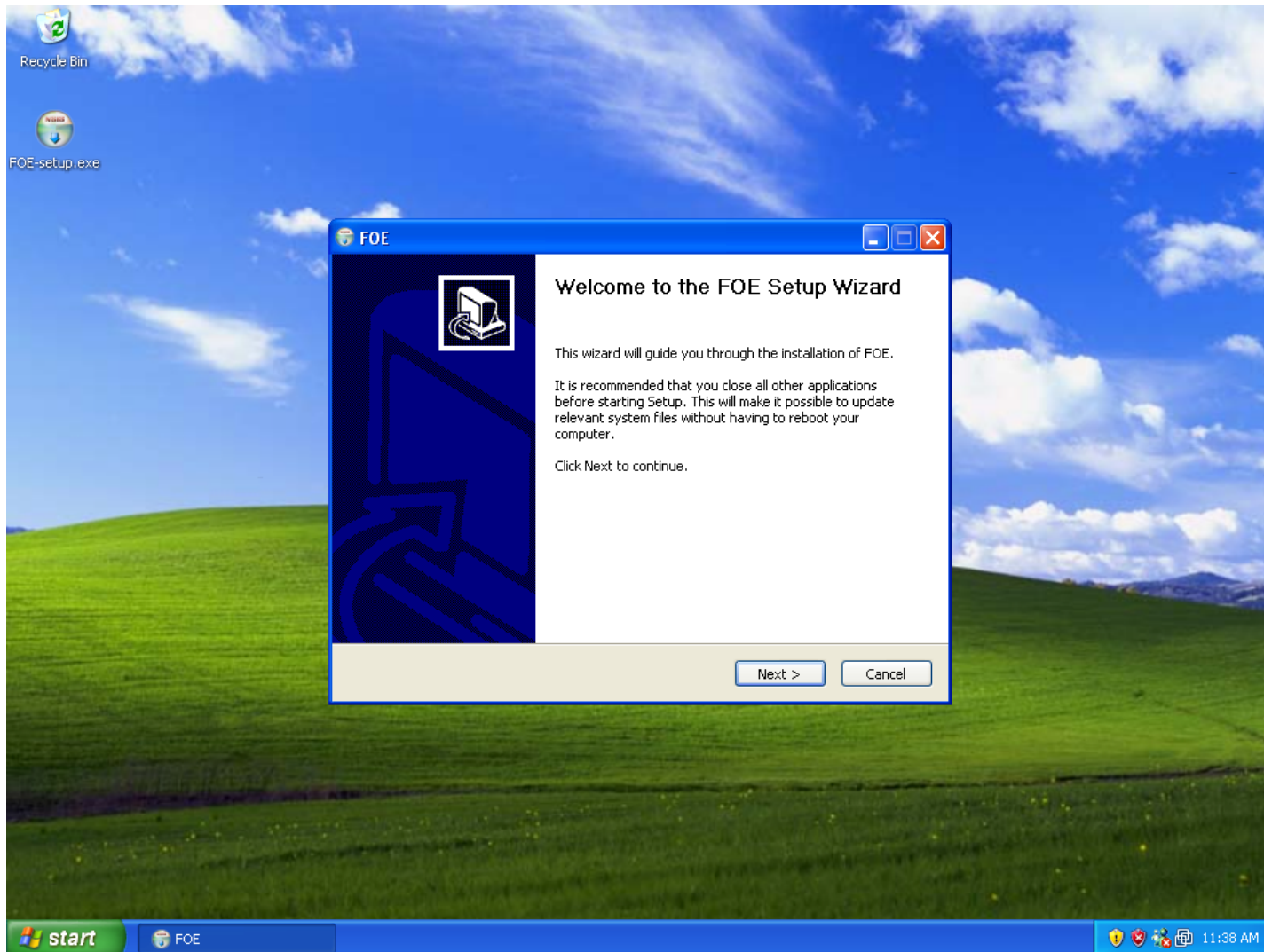
drillresults.py output

```
0x1c636361.0x1a2f7629 - Exploitability rank: 10
Fuzzed file: results\oi-multi-2\PROBABLY_EXPLOITABLE
\0x1c636361.0x1a2f7629\sف_7fd23297537035d4d1ed899c4838d862.lwp
exception 0: TaintedDataControlsCodeFlow accessing 0x00080800 *** Byte pattern is in fuzzed file!
***
1034ea66 8b01 mov eax,dword ptr [ecx] ds:0023:00080800=???????? Code executing in: C:\1-ix\redist
\lwpapin.dll
```

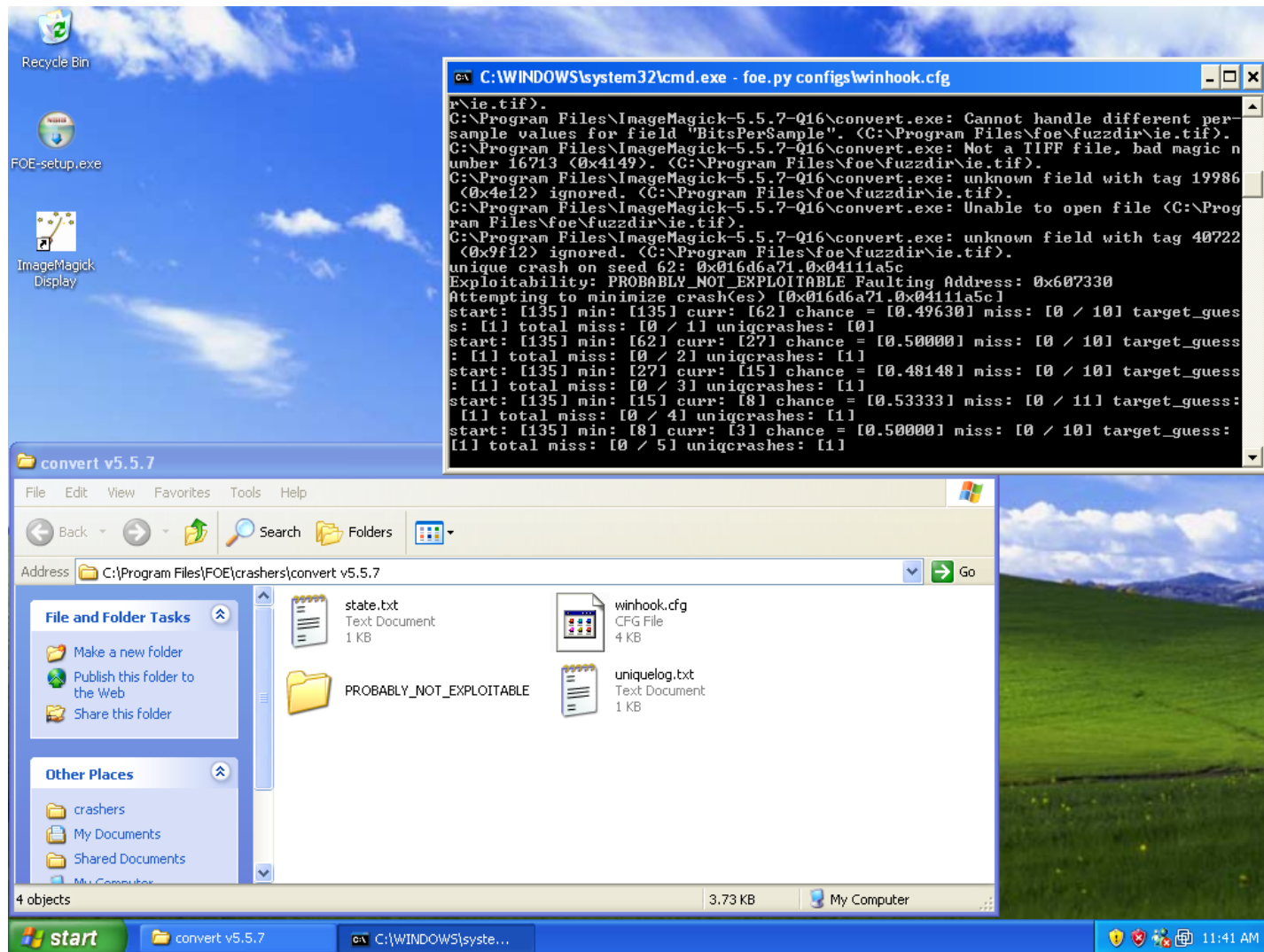
```
0x607f0d37.0x510f346f - Exploitability rank: 20
Fuzzed file: results\oi-multi-2\EXPLOITABLE\0x607f0d37.0x510f346f
\sف_1903537138d91f0dadd9511d3b7522ed.cdr
exception 0: WriteAV accessing 0x00130000 *** Byte pattern is in fuzzed file! ***
00c97be7 880417 mov byte ptr [edi+edx],al ds:0023:00130000=41
Code executing in: C:\1-ix\redist\vsgdsf.dll
exception 1: ReadAVonIP accessing 0x00003ff0 *** Byte pattern is in fuzzed file! ***
00003ff0 ?? ???
Instruction pointer is not in a loaded module!
```

```
0x0b535856.0x02751235 - Exploitability rank: 30
Fuzzed file: results\oi 8.3.7.77-noefa\PROBABLY_EXPLOITABLE
\0x0b535856.0x02751235\sف_4a4baf4f7167552d1144a9fefa29f9bf-69152-0x00000000.sxd
exception 0: TaintedDataControlsCodeFlow accessing 0x00000000 *** Byte pattern is in fuzzed file!
***
0140c574 8b11 mov edx,dword ptr [ecx] ds:0023:00000000=????????
Code executing in: C:\1-ix\redist\DEVECT.DLL
```

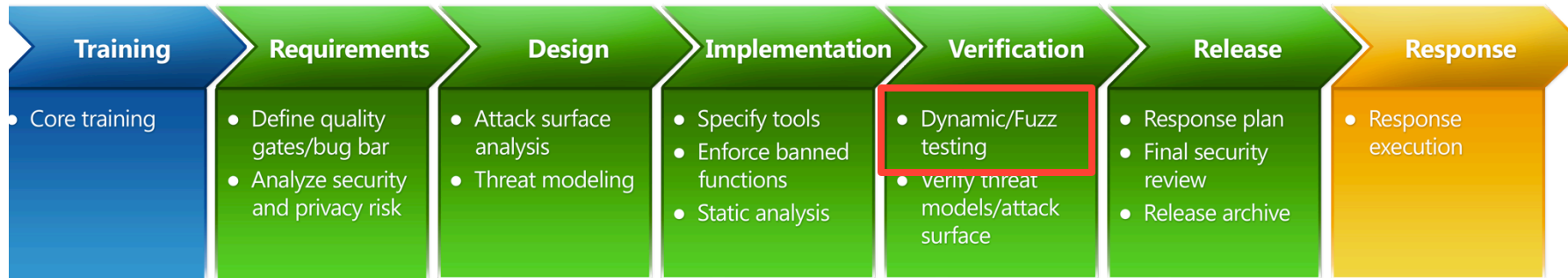
The CERT® FOE



The CERT® FOE

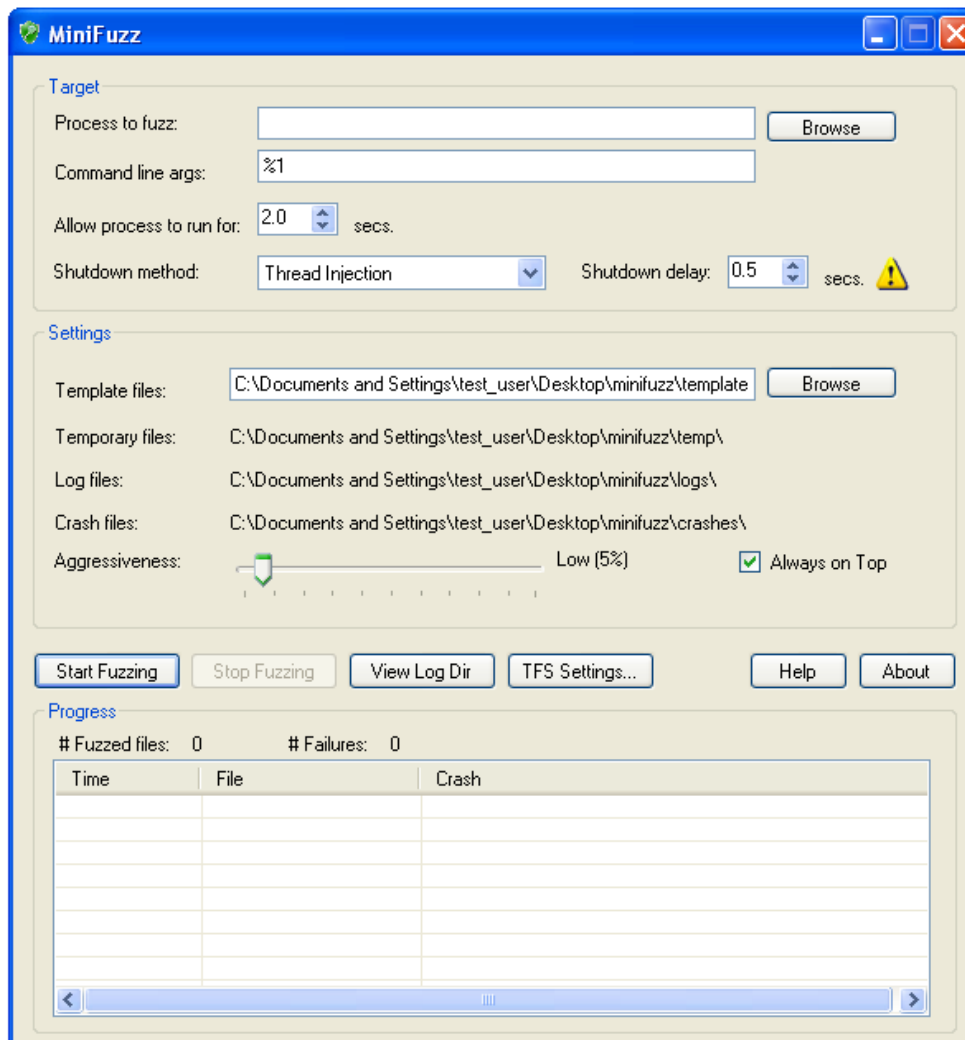


Microsoft SDL



The Microsoft SDL recommends Fuzz testing.

Microsoft MiniFuzz



MiniFuzz vs. FOE

~1 day of Fuzzing Oracle Outside In

	Unique Crashes	Seconds until first crash
MiniFuzz	1	74520
FOE	59	60
FOE 2.0	99	3



A Real-world FOE Example

The Target

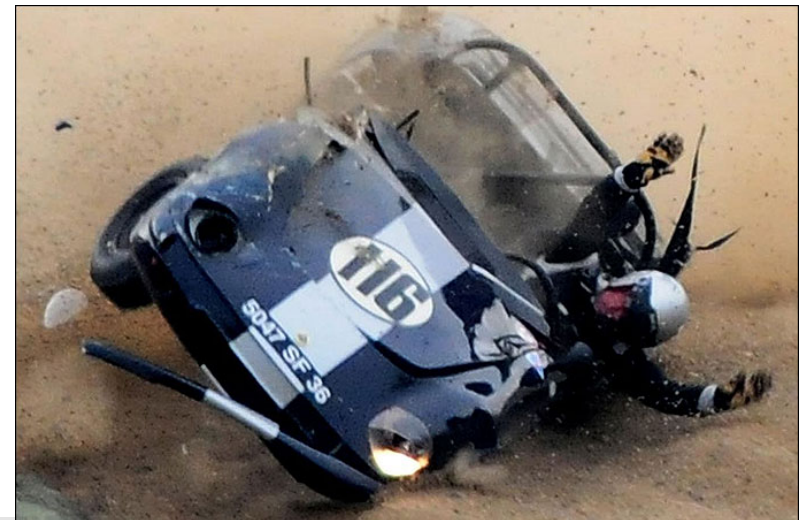
Oracle Outside in

- Decodes over 500 different file types
 - Large attack surface
- Used by a variety of applications
 - Oracle Fusion Middleware
 - Novell Groupwise
 - Microsoft Exchange
 - Guidance Encase Forensics
 - AccessData FTK
 - Paraben Device Seizure

Fuzzing results

Unique crashes found through 30 hours of fuzzing with FOE:

- 24 EXPLOITABLE
 - 40 PROBABLY_EXPLOITABLE
 - 67 UNKNOWN
 - 10 PROBABLY_NOT_EXPLOITABLE
-
- 141 Total unique crashes

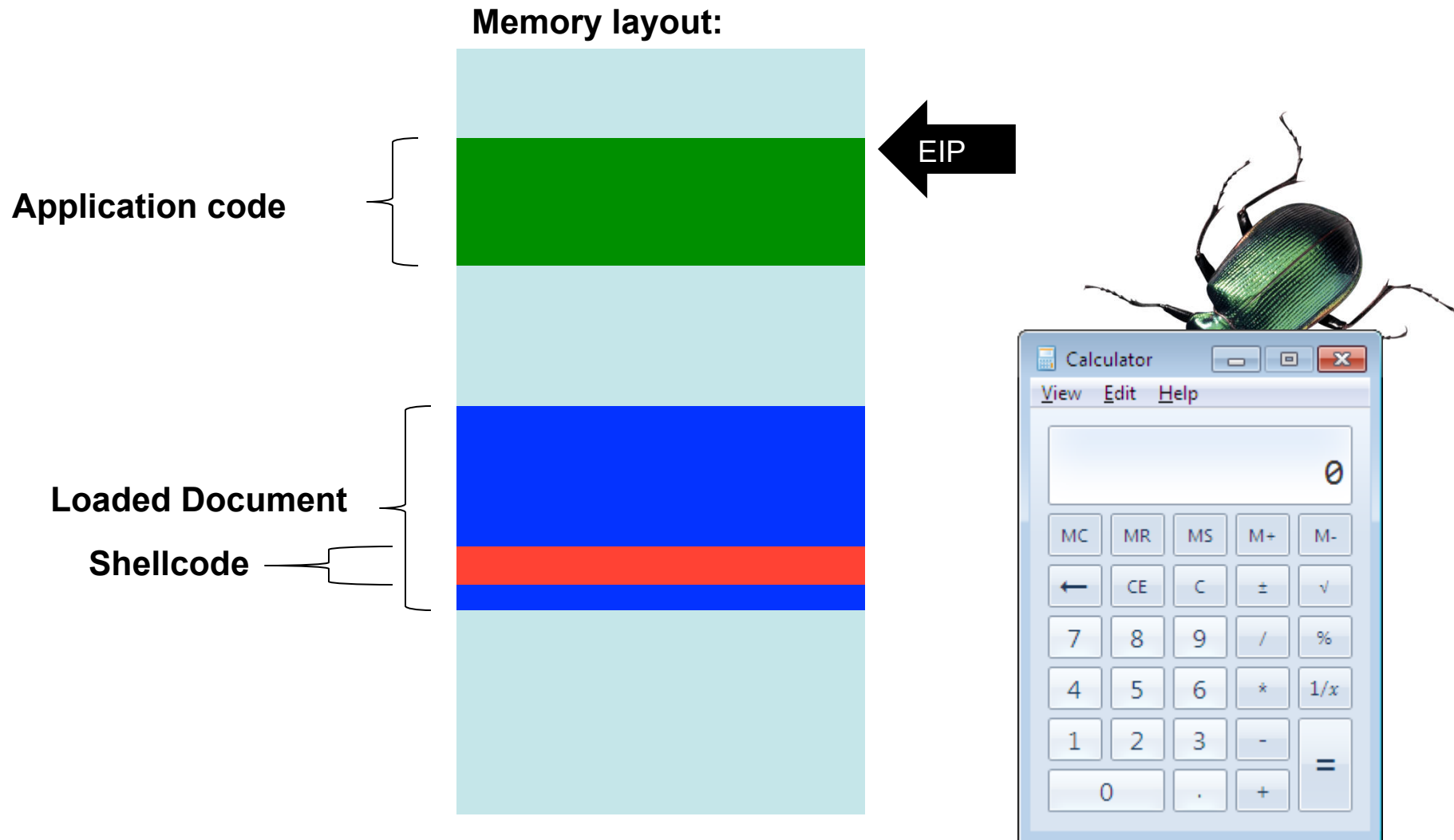


Exploiting vulnerabilities

Get control of Instruction Pointer (EIP)

- Control of EIP == Control of execution
- Point EIP to attacker's code (shellcode) : attacker's code executes

Exploiting vulnerabilities



An interesting bug

Eight hours into the fuzzing run, in the Lotus 123 v.6 file parser (vswk6.dll):

Exception Faulting Address: 0x284c584e

First Chance Exception Type: STATUS_ACCESS_VIOLATION (0xC0000005)

Exception Sub-Type: Read Access Violation

Description: Read Access Violation at the Instruction Pointer

Short Description: ReadAVonIP

Exploitability Classification: EXPLOITABLE

Proof-of-Concept Exploit





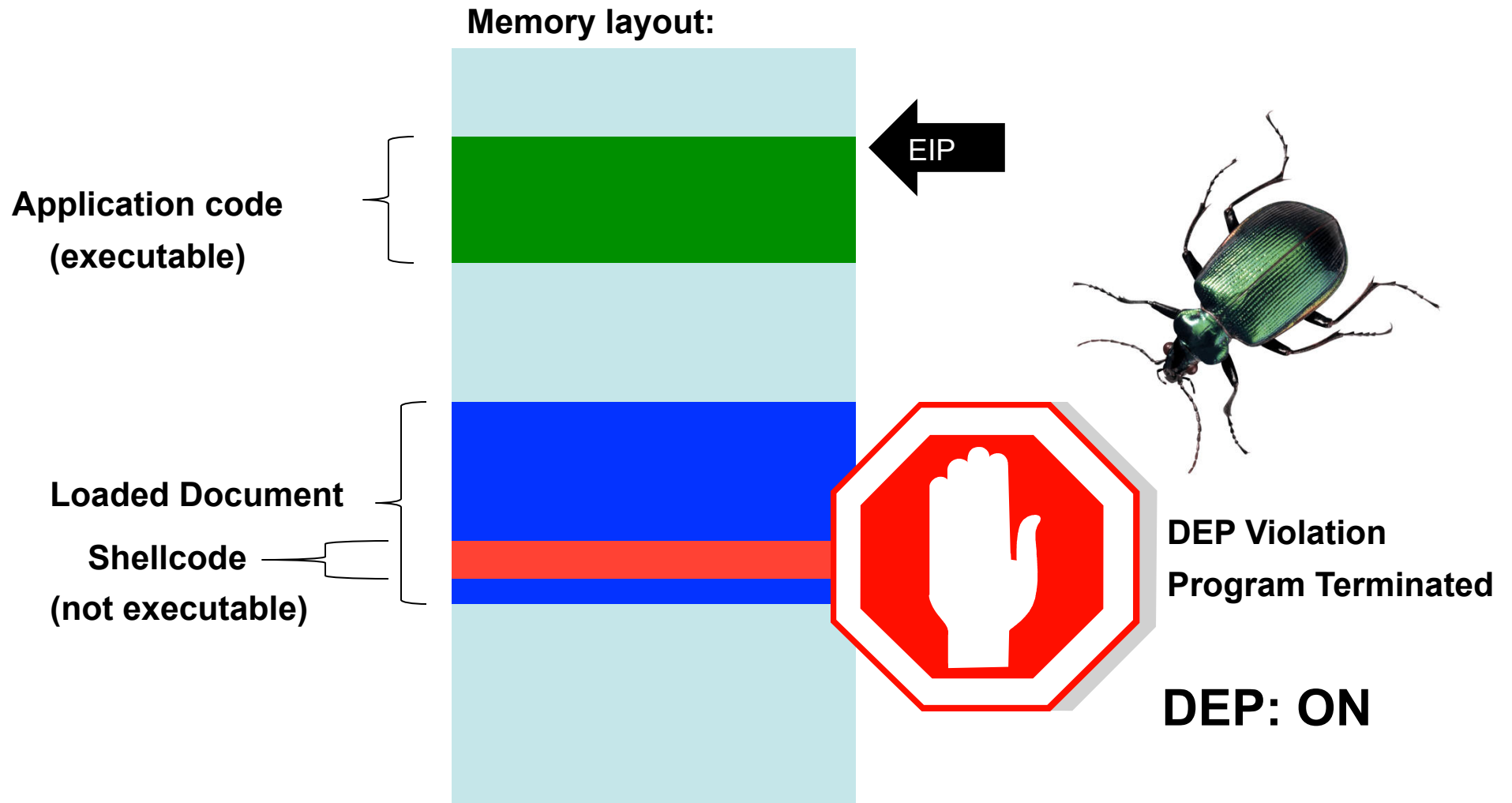
Exploitation Protections

Protection #1: DEP

Data Execution Prevention

- Do not execute memory locations that do not have execute permissions
- Requires processor support: NX bit
- Applications must opt-in

DEP Protection



Time to go home!

DEP solves the problem, right?



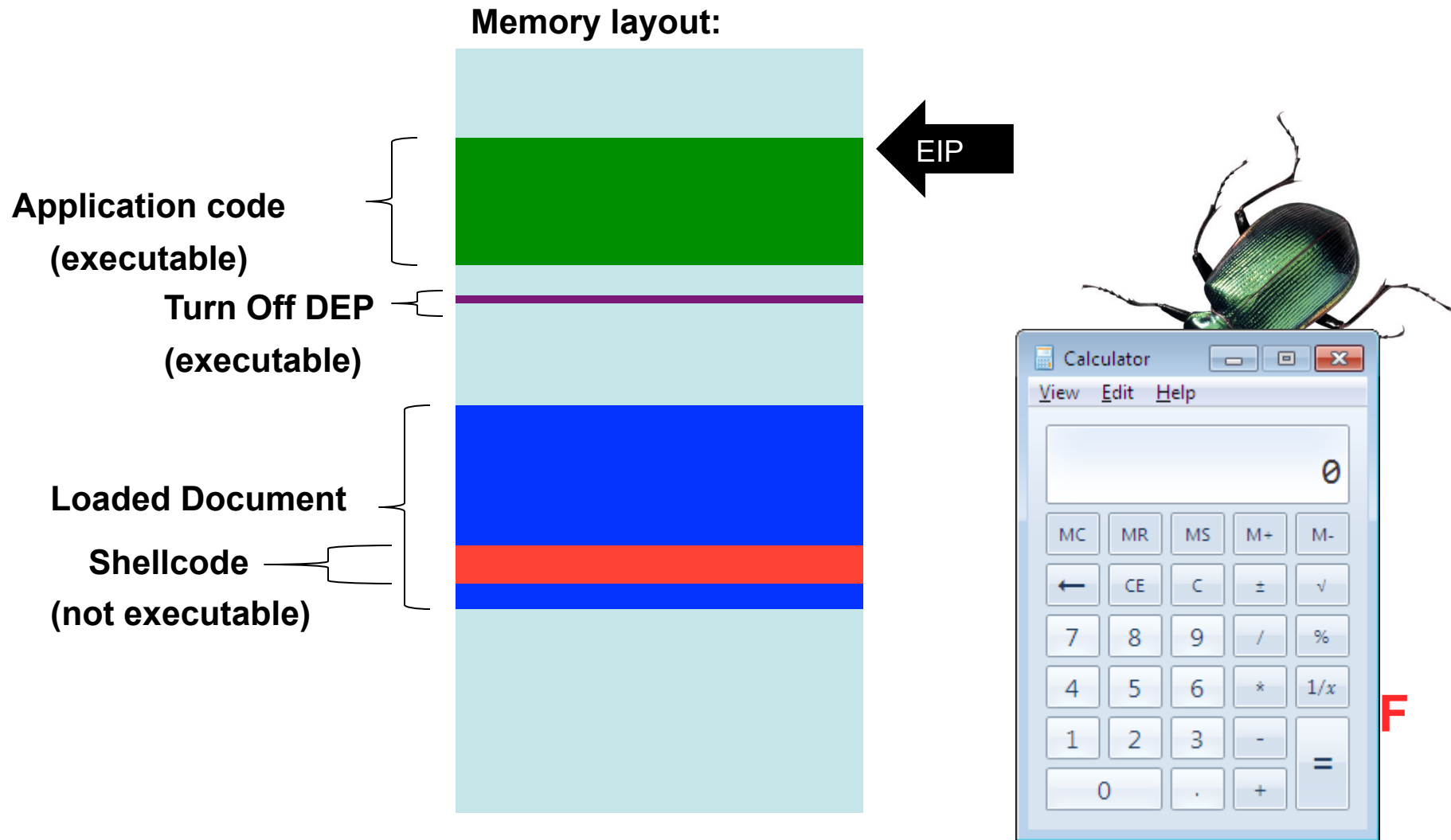
Return Oriented Programming

Use pieces of existing executable code to accomplish your goal of bypassing DEP. Several techniques can be used, including:

- Turn off DEP
- Mark memory as executable
- Allocate new executable memory
- Copy shellcode to executable memory

Outcome: Executable shellcode

Exploiting vulnerabilities



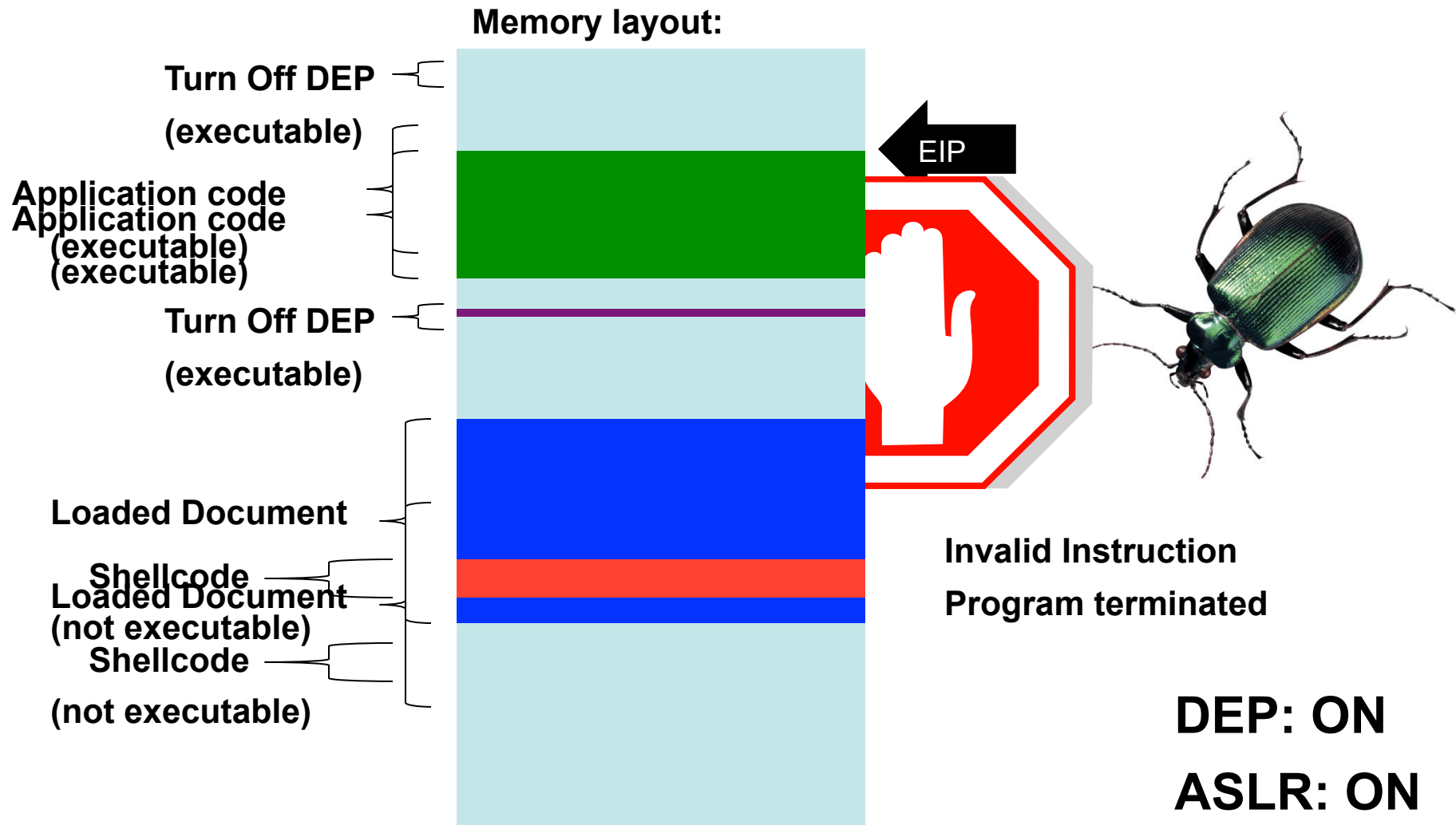
Protection #2: ASLR

Address Space Layout Randomization

- Executable modules loaded at randomized location
- Breaks ROP



Exploiting vulnerabilities



Exploit Mitigation

DEP and full ASLR together help prevent exploitation of vulnerabilities.

- DEP without ASLR is not effective
 - Vista or later is required for ASLR
- ASLR without DEP is not effective
- Every loaded module needs to opt in to ASLR



Exploit Mitigation Report Card

Default software installation

	DEP		ASLR		Exploit Mitigation?
Encase 6	No	+	No	=	No
Encase 7	No*	+	No	=	No
FTK 3.3	Yes	+	No	=	No
FTK 3.4	Yes	+	No	=	No
Device Seizure	No	+	No	=	No

* DEP Enabled on Vista or later

Everybody Fails



Vulnerability Exploit protection

What do we know about vulnerability protection?

- Vendors don't always opt in to exploit mitigations
- Vendors don't fix known vulnerabilities in a timely manner
- We want protection from unknown vulnerabilities

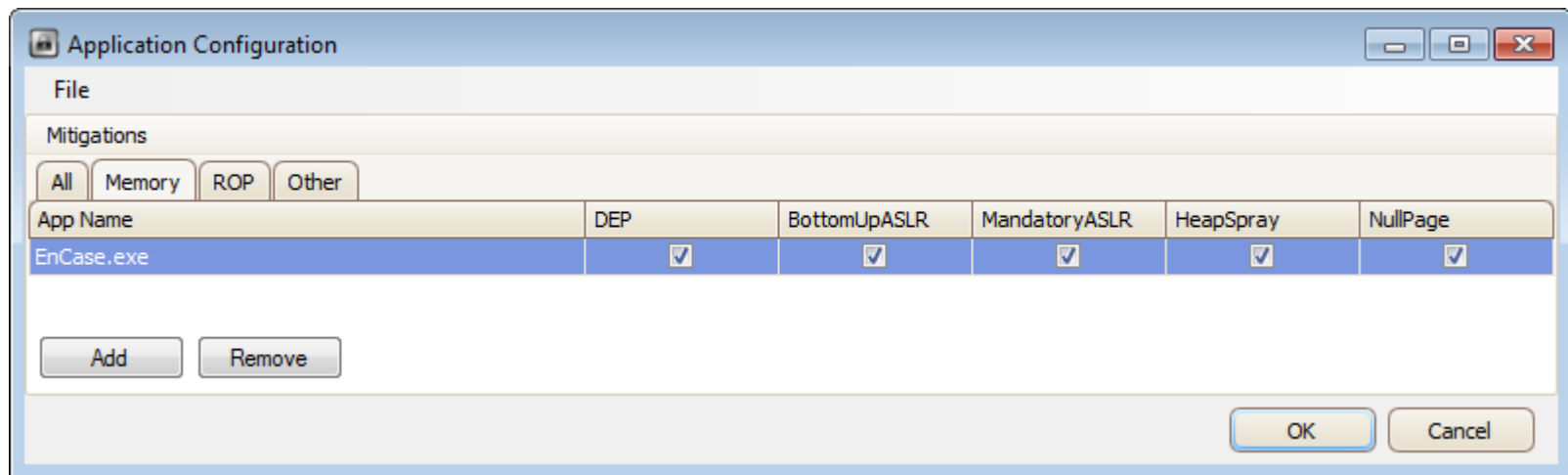
Microsoft EMET

Don't be at the mercy of your software vendors. Microsoft Enhanced Mitigation Experience Toolkit can force-enable:

- DEP
- ASLR (Vista and newer)
- SEHOP
- Additional exploit mitigations

<http://support.microsoft.com/kb/2458544>

Microsoft EMET



Exploit Mitigation Report Card

Configured with EMET

	DEP		ASLR*		Exploit Mitigation?
Encase 6	Yes	+	Yes	=	Yes
Encase 7	Yes	+	Yes	=	Yes
FTK 3.3	Yes	+	Yes	=	Yes
FTK 3.4	Yes	+	Yes	=	Yes
Device Seizure	Yes	+	Yes	=	Yes

* ASLR Enabled on Vista or later

Everyone's a winner!



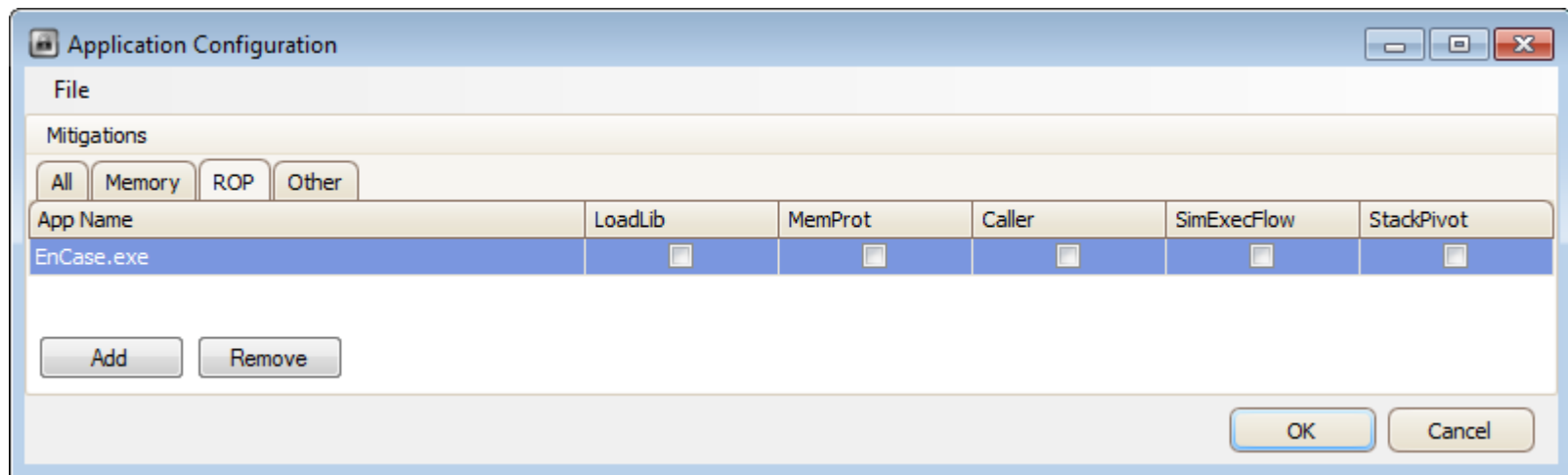
ASLR Requires Vista or Newer

Windows XP (Server 2003) does not
support ASLR!



ROP Mitigations

EMET 3.5 introduces explicit ROP mitigations



EMET Without ROP Mitigations



EMET With ROP Mitigations



Use EMET to stay safe

The way to more safely run applications on Windows is to use EMET!

- Minimize risk of delayed patching
- Protect against known vulnerabilities
- Protect against 0day vulnerabilities
- Protect against future vulnerabilities
- EMET 3.5 ROP protection buys time for migration off of Windows XP



Lessons Learned and Future Plans

BFF ~~Victims~~ Successes

Crashes with evidence of exploitability:

- Apple Mac OSX
- Adobe Reader
- Adobe Flash
- Foxit Reader
- Xpdf / Evince / Poppler
- ImageMagick
- JasPer
- Clamav
- Swfdump
- File
- Microsoft / Intel Indeo codec
- VMware vmnc codec
- Apple QuickTime
- Apple Preview
- Microsoft Office
- OpenOffice
- openjpeg
- ffmpeg (mplayer, VLC, ffdshow, etc.)

FOE ~~Victims~~ Successes

Crashes with evidence of exploitability:

- Adobe Reader
- Adobe Flash
- Adobe Shockwave
- Foxit Reader
- SumatraPDF
- LibreOffice
- Icenix Argus
- Microsoft Paint
- Microsoft Picture and Fax Viewer
- Microsoft Office
- Microsoft Windows
- Oracle OpenOffice
- Oracle Outside In
- Autonomy Keyview
- RealNetworks RealPlayer
- Winamp
- Java
- ffdshow
- Google Chrome

Lessons Learned

Throughput is king

- Minimize I/O
- CPU-bound
- Increase code coverage

Techniques:

- Web browser JavaScript that closes browser
- Print to Null printer
- Output to /dev/null
- Export / convert file

Lessons Learned

Everything is broken

- Dumb fuzzing shouldn't be so effective

Defense in depth:

- Runtime mitigations
- Compile-time mitigations
- Continuous fuzzing

Fuzzing Obstacles

GUI applications

- When is it “done” ?

Crashes vs. vuls

- More crashes

Can we handle all of the output?

Future Plans

Planned improvements for the BFF and related projects:

- Code coverage awareness
- Distributed fuzzing
- Improved crash triage and exploitability
- Multiple mutation strategies
- Brute-force determination of bytes that affect the faulting address
- Optimized pattern for cycling through bytes (inverse Gray code)

For More Information

Visit CERT® web sites:

<http://www.cert.org/vuls/discovery/>

<http://www.cert.org/blogs/certcc/>

<https://www.cert.org/vuls/discovery/bff.html>

<https://www.cert.org/vuls/discovery/foe.html>



Contact Presenter

Will Dormann

wd@cert.org

(412) 268-8922

Contact CERT:

Software Engineering Institute

Carnegie Mellon University

4500 Fifth Avenue

Pittsburgh PA 15213-3890

