Fuzz Testing: Vulnerabilities and Exploit mitigation

Will Dormann [wd@cert.org]

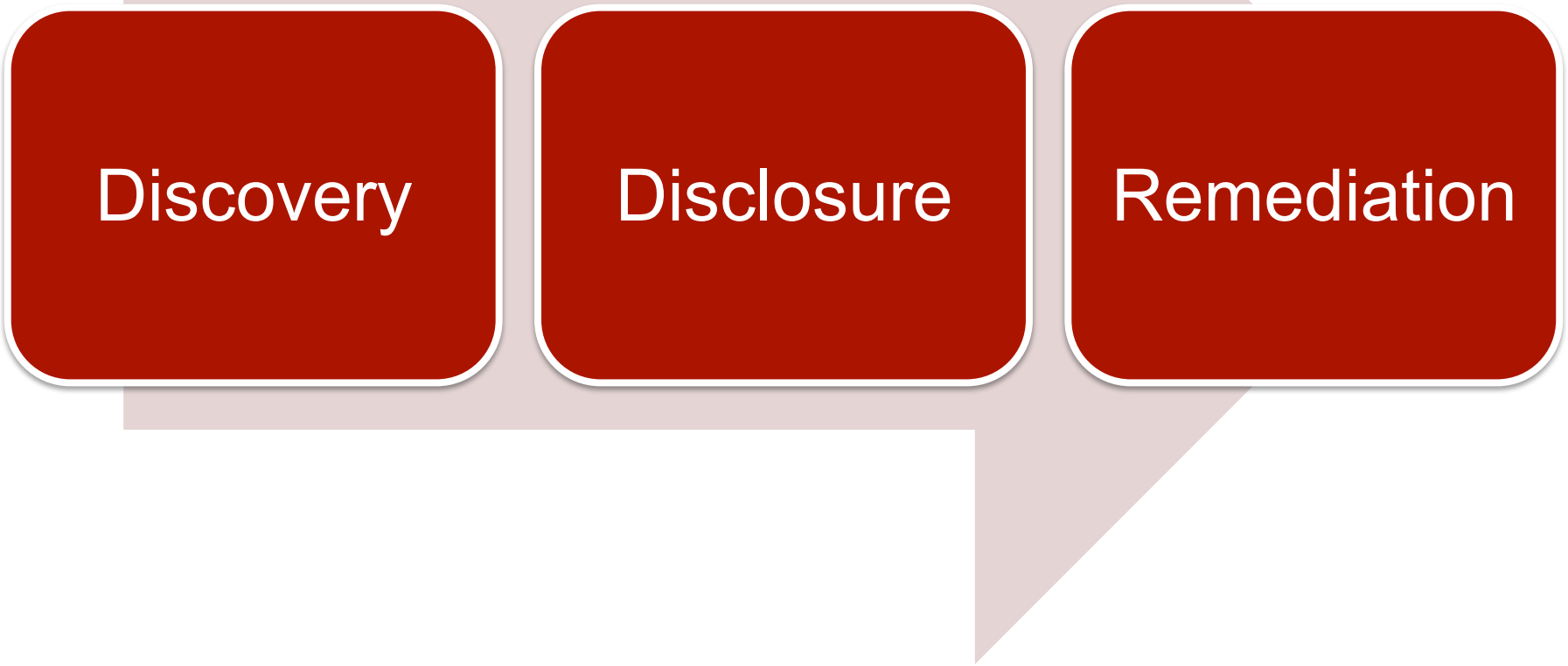
Outline

- 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



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

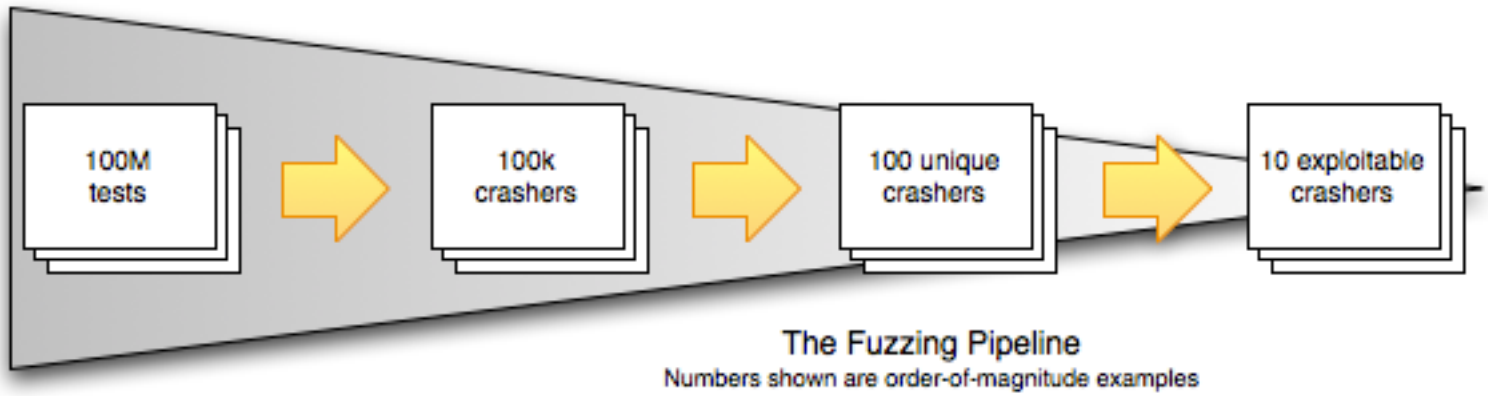
- Incurs high up-front costs to model input/protocol
- Easy to omit large branches of test cases

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



The Fuzzing Pipeline
Numbers shown are order-of-magnitude examples

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

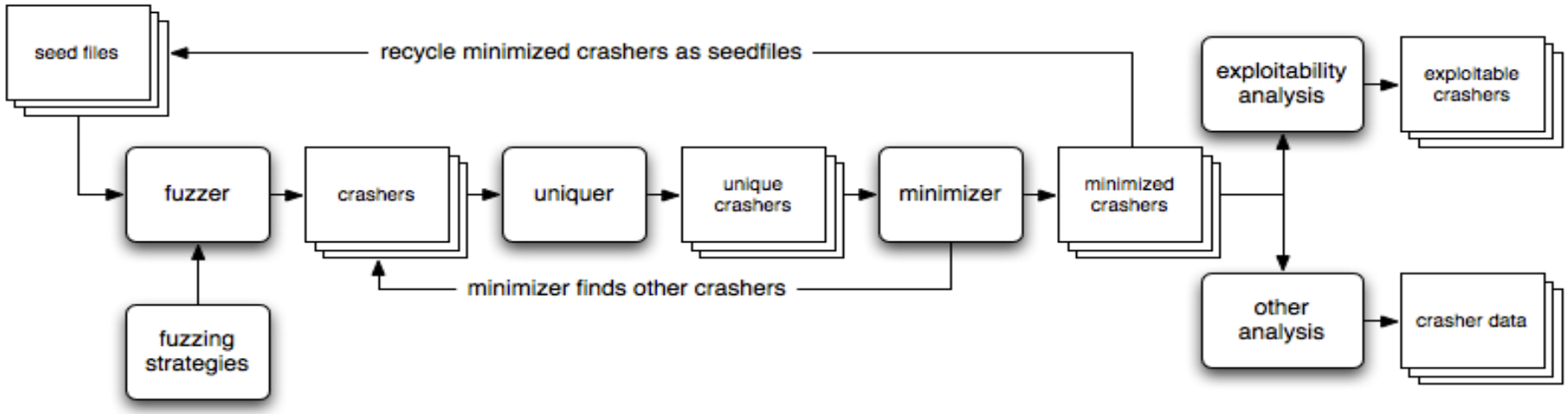
- Testcase generation
- Process killer
- Crash verification
- Crash deduplication
- Crash minimization

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”

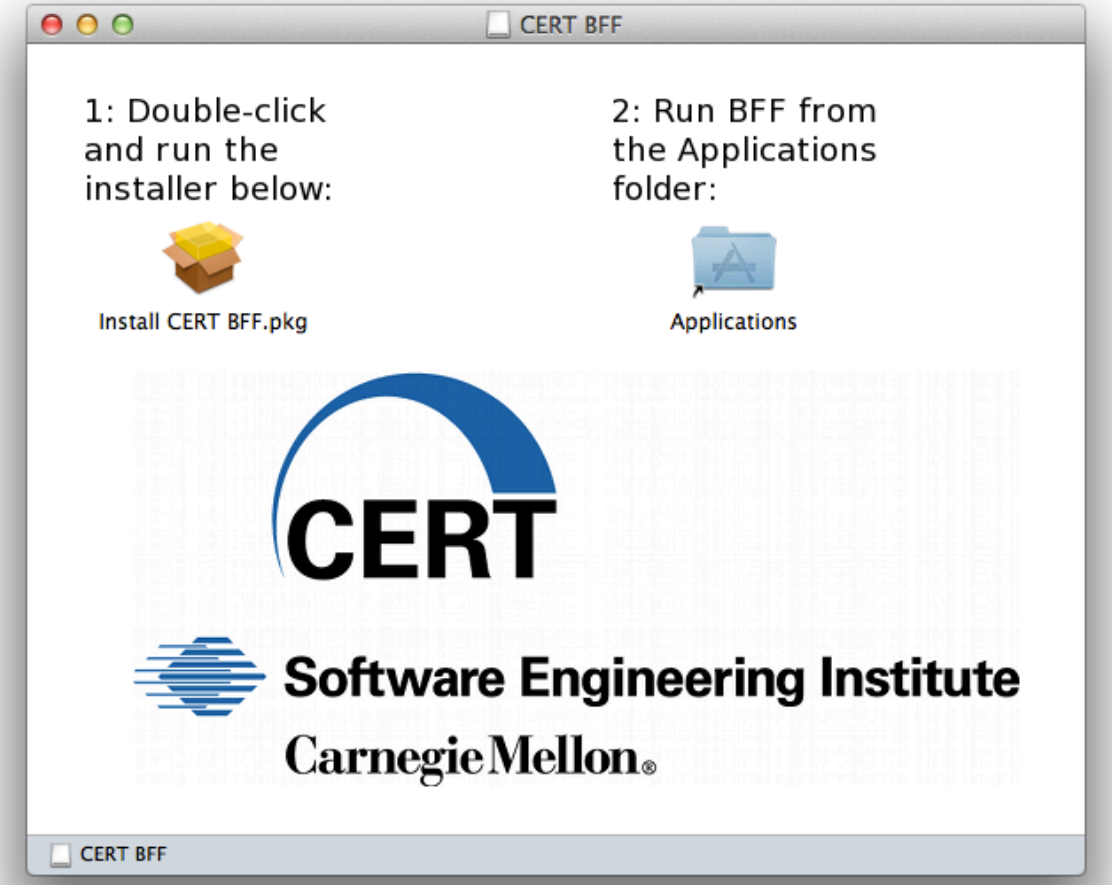


BFF Requirements

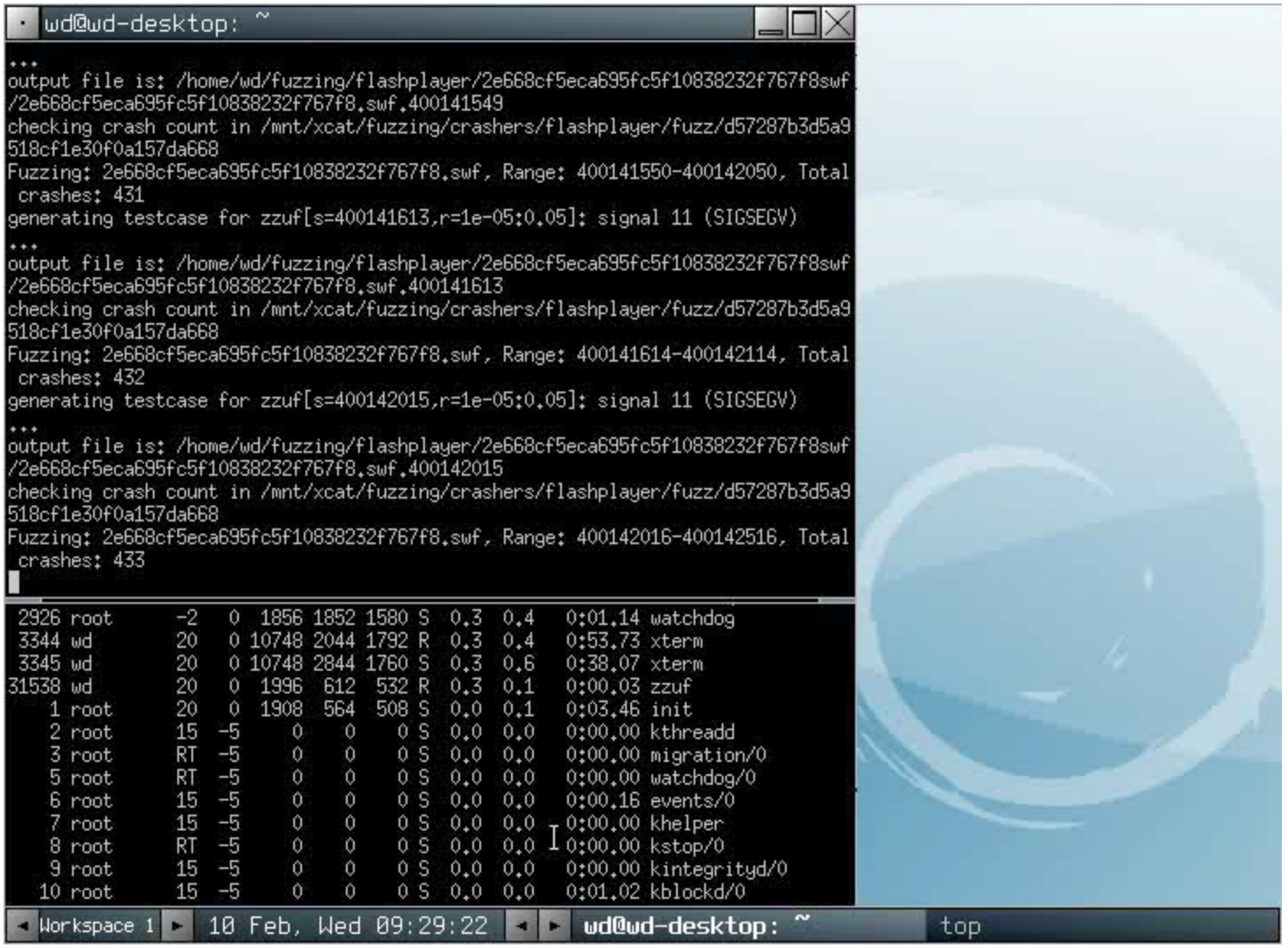
Prerequisites:

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

BFF on OS X


CERT BFF
1: Double-click and run the installer below:
Install CERT BFF.pkg
2: Run BFF from the Applications folder:
Applications
CERT
Software Engineering Institute
Carnegie Mellon
CERT BFF

Flash Fuzzing VM



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

Paper to be submitted to ISSTA-2012

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

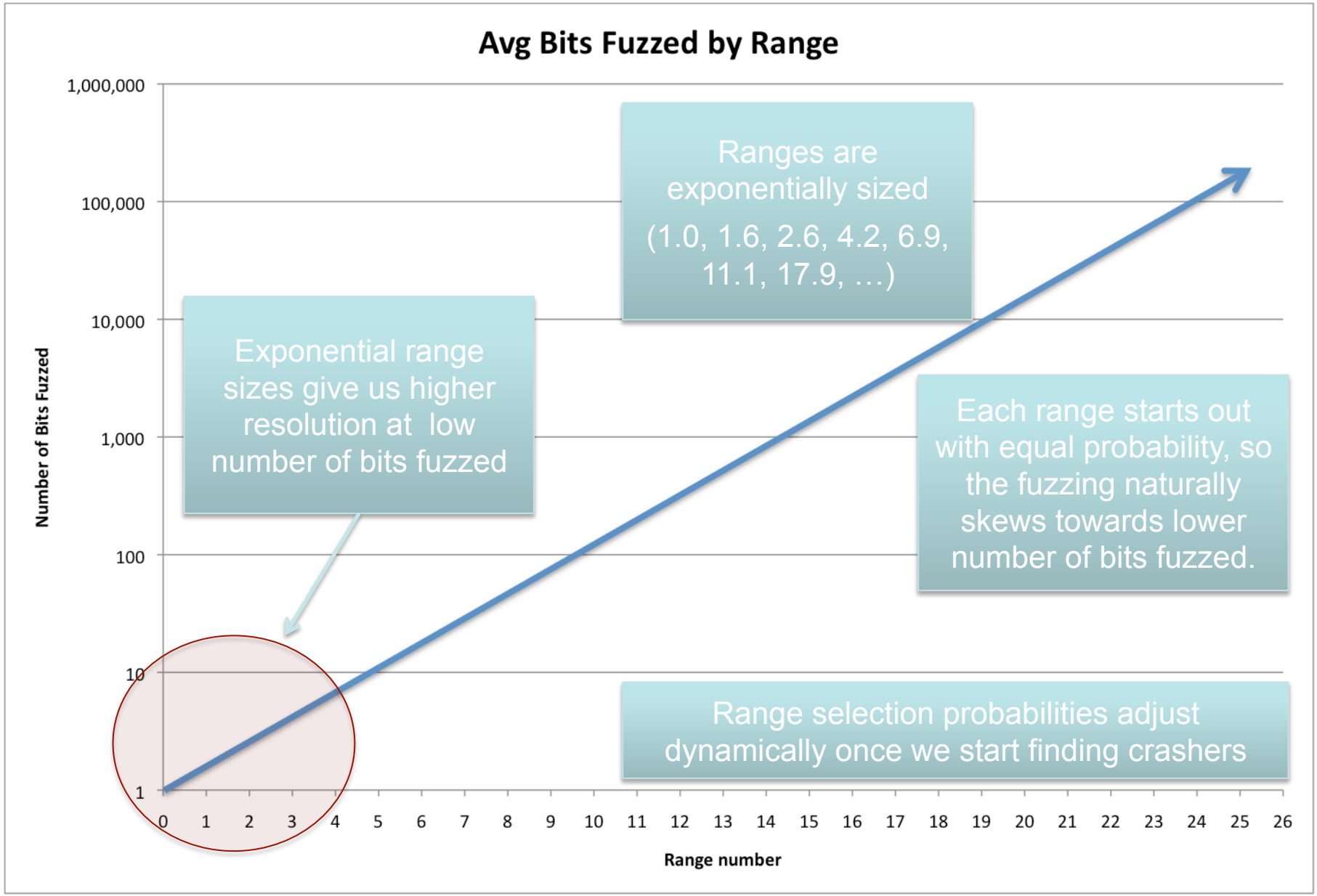
Segment proportion of file to be fuzzed into ranges

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

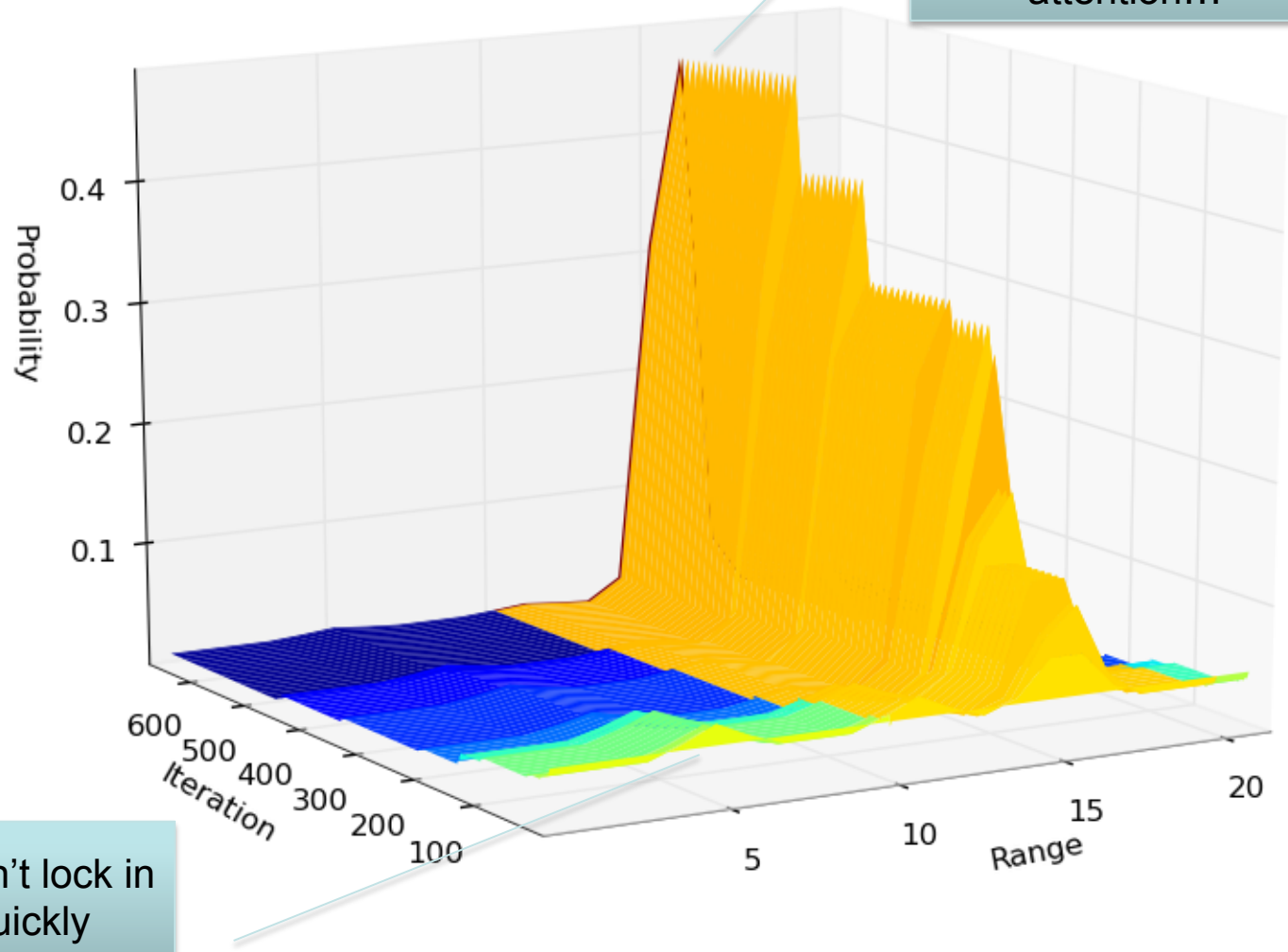
Prefer higher granularity at lower proportion of mangled bits

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
Number of Bits Fuzzed
1,000,000
100,000
10,000
1,000
100
10
1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Range number
Ranges are exponentially sized
(1.0, 1.6, 2.6, 4.2, 6.9, 11.1, 17.9, …)
Exponential range sizes give us higher resolution at low number of bits fuzzed
Each range starts out with equal probability, so the fuzzing naturally skews towards lower number of bits fuzzed.
Range selection probabilities adjust dynamically once we start finding crashers

Successful ranges should get more attention…



…but don't lock in too quickly

Problem: Volume of crashing test cases

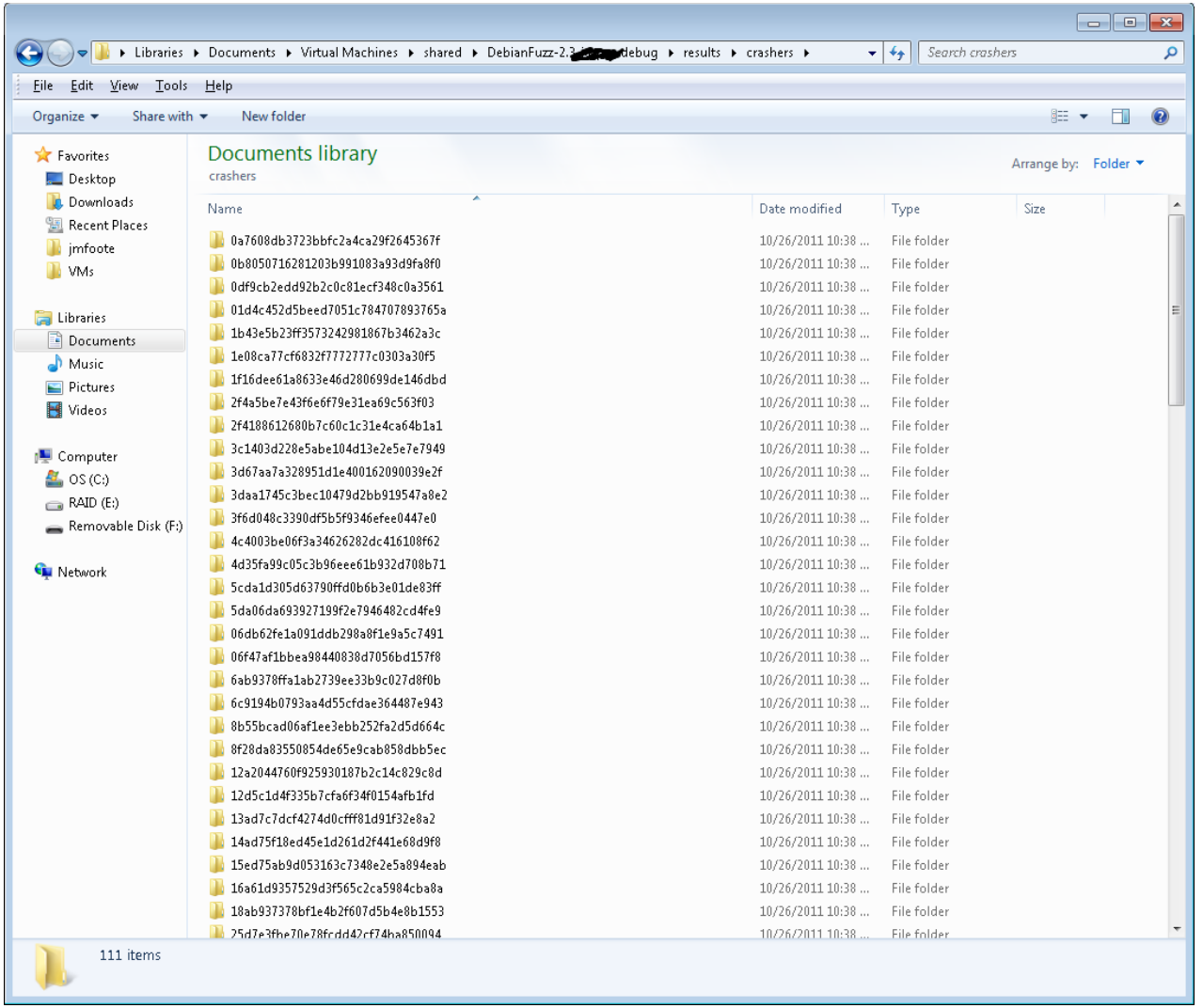
File fuzzing can yield a large number of crashing test cases

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 direct
ory.
        in ../sysdeps/i386/i686/memcpy.S
(gdb) source exploitable.py
(gdb) exploitable
Description: Possible stack corruption
Short description: PossibleStackCorruption (6/20)
Hash: dc64d713b1eb2f213638e3aa329f27fa.dc64d713b1eb2f213638e3
aa329f27fa
Exploitability Classification: EXPLOITABLE
Explanation: GDB generated an error while unwinding the stack
 and/or the stack contained return addresses that were not ma
pped 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, wh
ich 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/3f6d048c3390df5b5f9346efee0447e0/sf_19d59a74213e2410ce6b9f86b1b57e46-26302406.j2k
/mnt/hgfs/fuzz/results/crashers/55e22c6a6e2193a99f1739a843d6201c/sf_9db558c6149b7f52e24030fc177138e5-8279699.j2k
/mnt/hgfs/fuzz/results/crashers/6933ea7f9529662eeeec8c90514ae7a5/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/bfff89809721cdd64f6e176d978c885a/sf_422b371ea60d1f8766e86f914fda134f-742911.jp2

EXPLOITABLE: HeapError
/mnt/hgfs/fuzz/results/crashers/12a2044760f925930187b2c14c829c8d/sf_65cc567685f54ddf23e7d13e3b034d30-5141.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/13857d2feef6227ee8cdb43c01be59ce/sf_76905a851f5ee1fc4ad828431bcce5b5-27561820.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/16a61d9357529d3f565c2ca5984cba8a/sf_3fba9442b5afd7329e708852c52d3451-768216.jp2 (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/1f16dee61a8633e46d280699de146dbd/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/629dfc07884eaf43f2c44bc46aa66842/sf_4d55b040cfc6748007ee53a6c0e4528e-12087.j2k (AbortSignal)
/mnt/hgfs/fuzz/results/crashers/65841a85bd9dd36c1df59c997ff38040/sf_7a09938fbe57cef8ca2f7bc7cc9da978-1954601.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)

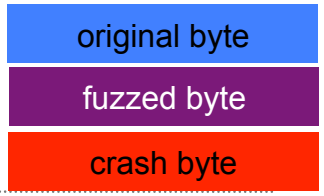
Steps to a solution

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

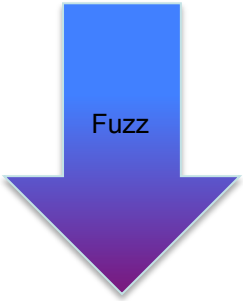
Test to see if we still see the same crash

Iterate and update strategy based on what we learn

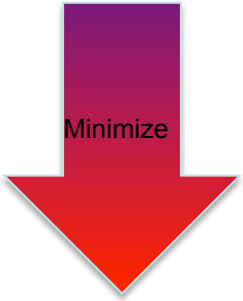
What Minimizer Does



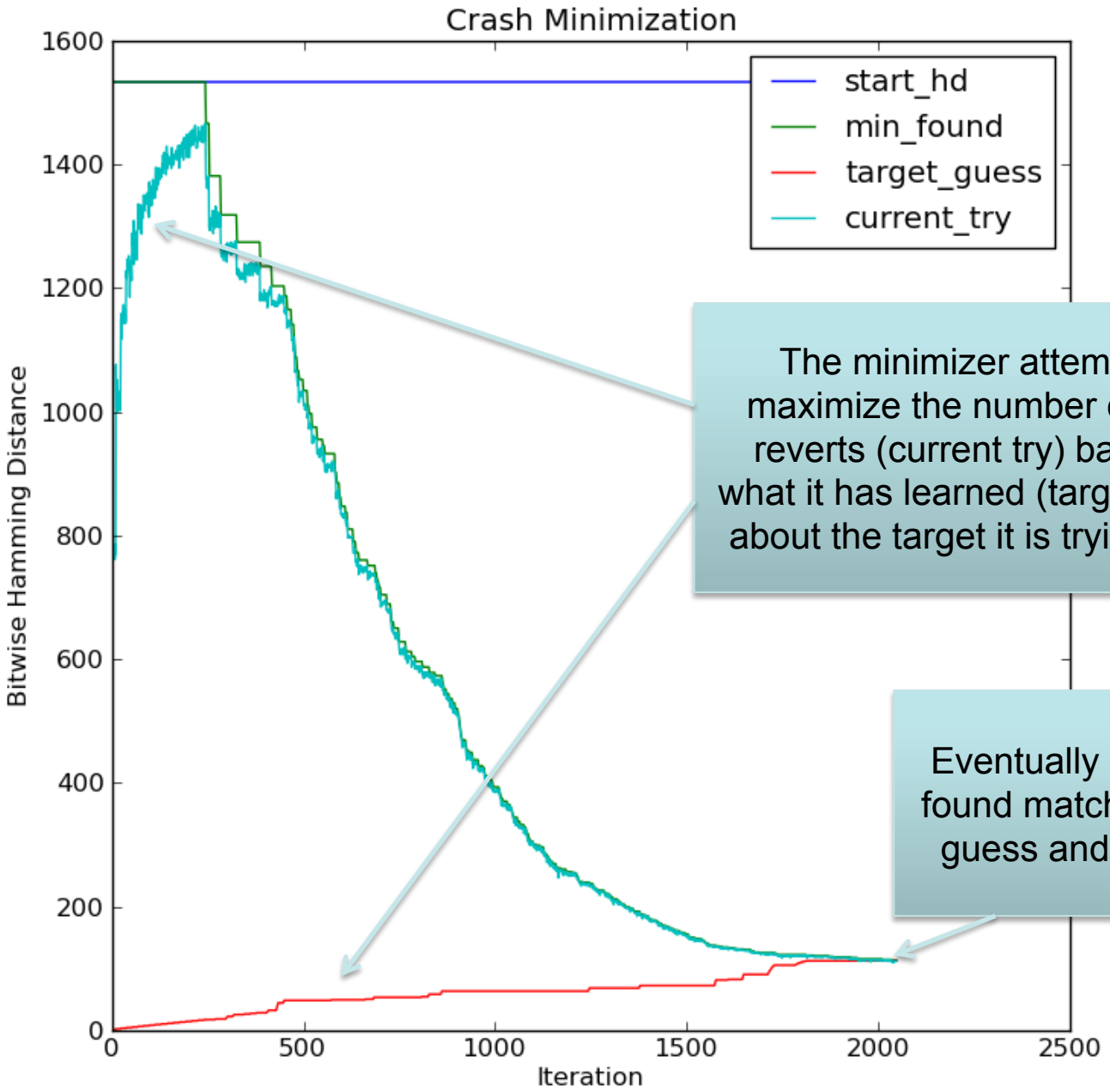
Known good seedfile – does not cause crash



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



The minimizer attempts to maximize the number of bits it reverts (current try) based on what it has learned (target guess) about the target it is trying to hit.

Eventually the minimum found matches our target guess and we're done.

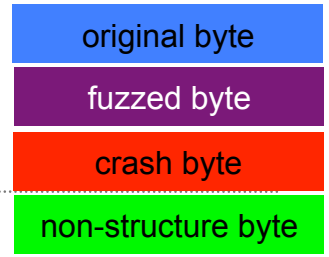
Minimize to string

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

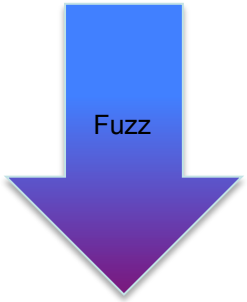
We want to know:

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

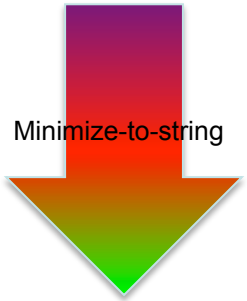
What Minimize-to-String Does



Known good seedfile – does not cause crash

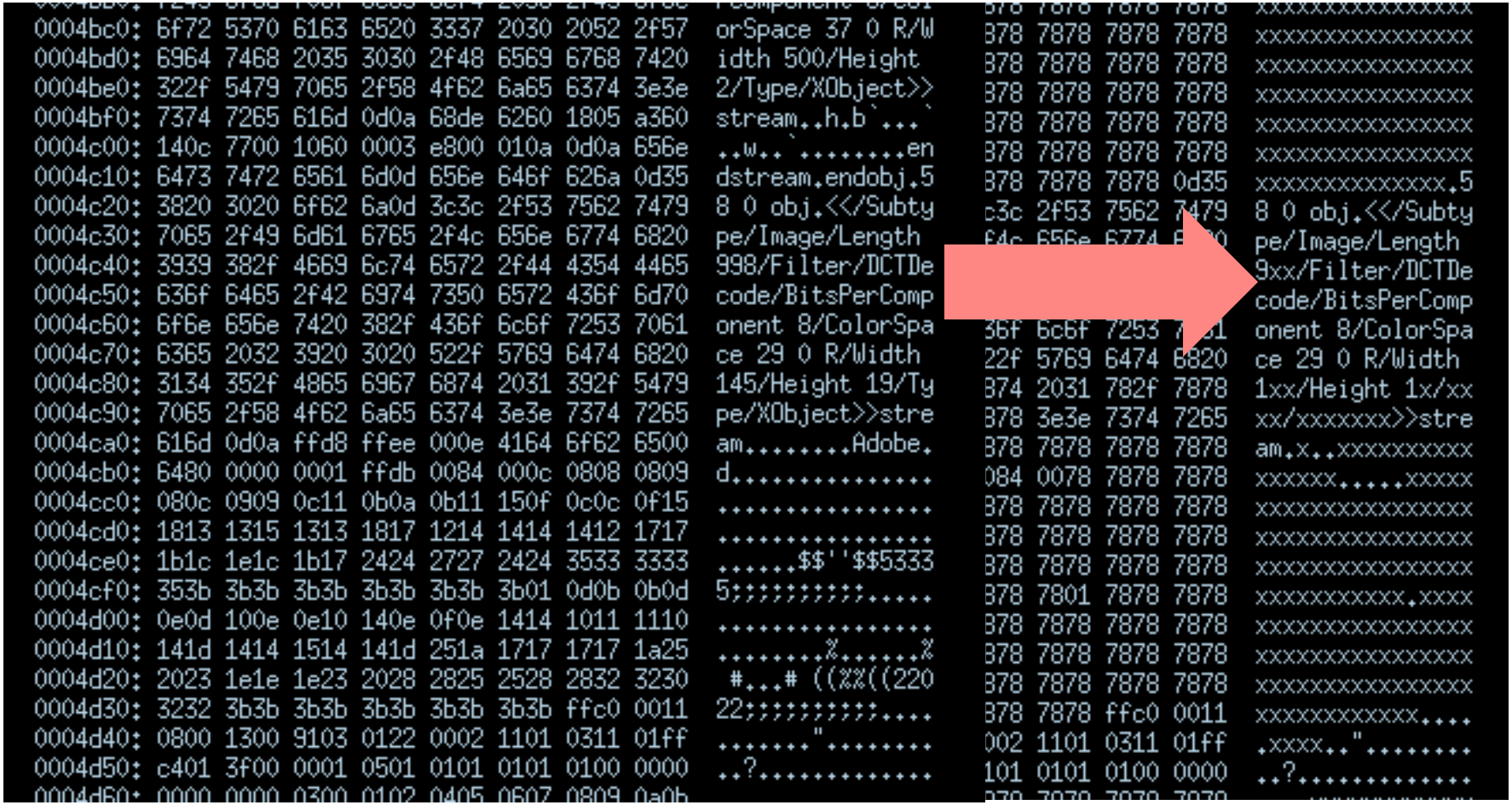


Fuzzed file – causes crash, many changed bytes are not involved in the crash



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

Minimize to string example



Minimize to string downside

It's a more complex problem to solve.

- It's slow!

Mitigation:

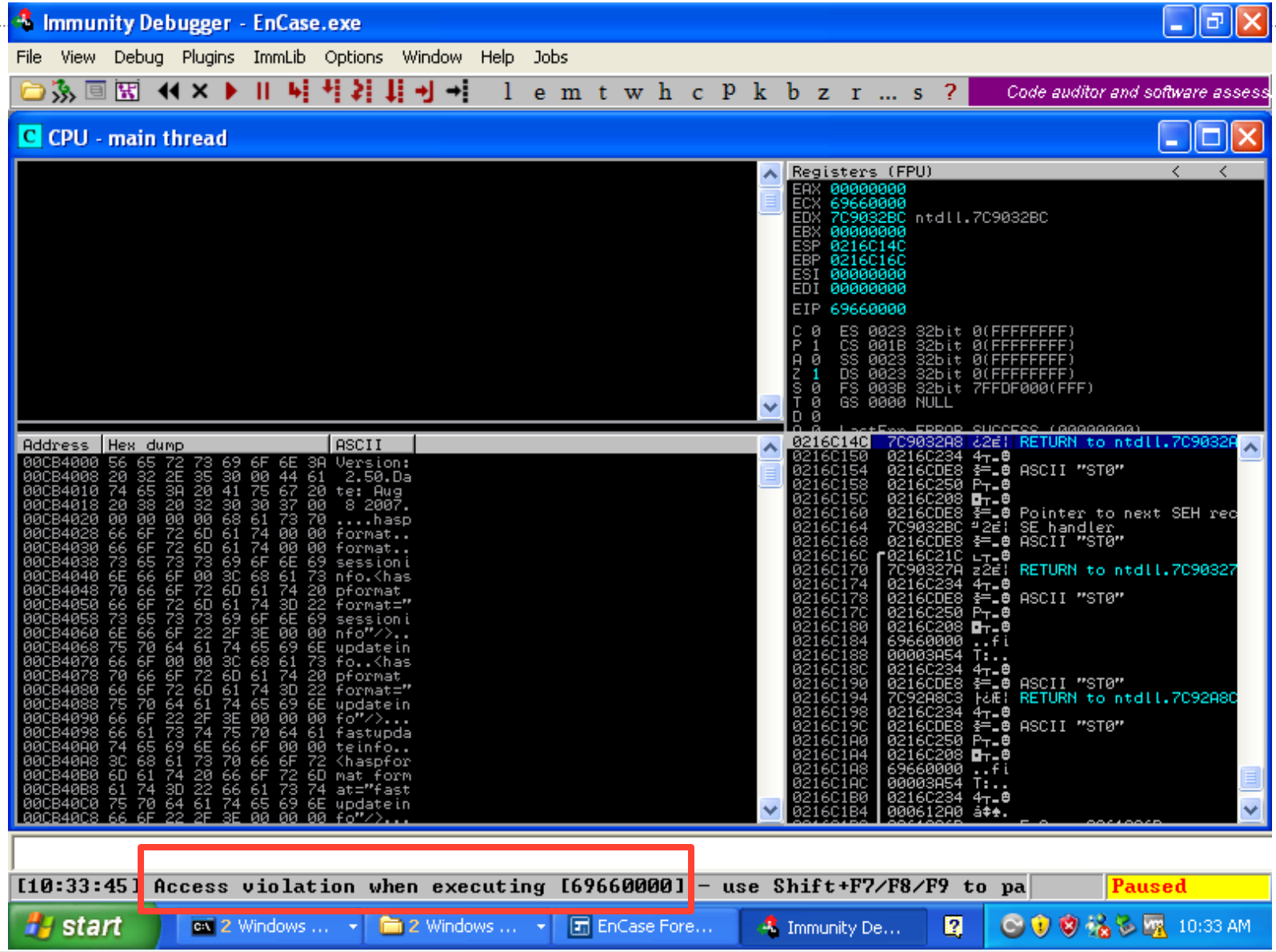
Only run it for cases that you want to write a PoC for.

Writing a PoC

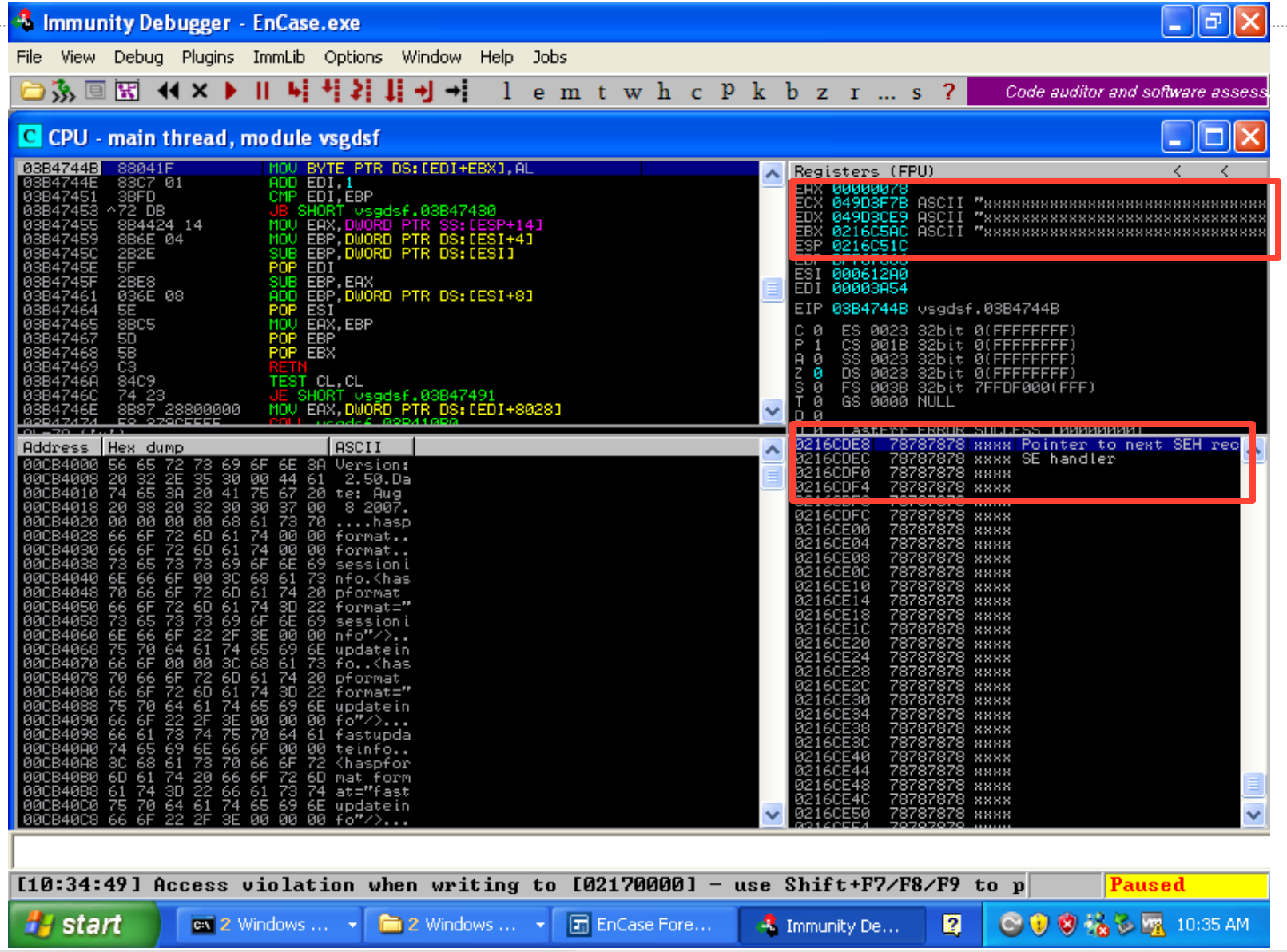
Achieving code execution with a memory corruption vulnerability requires two pieces of knowledge:

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

The original crash



The minimized-to-string crash



Which 0x78787878 ?

Minimization to x shows:

- Which bytes are under my control (‘xxxx…’)
- How to get there (JMP ECX)

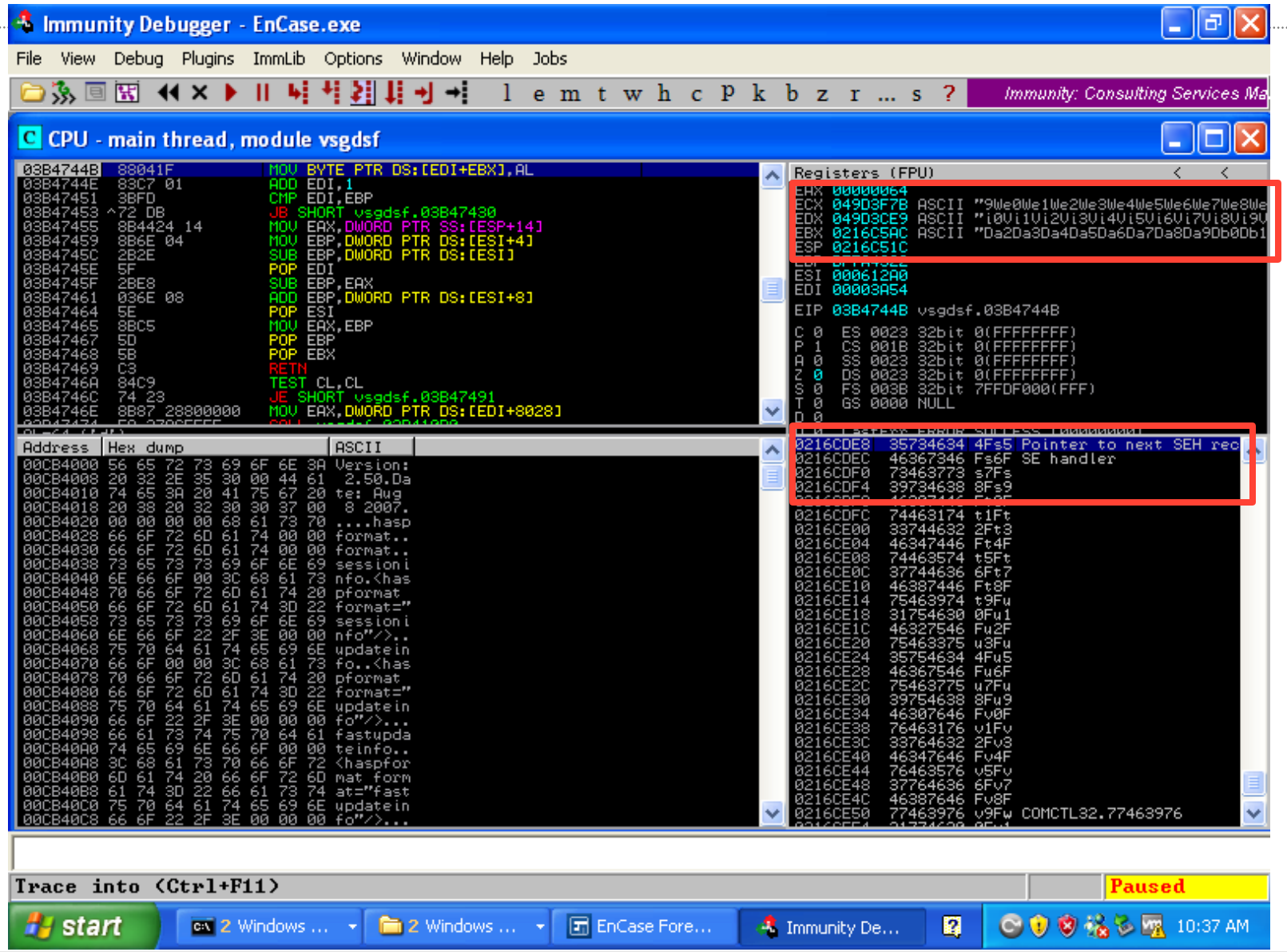
The problem:

Which ‘x’ is which?

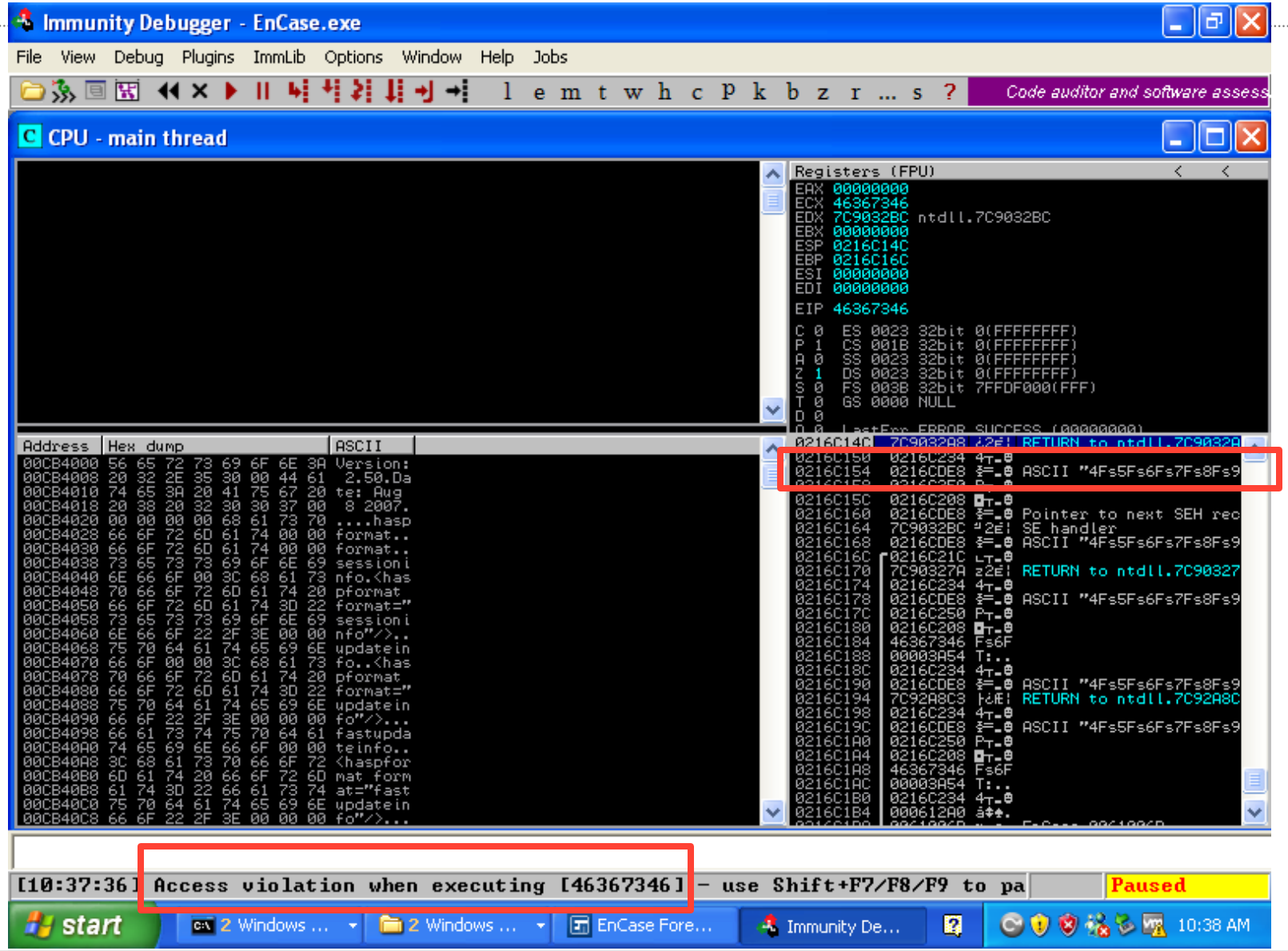
The solution:

Metasploit string pattern.

The minimized-to-Metasploit crash



The minimized-to-Metasploit crash



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 *AppInit_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.

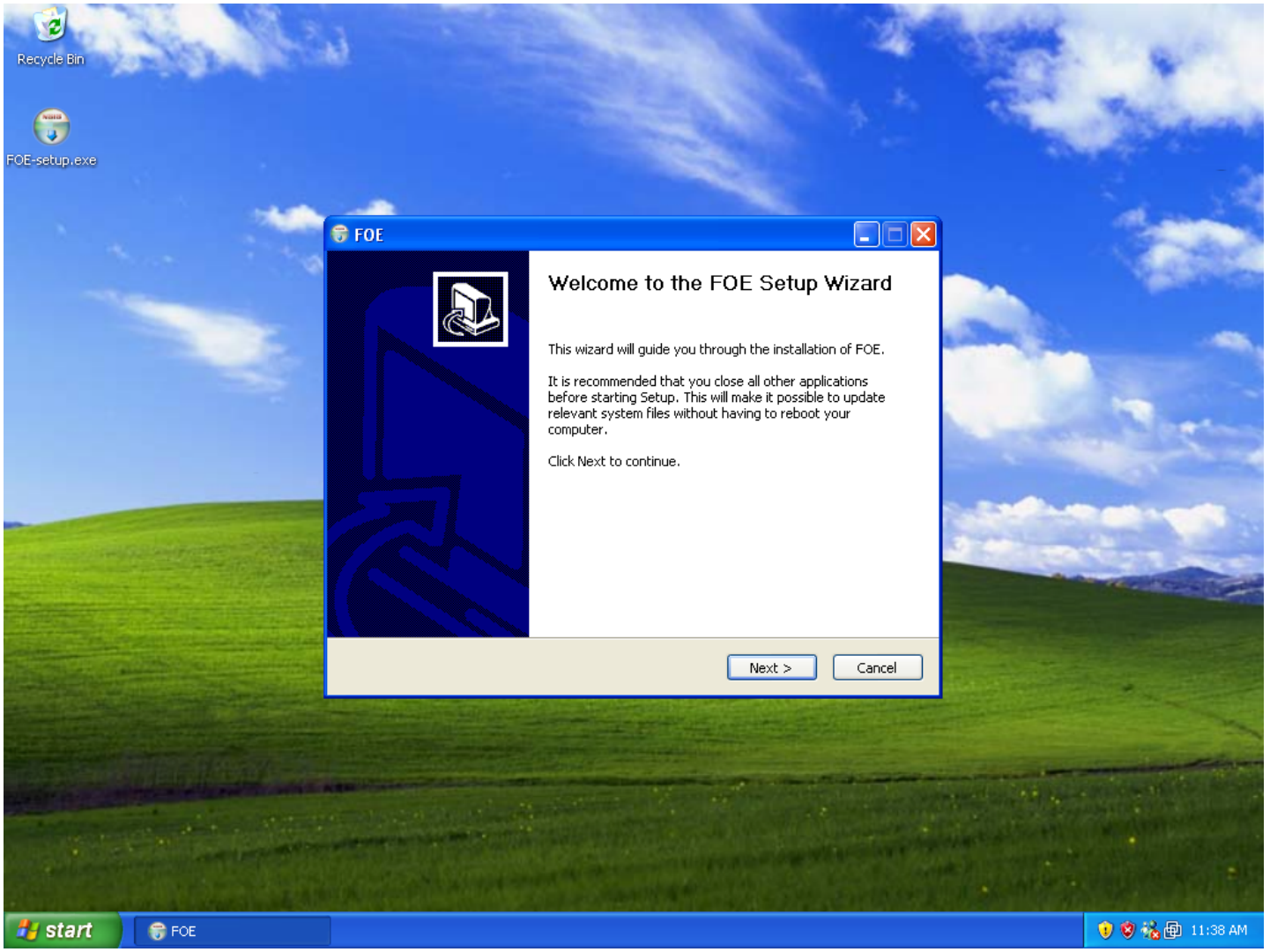
Solution: drillresults.py

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

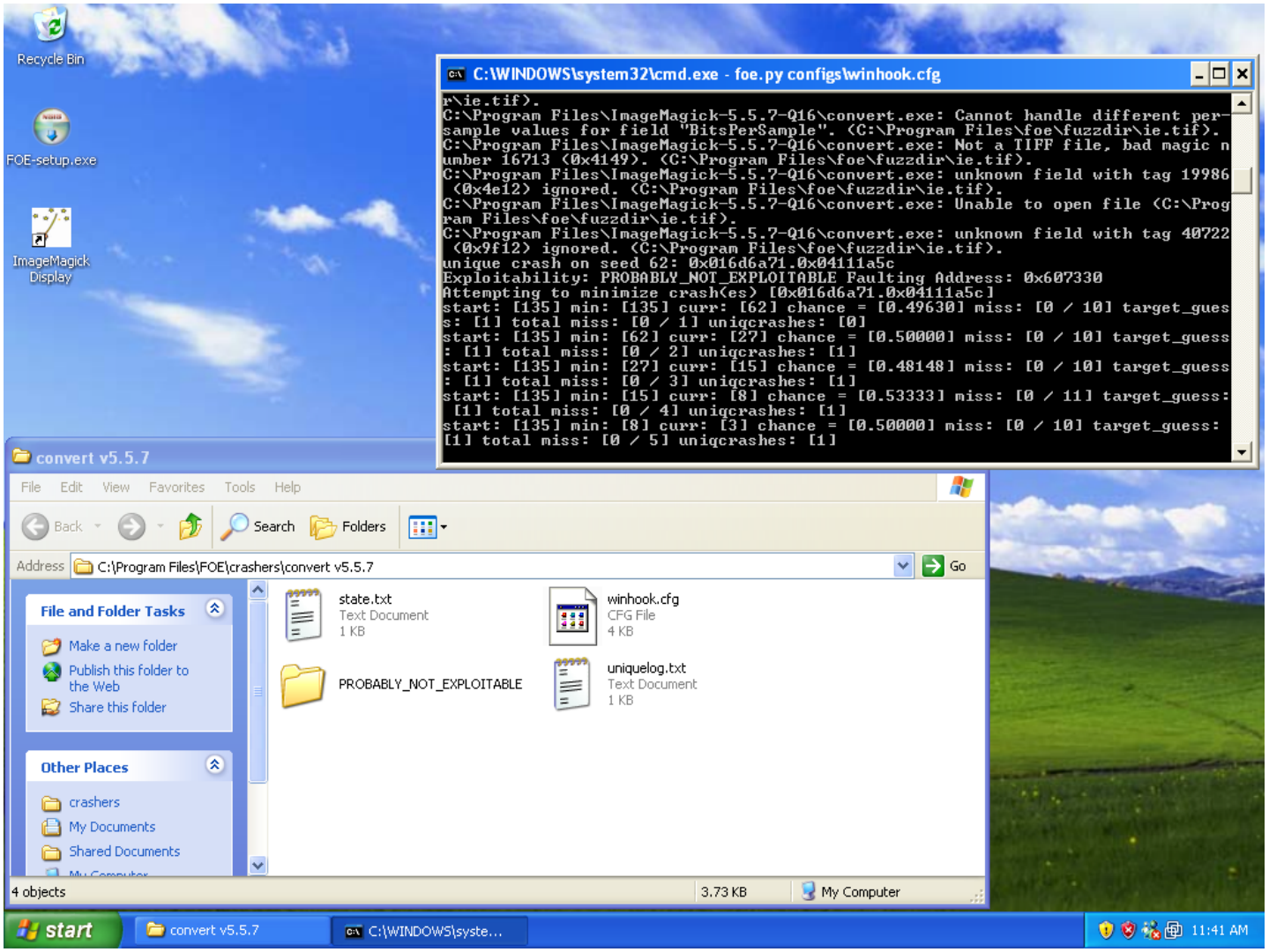
drillresults.py output

```
0x1c636361.0x1a2f7629 - Exploitability rank: 10
Fuzzed file: results\oi-multi-2\PROBABLY_EXPLOITABLE
\0x1c636361.0x1a2f7629\sf_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
\sf_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\sf_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
```

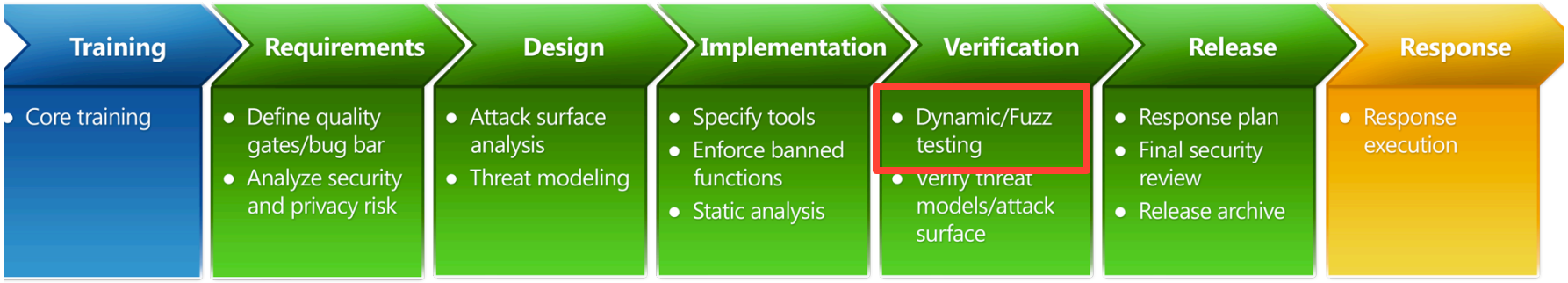
The CERT® FOE


Recycle Bin
FOE-setup.exe
FOE
Welcome to the FOE Setup Wizard
This wizard will guide you through the installation of FOE.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel
start
FOE
11:38 AM

The CERT® FOE

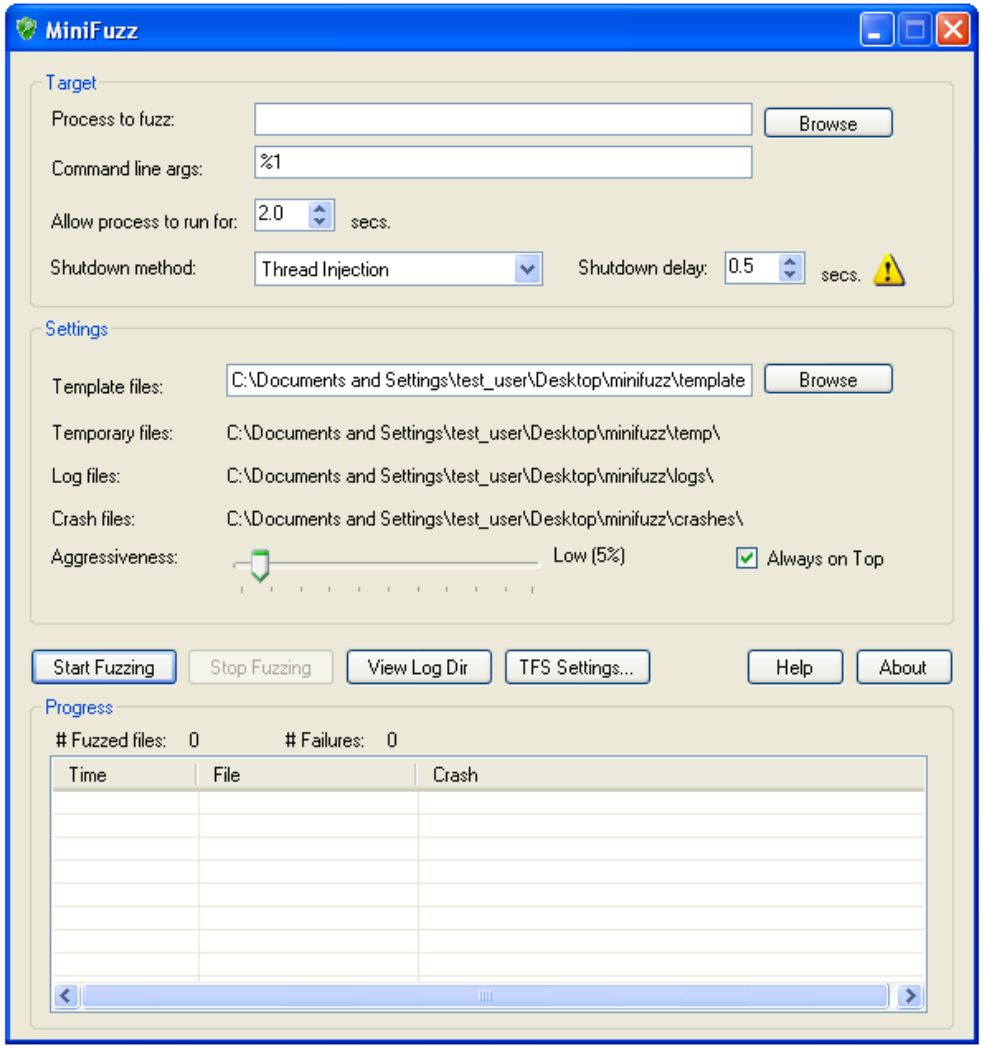

Recycle Bin
FOE-setup.exe
ImageMagick Display
C:\WINDOWS\system32\cmd.exe - foe.py configs\winhook.cfg
r\ie.tif).
C:\Program Files\ImageMagick-5.5.7-Q16\convert.exe: Cannot handle different per-sample values for field "BitsPerSample". (C:\Program Files\foe\fuzzdir\ie.tif).
C:\Program Files\ImageMagick-5.5.7-Q16\convert.exe: Not a TIFF file, bad magic number 16713 (0x4149). (C:\Program Files\foe\fuzzdir\ie.tif).
C:\Program Files\ImageMagick-5.5.7-Q16\convert.exe: unknown field with tag 19986 (0x4e12) ignored. (C:\Program Files\foe\fuzzdir\ie.tif).
C:\Program Files\ImageMagick-5.5.7-Q16\convert.exe: Unable to open file (C:\Program Files\foe\fuzzdir\ie.tif).
C:\Program Files\ImageMagick-5.5.7-Q16\convert.exe: unknown field with tag 40722 (0x9f12) ignored. (C:\Program Files\foe\fuzzdir\ie.tif).
unique crash on seed 62: 0x016d6a71.0x04111a5c
Exploitability: PROBABLY_NOT_EXPLOITABLE Faulting Address: 0x607330
Attempting to minimize crash(es) [0x016d6a71.0x04111a5c]
start: [135] min: [135] curr: [62] chance = [0.49630] miss: [0 / 10] target_guess: [1] total miss: [0 / 1] uniqcrashes: [0]
start: [135] min: [62] curr: [27] chance = [0.50000] miss: [0 / 10] target_guess: [1] total miss: [0 / 2] uniqcrashes: [1]
start: [135] min: [27] curr: [15] chance = [0.48148] miss: [0 / 10] target_guess: [1] total miss: [0 / 3] uniqcrashes: [1]
start: [135] min: [15] curr: [8] chance = [0.53333] miss: [0 / 11] target_guess: [1] total miss: [0 / 4] uniqcrashes: [1]
start: [135] min: [8] curr: [3] chance = [0.50000] miss: [0 / 10] target_guess: [1] total miss: [0 / 5] uniqcrashes: [1]
convert v5.5.7
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Program Files\FOE\crashers\convert v5.5.7
Go
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
crashers
My Documents
Shared Documents
state.txt
Text Document
1 KB
winhook.cfg
CFG File
4 KB
PROBABLY_NOT_EXPLOITABLE
uniquelog.txt
Text Document
1 KB
4 objects
3.73 KB
My Computer
start
convert v5.5.7
C:\WINDOWS\syste...
11:41 AM

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

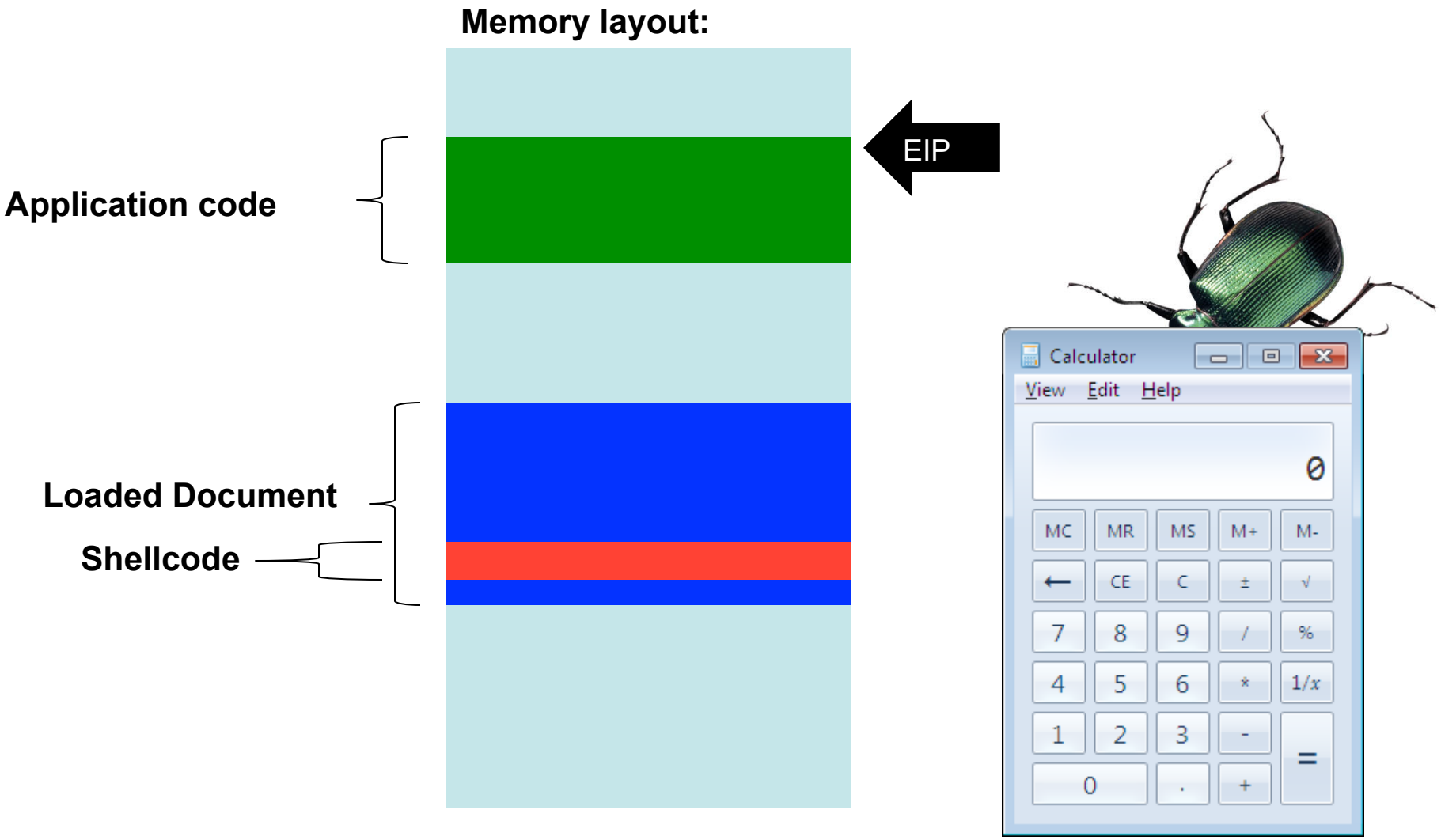
- 141 Total unique crashes

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

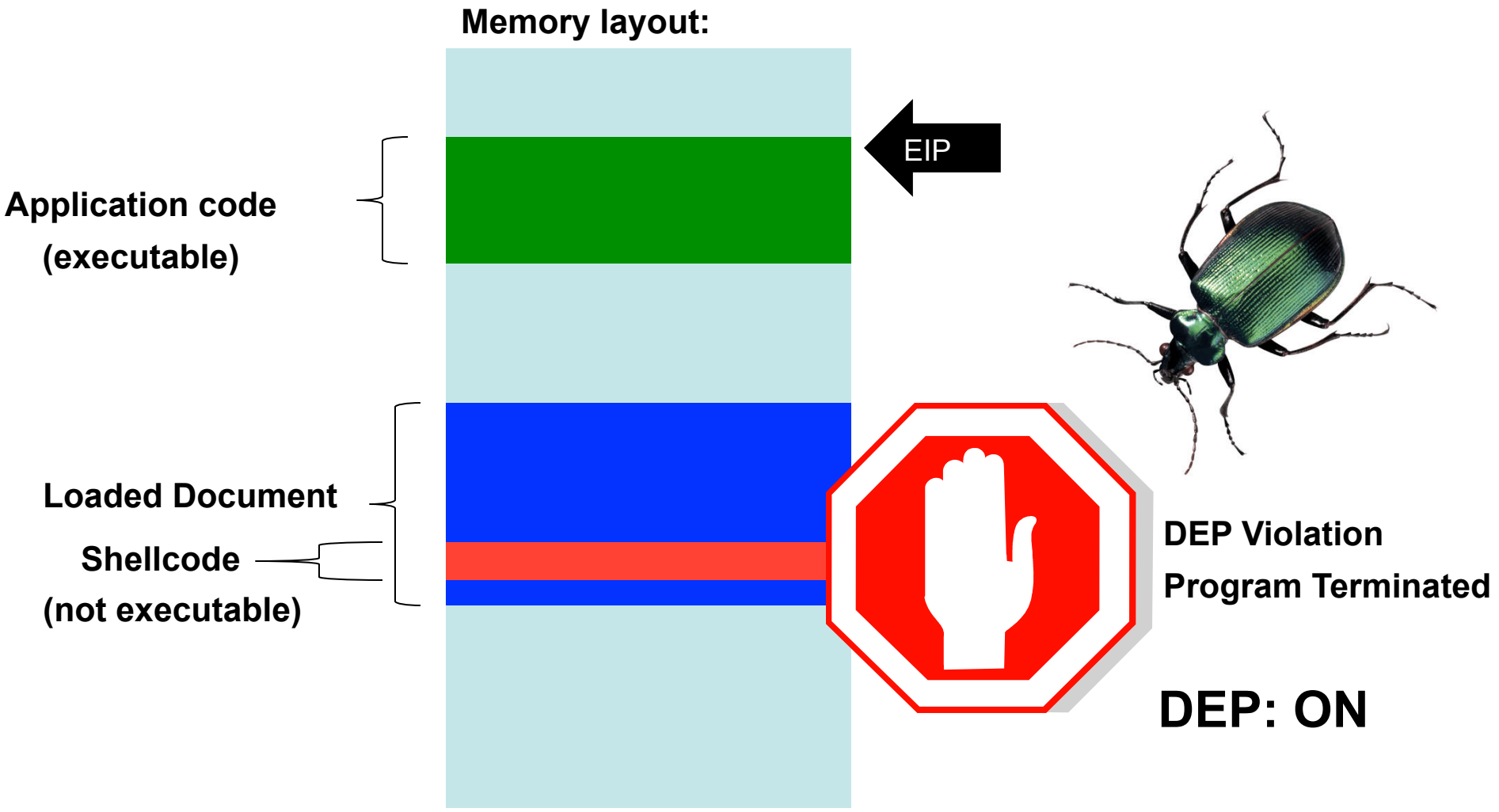
Exploitation Protections

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?

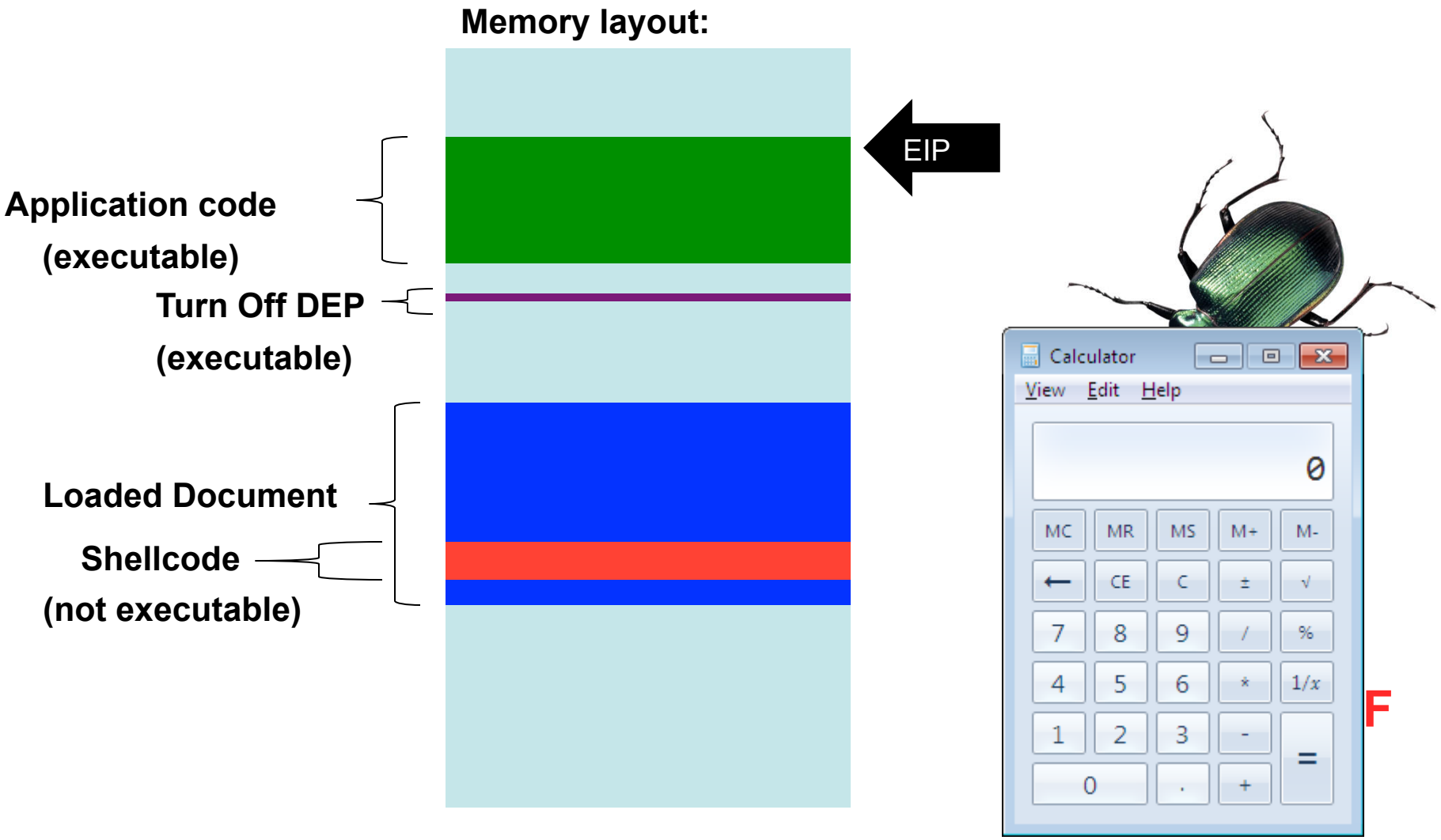
Return Oriented Programming

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

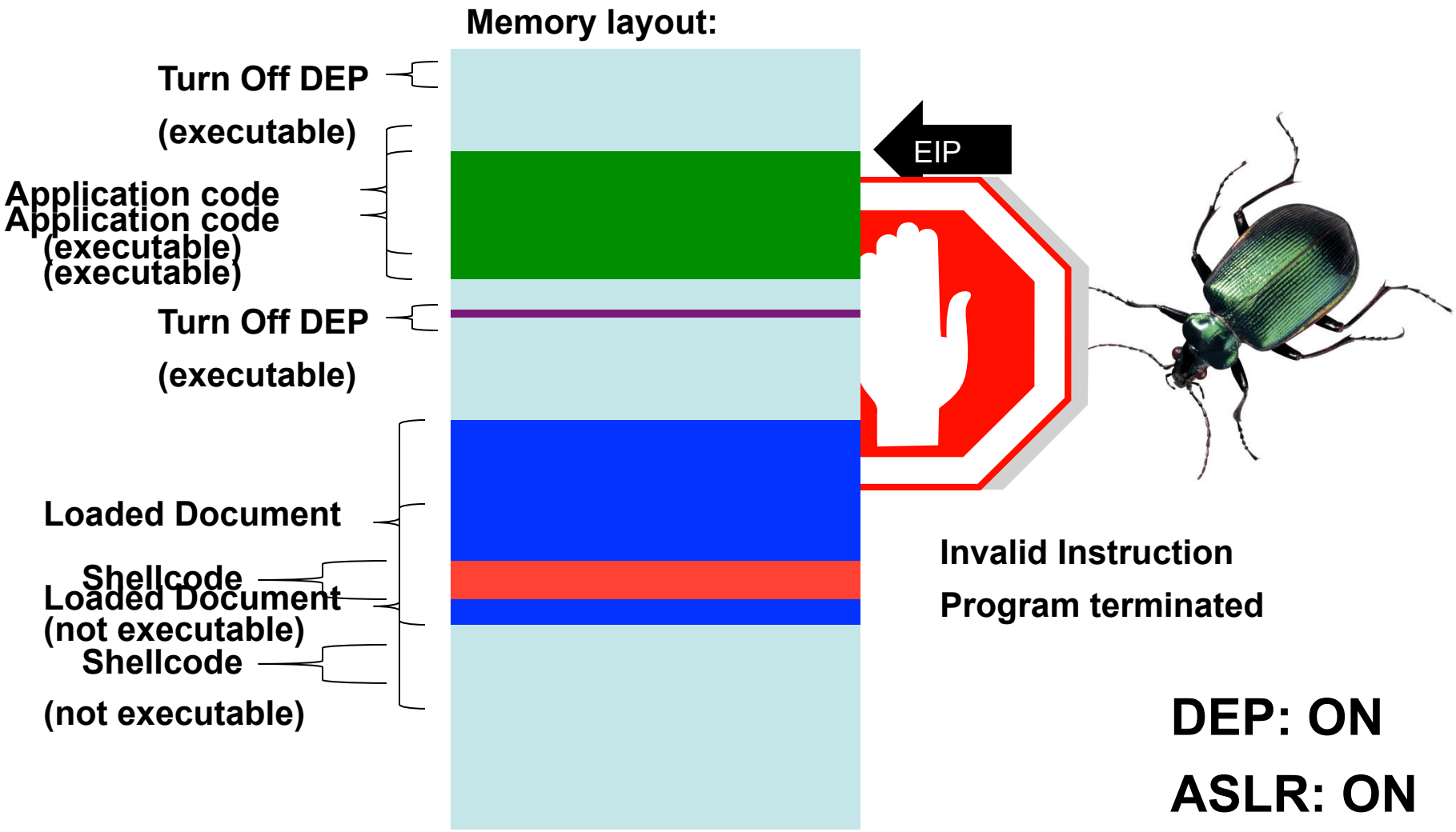


Protection #2: ASLR

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

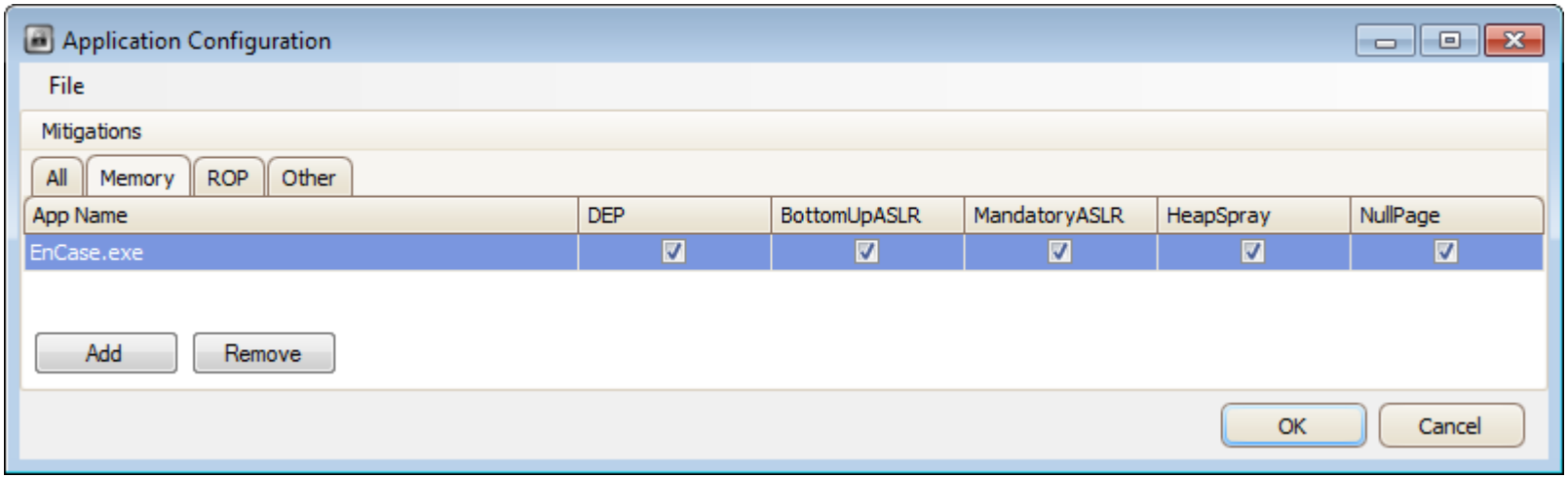
Microsoft EMET

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

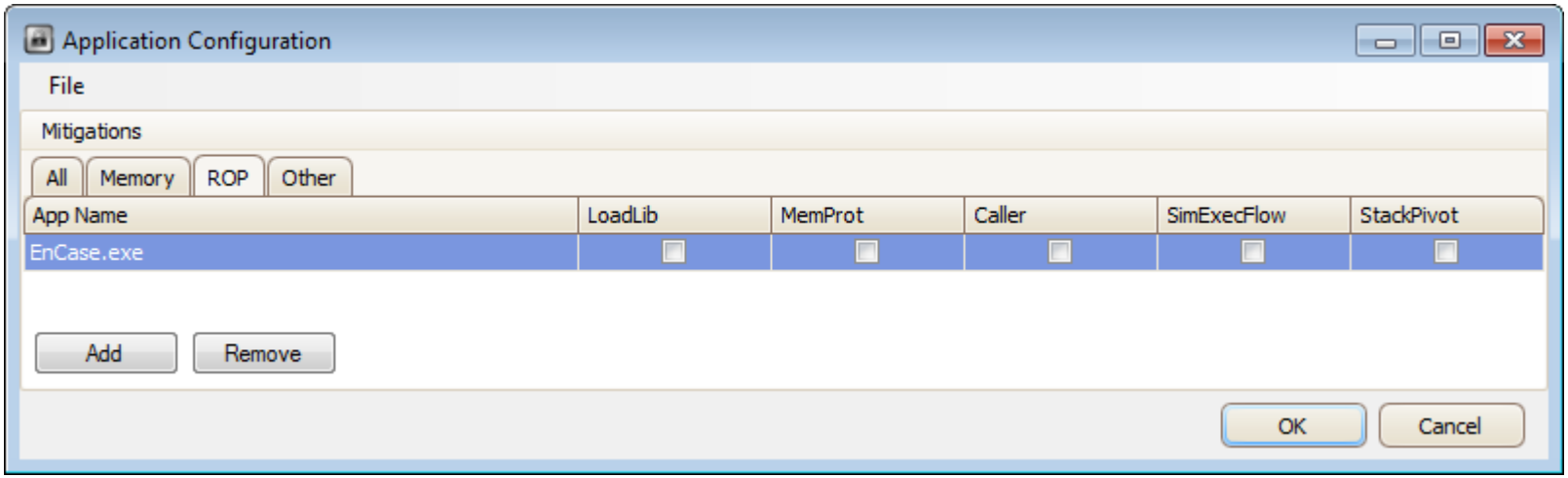
Everyone's a winner!

ASLR Requires Vista or Newer

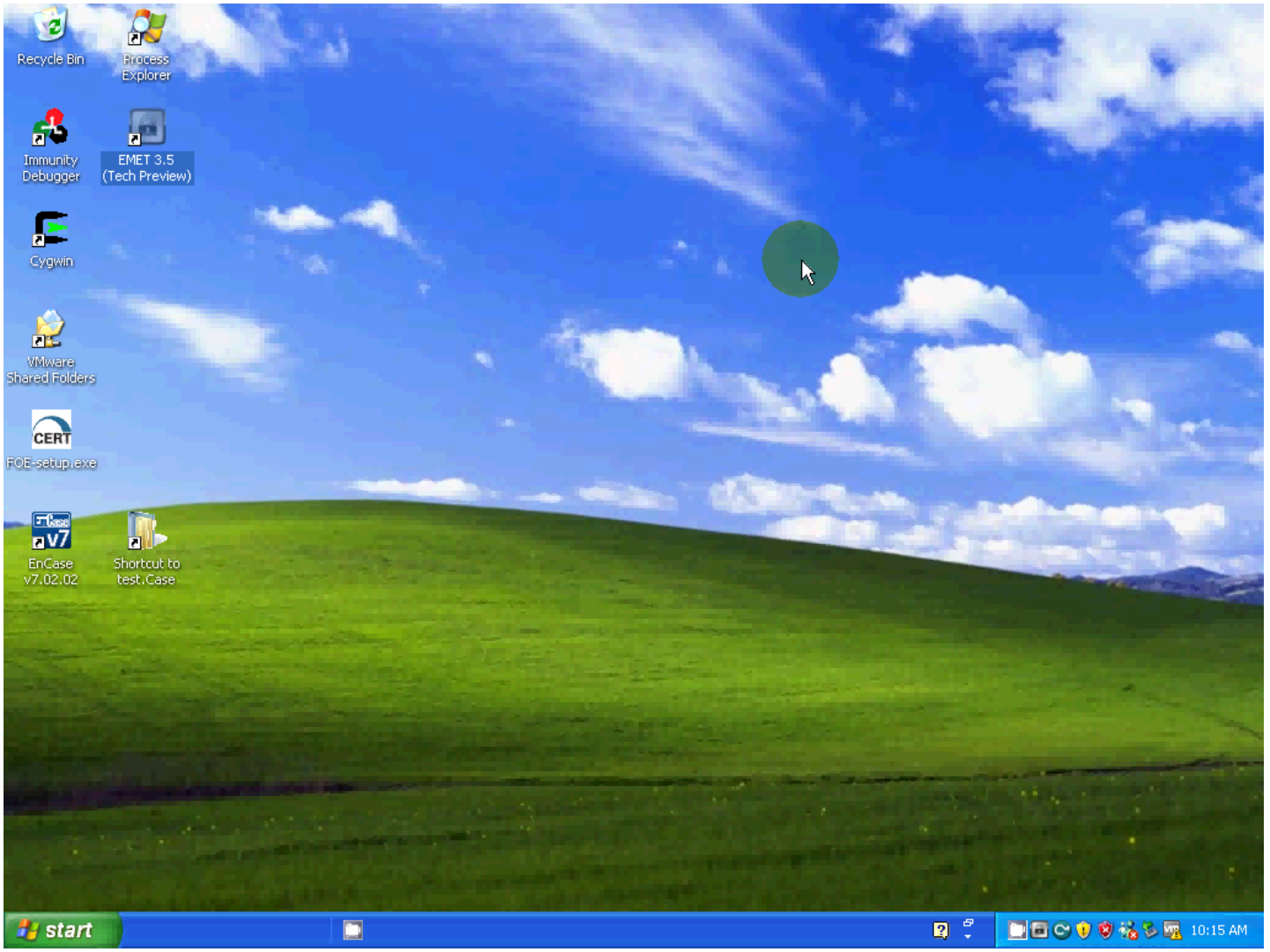
Windows XP (Server 2003) does not support ASLR!

ROP Mitigations

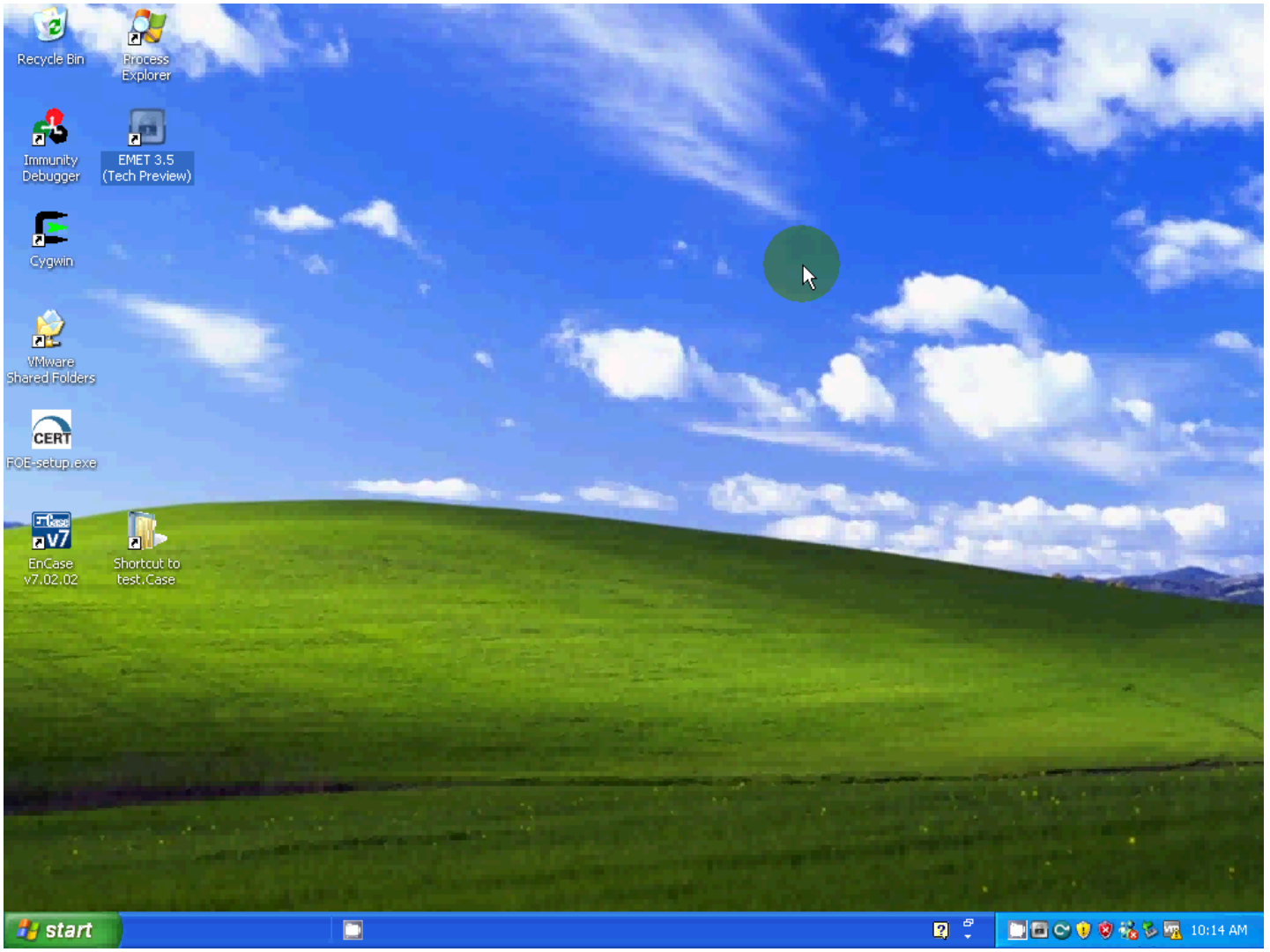
EMET 3.5 introduces explicit ROP mitigations



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
- Iceni 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