

RUNTIME Decompilation

Using a Dynamic Sampling Decompiler

By: Greg Hoglund
HBGary, LLC.

Chapter One

Methodologies

White Box

- **White box:**
 - operating with full knowledge about the inner workings of the system
- Can be used with source code or with deadlisting*
- We build or use a **mental model** based on
 - Intent as the builder
 - Understanding of source code or deadlisting*

*the disassembly of the binary program

The model is not the machine

We know the mental model is not accurate

- The model is a close approximation of what is really happening in the machine
 - We hope the model has similar properties and behavior to the machine, this helps us make predictions
- Emergent properties of software
 - Without which we would not have (a very large) computer security industry
 - Emergent properties are often based on complex behavior that is not replicated in the model
 - The model is best when we already know what we are looking for

Automatic Reverse Engineering

- Effective when certain conditions exist
 - Availability of type information
 - Separation of data and code
 - All instructions can be recovered
 - Data that drives control flow can be mapped with enough resolution
 - Just enough to help us find a potential vulnerability and filter out the false positives

Branching Decisions

- Many branches are made based on values that are calculated at runtime
- The static analyzer must emulate execution to determine these values
- At some point, is the emulation is computationally equivalent to running the program in the first place?

Back-traces reach dead ends

- Back traced cross references can be used to connect input with a code location
- Many times a static backtrace dead-ends
 - Windows message handler
- We need to run the program to trace where the input is coming from

Black Box

- All we see are the outputs from the software – no inner workings
- Requires deep protocol knowledge to build one a fuzzer
 - ‘Fuzzers’:
 - Hailstorm and Spike
- Requires no knowledge to run one
- Automated (unattended)

Grow old waiting for this

- Fuzzers take FOREVER to complete their input sequences.
- If the program is slow, this compounds the problem
- Amounts to 'brute forcing'
- Crashes require a skilled debugger to determine if an attacker can exploit the fault

Evolution to Grey-box?

- Combine fault-injection with code analysis
- When you use a program debugger, you are performing grey-box analysis
- Performed at runtime so software can be observed
- All instructions which are executed can be obtained. All data involved at these points can be tracked

Chapter Two

The Bugs

Easy Stuff

- These can be scanned for in static code
 - NULL termination on strings
 - strcpy, etc
 - Off by one in string operations
 - strncpy, etc
 - Signed/Unsigned conversion errors
 - Format strings

Hard Stuff

- These require runtime analysis
 - Crafted-input parsing
 - State corruption
 - Control flow through computed values
 - API call data *indirectly* calculated from user-supplied data*

*to do this statically requires emulation – this only makes sense if you cannot run the program or the code location cannot be reached using reasonable input

Is it actually exploitable?

- Depends on many variables in the environment
- **All** automatic analysis tools have this problem
- It almost ***always takes an expert*** reverse engineer to determine if a condition is exploitable

Does it matter?

- Even if a vulnerability cannot be reached ***today*** – what can you say about ***tommorrow?***
- What if interface changes?
- What if code gets used from other locations?
- Is the original author going to be maintaining this code in 10 years?

Chapter Three

Bug Scan

Easy Stuff – Introducing BugScan!

- BugScan is **extremely simple** to use
- Submit binary and get report
- Report cannot verify if conditions are actually exploitable
 - But it takes 30 seconds, not 30 hours
 - Defensive stance – don't wait for someone to attack before you protect yourself

Submit a File



View the Report

BugScan v1.0 [Browse Reports](#) [Analyze a File](#) [Log Off](#)

< Previous Next >

ID	Sev	Name	Risk	Description
0	2	wsprintfA	overflow	Replace this call with the more secure call, wsnprintf. wsnprintf is a variant of wsprintf where the user explicitly specifies the length of the destination buffer. This feature helps avoid the possibility of the destination buffer being written past. Though not officially a part of the ISO C99 standard, this call is available in most modern compilers. Two problems can still persist: format string bugs and specifying an incorrect length for the destination buffer.
1	2	lstrcpyA	low	Double check that the destination buffer is larger than the specified length in this call. If the supplied string is exactly equal to the size of the destination buffer, the string will not be NULL terminated.
2	2	lstrcpyA	overflow	Replace this call with the more secure call, lstrcpy. lstrcpy is a variant of lstrcpyA where the user explicitly specifies the length of the destination buffer. This feature helps avoid the possibility of the destination buffer being written past. Though not officially a part of the ISO C99 standard, this call is available in most modern

Discussions not available on http://216.27.185.83/

Internet

Desktop Firewall

“Engine.DLL” (overall, fairly good)

<code>sprintf</code>	7
<code>wsprintfA</code>	2
<code>sscanf</code>	2
<code>_snprintf (good)</code>	11
<code>strncpy (good)</code>	19
Signed/unsigned mismatches	1

“████.exe” (overall, fairly good)

<code>wsprintfA</code>	8
Signed/unsigned mismatches	5

Win32 Apache

“Apache.exe” (good)

Nothing at all!

“ApacheCore.DLL” (this should be spic-n-span)

<code>sprintf</code>	5
<code>strcat</code>	3
<code>strcpy</code>	3
<code>sscanf</code>	3
<code>_snwprintf</code> (good)	24
Signed/unsigned mismatches	14

FTP Server

“ftp.exe” (a little unsettling)

lstrcpyA

29

Signed/unsigned mismatches

1

AV Auto-update

“mserver.exe” (dangerous!)

<code>lstrcpyA</code>	5
<code>lstrcpyA</code>	12
Signed/unsigned mismatches	50

Trillian

“trillian.exe” (a little unsettling)

`wsprintfA`

36

`lstrcpyA`

4

Signed/unsigned mismatches

6

“irc.dll” (not that bad)

Signed/unsigned mismatches

9

“http.dll” (not that bad)

Signed/unsigned mismatches

6

Video-conferencing (H323)

“████████f.exe” (I’m getting very nervous)

wsprintfA	36
lstrcpyA	47
Signed/unsigned mismatches	16
lstrcpynA (good)	42

“████████chat.exe” (I’m getting very nervous)

wsprintfA	15
lstrcpyA	19
lstrcpynA	2

More on bugscan

www.bugscaninc.com

info@bugscaninc.com

310-654-8745

Chapter Four

Tempest

Introducing TEMPEST



Hard Stuff

- Designed for experts working in a lab process
- Requires reverse engineering skills not limited to:
 - Runtime debugging
 - Assembly code
 - Protocols
 - Technical knowledge of programming bugs

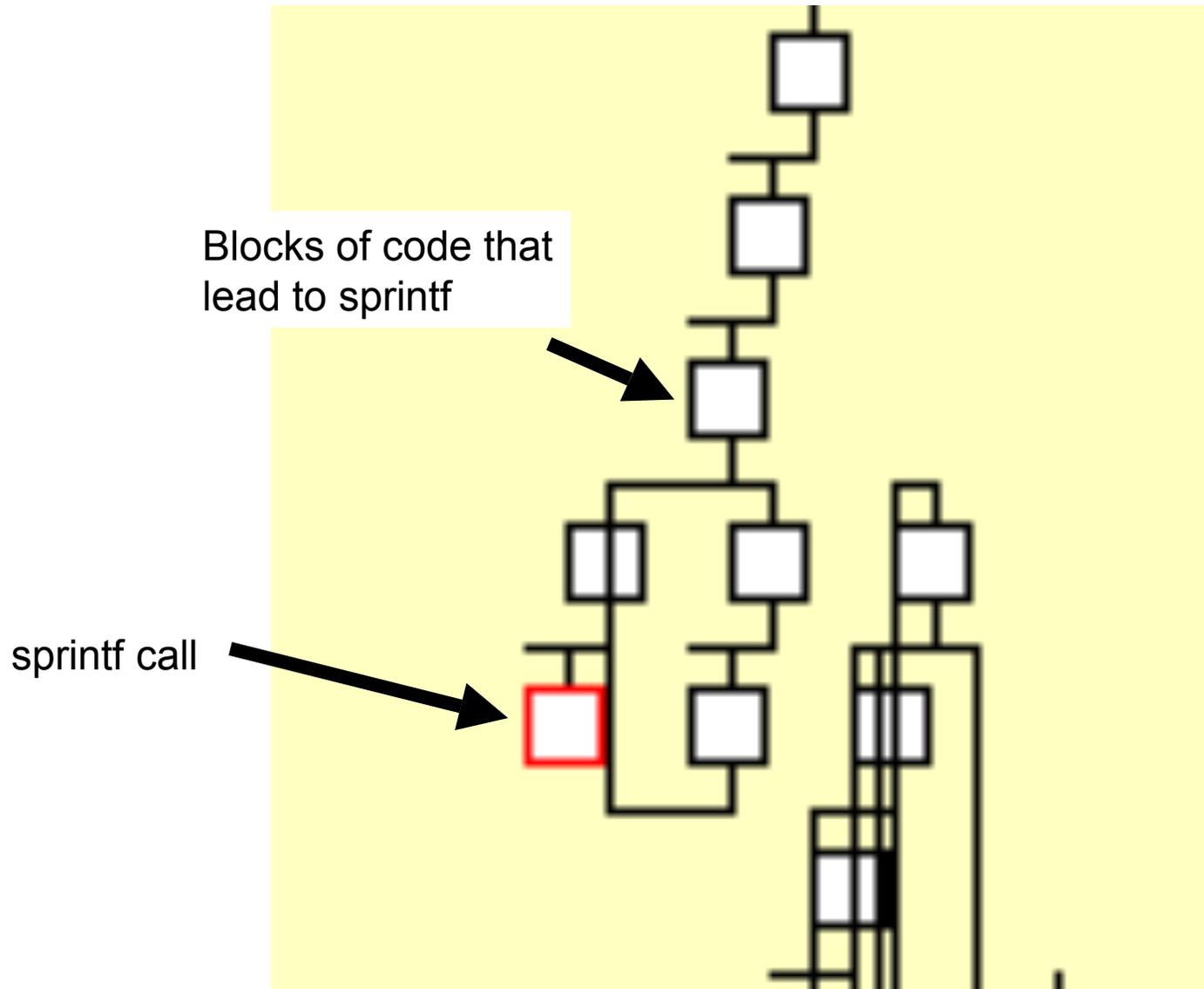
TEMPEST

- Connect the inputs with the bugs
- Verify the exploit
- Build a working exploit
- Offensive stance – find working injection vectors
- Based on a **WORKFLOW** process
 - This is NOT a product

How does it work?

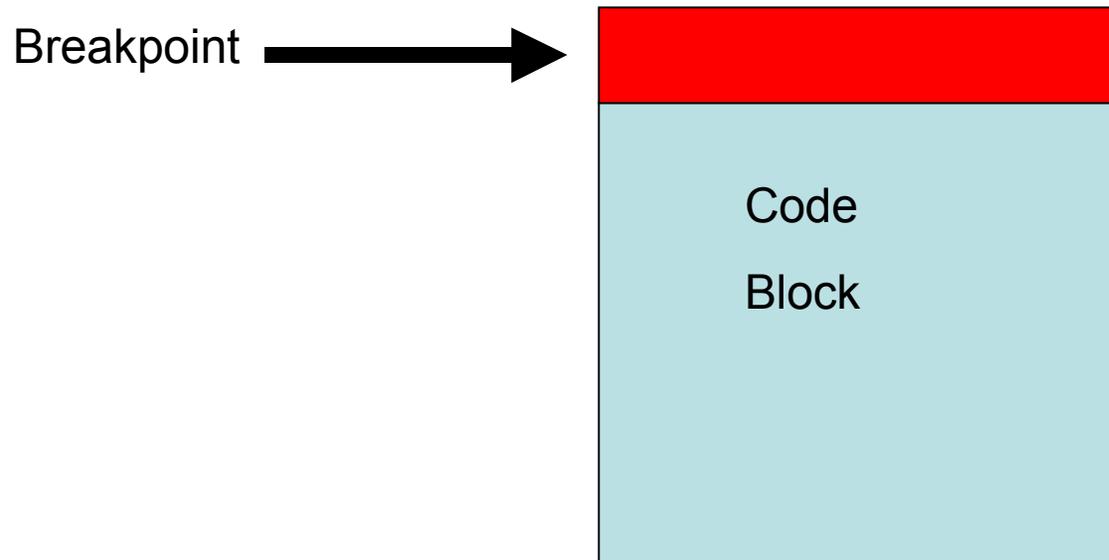
1. Find locations using static analysis
 - IDA Pro is a good choice for this
2. Static backtraces from potential vulnerable points
3. Dynamic forward traces from user-input points
4. Tune your fuzzer until you “connect the dots”!

Static backtrace from suspect locations



Coverage

- As program is used, if a code block is visited it will be highlighted 'grey'*



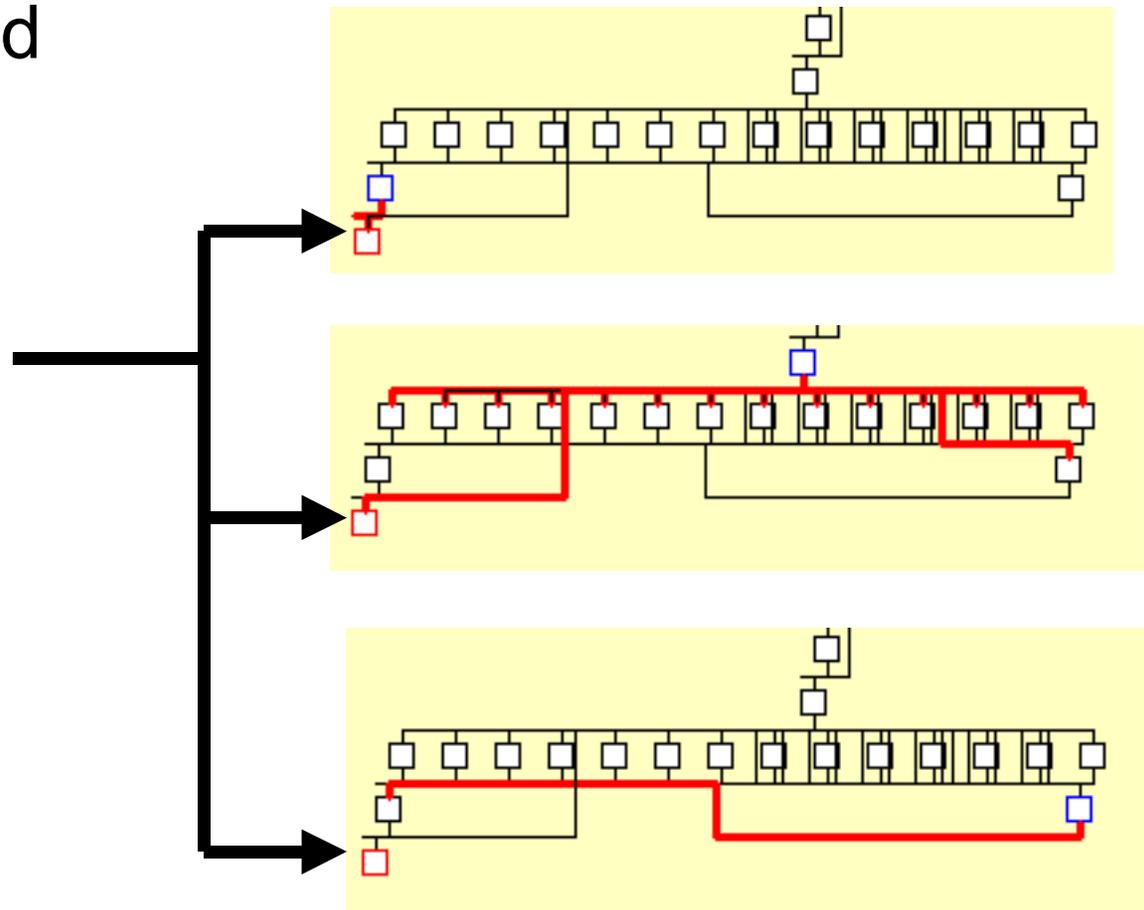
*this technique published by Halvar Flake, BlackHat Briefings (www.blackhat.com)

Fly-By's & Drill Downs

- If we hit code blocks 'above' a suspect location we are alerted to potential operations that will cause the target to be exercised
- Coverage helps us tune our input data to drill down to a target location
 - This is the fundamental advantage

Trillian IRC DLL

Signed/Unsigned
mismatch in
subroutine
at 0x1000FE40

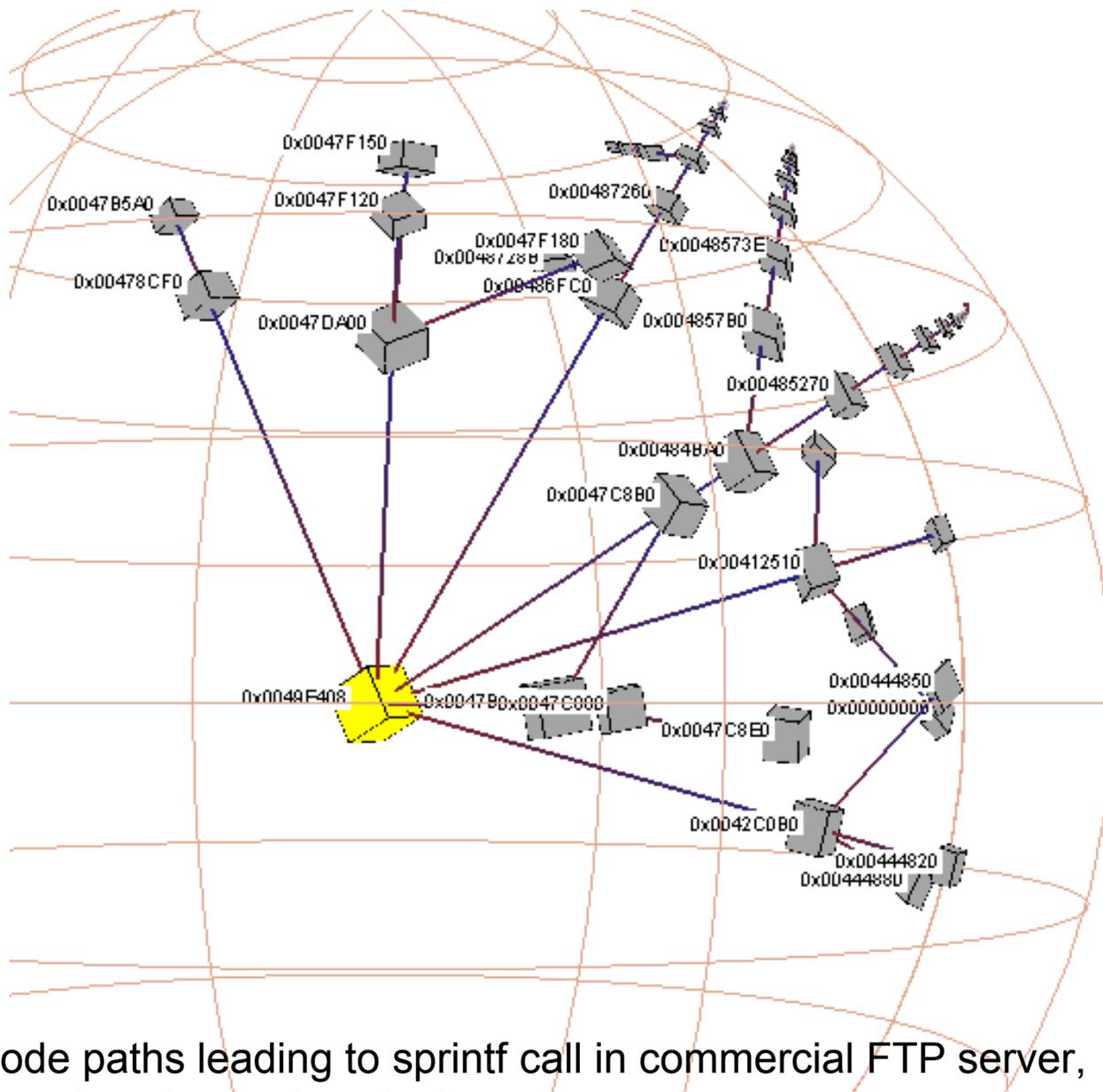


Graphing Problems

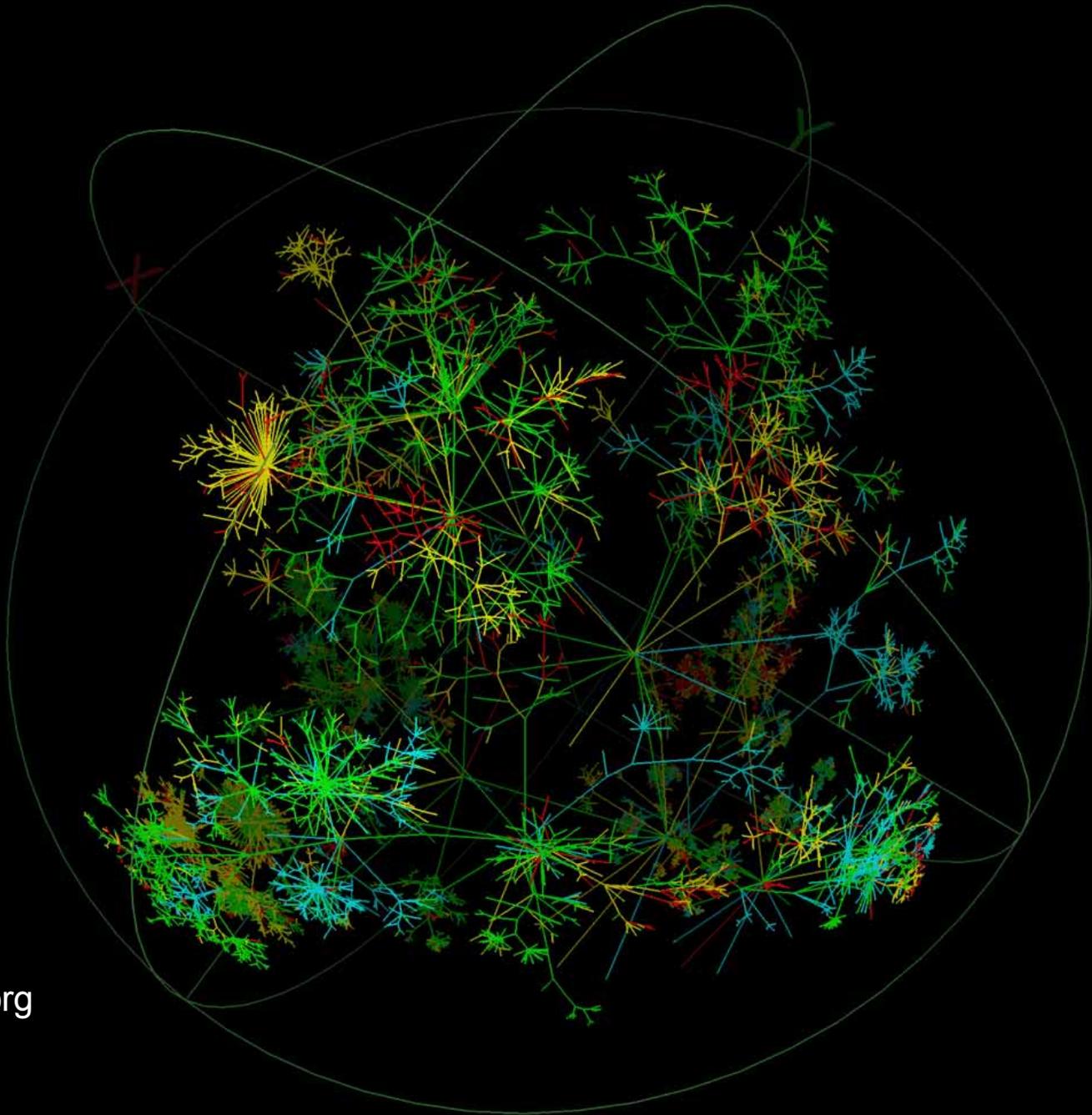
- Graph complexity increases with the number of back traces
- Using tempest on more than a few target points at a time results in a huge, unwieldy graph

Advanced Graphing

- Different graphing algorithms can be used
- Hyperbolic graphs serve better for browsing a large number of nodes



All code paths leading to sprintf call in commercial FTP server, information obtained statically



WALRUS

www.caida.org

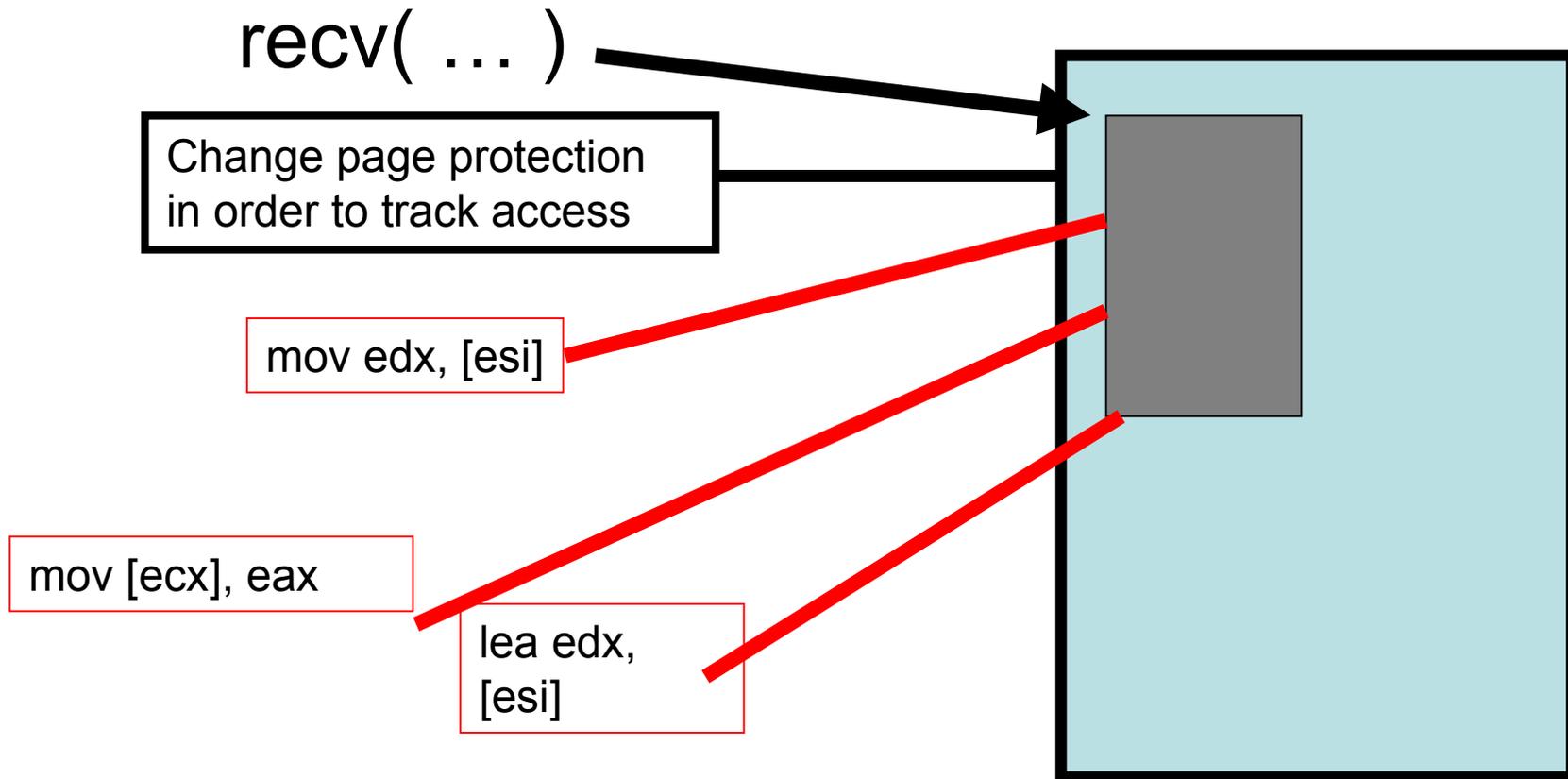
Filtering the set

- Don't worry about sprintf if the format string doesn't contain %s
- Don't worry about off by ones if the size parameter is less than the stack correction
- Don't worry about ***anything*** if the source data is ***not*** obtained from outside the function

Boron Tagging

- Traces from known points
- Breakpoints on suspect calls
- Can be used as a strategy to skip large sections of the graph
 - These become ‘clusters’
 - We cannot create a spanning tree graph unless everything is connected

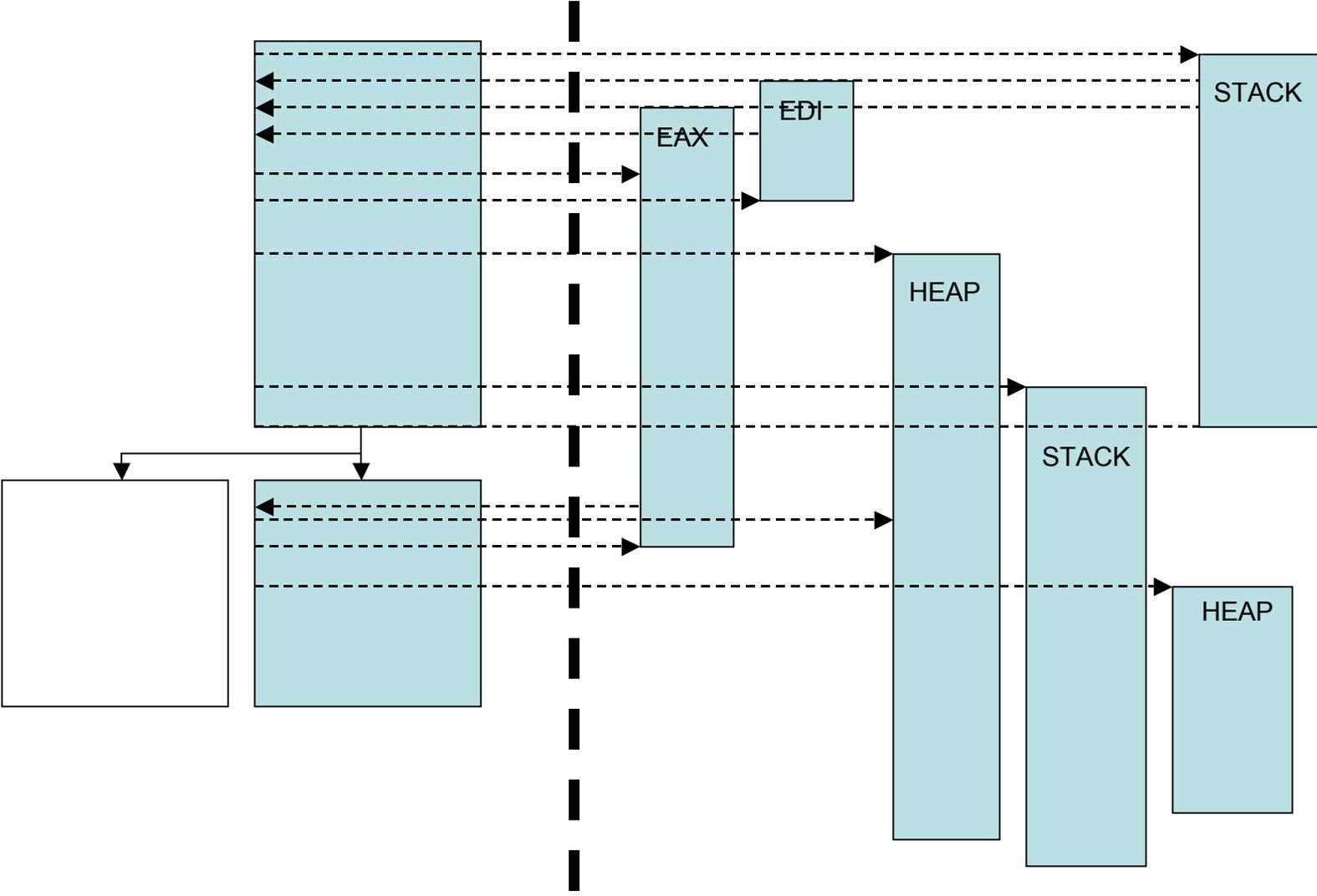
Leap Frogging



Leapfrog with Boron

- Read memory to find all boron strings
- Set memory breakpoints on all these locations
- Locations are typically re-used
- Doesn't always work because memory is cleared after use

Data Flow Analysis



SQL Inject an FTP Server?*

```
C:\>ftp localhost
Connected to GREG-C840.clicktosecure.com.
220-██████████ FTP Server Version 2.6.5 Release 5 - Build 1690
220 service ready
User (GREG-C840.clicktosecure.com:(none)): ffff';DELETE FROM ACCOUNTS WHERE ACCO
UNT_ID = 1;SELECT * FROM ACCOUNTS WHERE NAME = 'ff
331 User name okay, need password.
Password:
530-Database Exception occurred. See server log files for more information
530 closing control connection.
Login failed.
ftp> Invalid command.
ftp>
```

```
Access Data Objects Exception Recorded    Code = 80040e14
Msg: IDispatch error #3092
Source: Microsoft JET Database Engine
Description: Characters found after end of SQL statement.
Tracer: CFTPServer::GetUser
Tue, 01 Apr 2003 (20:09:23) - Closing connection for 127.0.0.1.
```

*this vulnerability is undisclosed, therefore the vendor will not be identified

Buffer Overflow*

- Quote CLNT

ff
ff
ffffffffffffffffffffffffffffffffffff (a few hundred of
these)

* Included because I would feel like a complete loser if I did not reveal at least one buffer overflow in this talk.

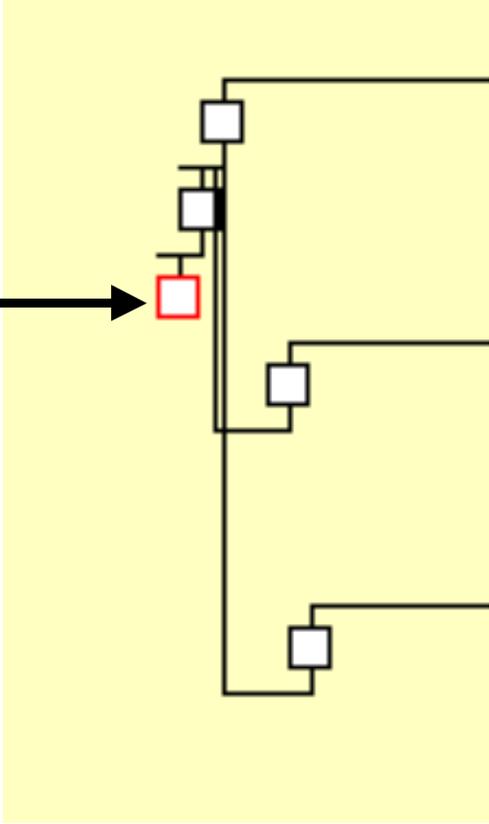
Chapter Five

‘The Process’

ID locations
using static
analysis

Static traces

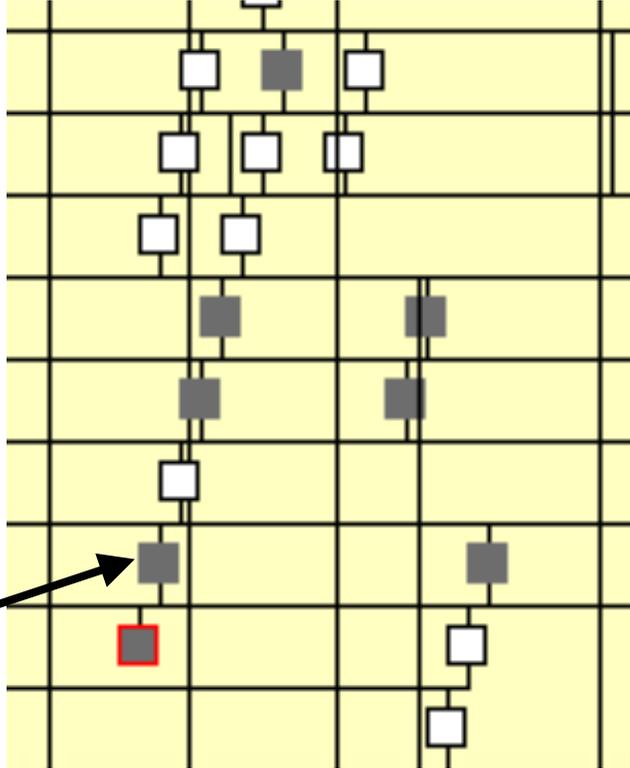
Backtrace
from
potentially
vulnerable API
call or location



ID locations
using static
analysis

Static traces

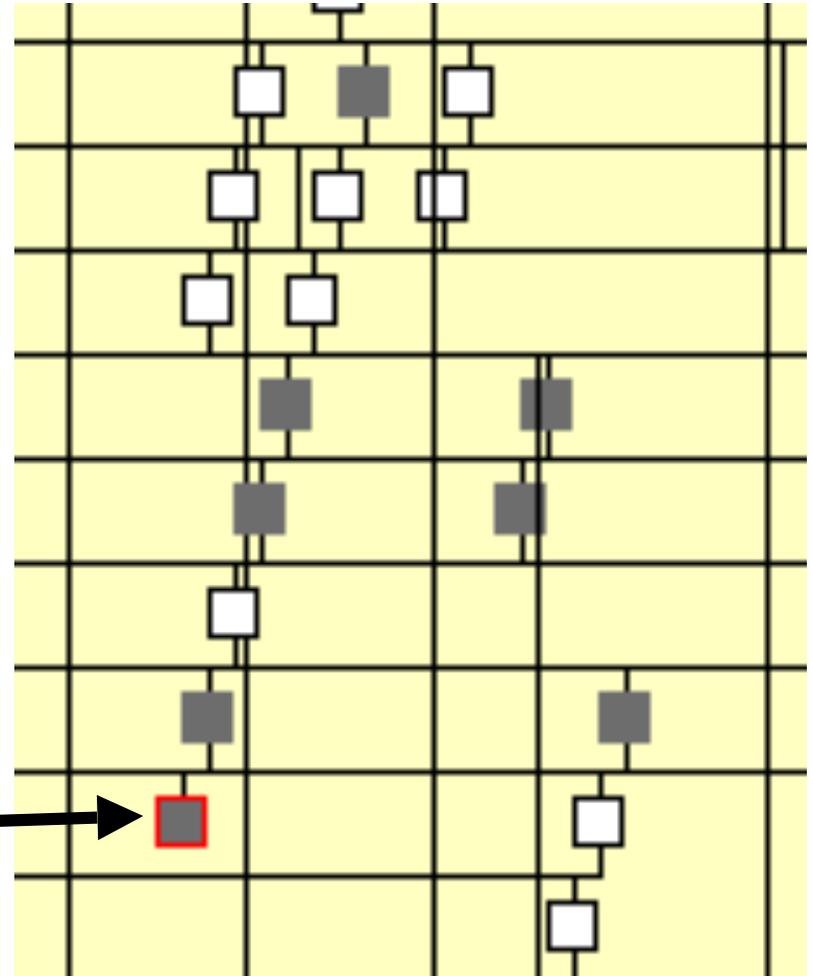
FUZZ



Locations which
are visited are
tagged grey

Work Items

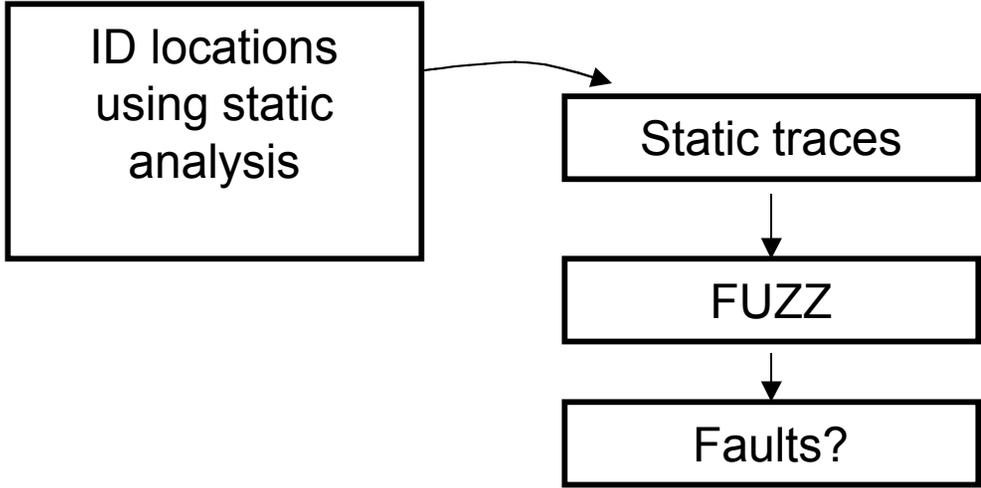
- BP call location 0x00412542 (sprintf)
- xB call location 0x00414DFC (sprintf)
- BP call location 0x0042C0E2 (sprintf)
- xB call location 0x00430750 (sprintf)
- BP SQL query 00444CF1
- BP call location 0x00478D49 (sprintf)
- BP call location 0x0047BB22 (sprintf)
- BP call location 0x0047DA32 (sprintf)
- BP call location 0x00484D27 (sprintf)
- BP call location 0x00484D3B (sprintf)
- BP call location 0x00487103 (sprintf)
- BP call location 0x00487360 (sprintf)



This is a HIT
- This causes
a work item to
be exercised.

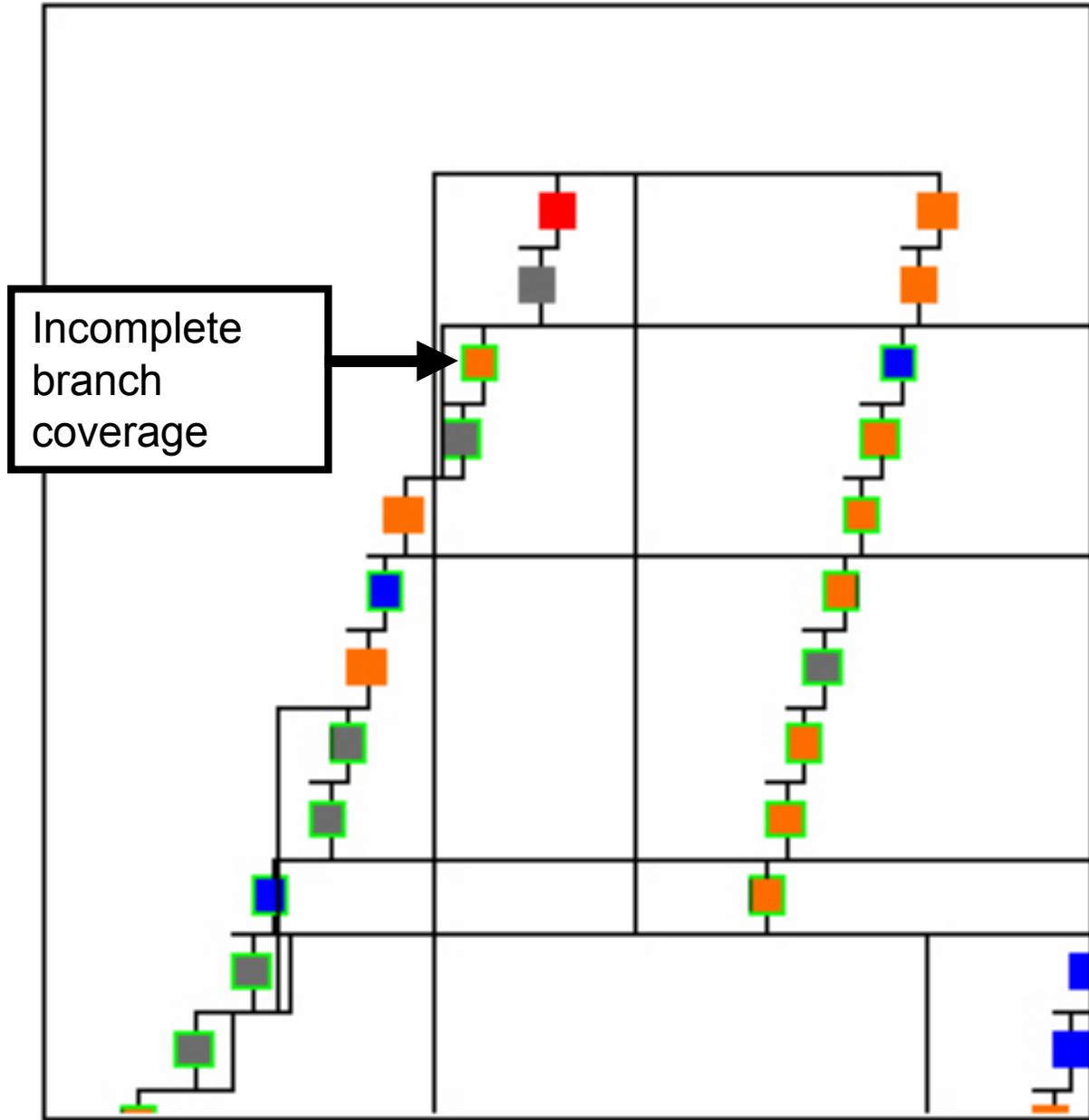
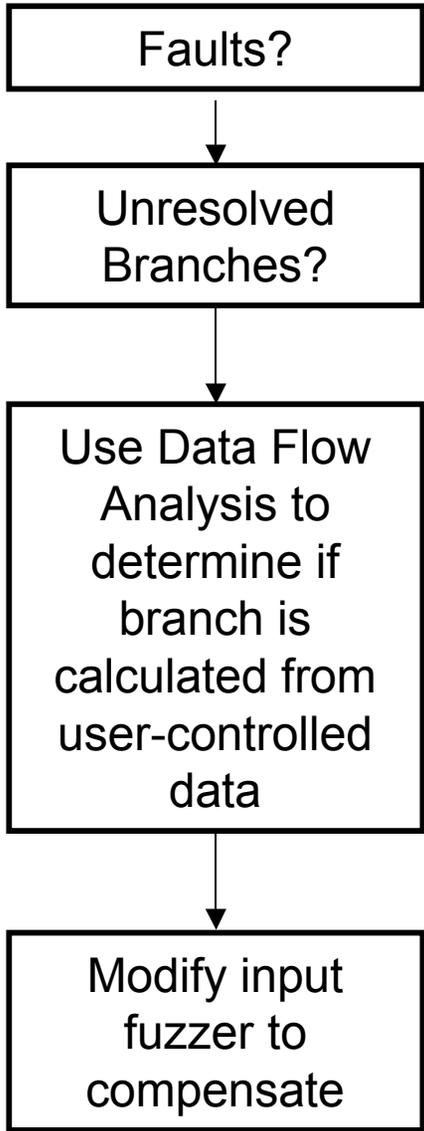
Is user-supplied data used in the suspect call?

Hits	
Time: 12:25:57:257	EAX:08984058(144195672) -> SELECT * FROM ACCOUN
Time: 12:25:57:257	EBX:00B4F0F4(11858164) -> .w. L..
Time: 12:25:57:257	ECX:00000014(20)
Time: 12:25:57:257	EDX:00000014(20)
Time: 12:25:57:257	ESI:00B4F7AC(11859884) -> X@. .k> ...
Time: 12:25:57:257	EDI:0000002A(42)
Time: 12:25:57:257	EBP:004A0604(4851204) -> SELECT * FROM GROUPS
Time: 12:25:57:257	ESP:00B4F0C0(11858112) -> X@. J
Time: 12:25:57:257	+0:08984058(144195672) -> SELECT * FROM ACCOUN
Time: 12:25:57:257	+4:004A0604(4851204) -> SELECT * FROM GROUPS
Time: 12:25:57:257	+8:00B4F0F4(11858164) -> .w. L..
Time: 12:25:57:257	+12:77121644(1997674052) -> .D\$ f.
Time: 12:25:57:257	+16:003E4F50(4083536) -> .5J



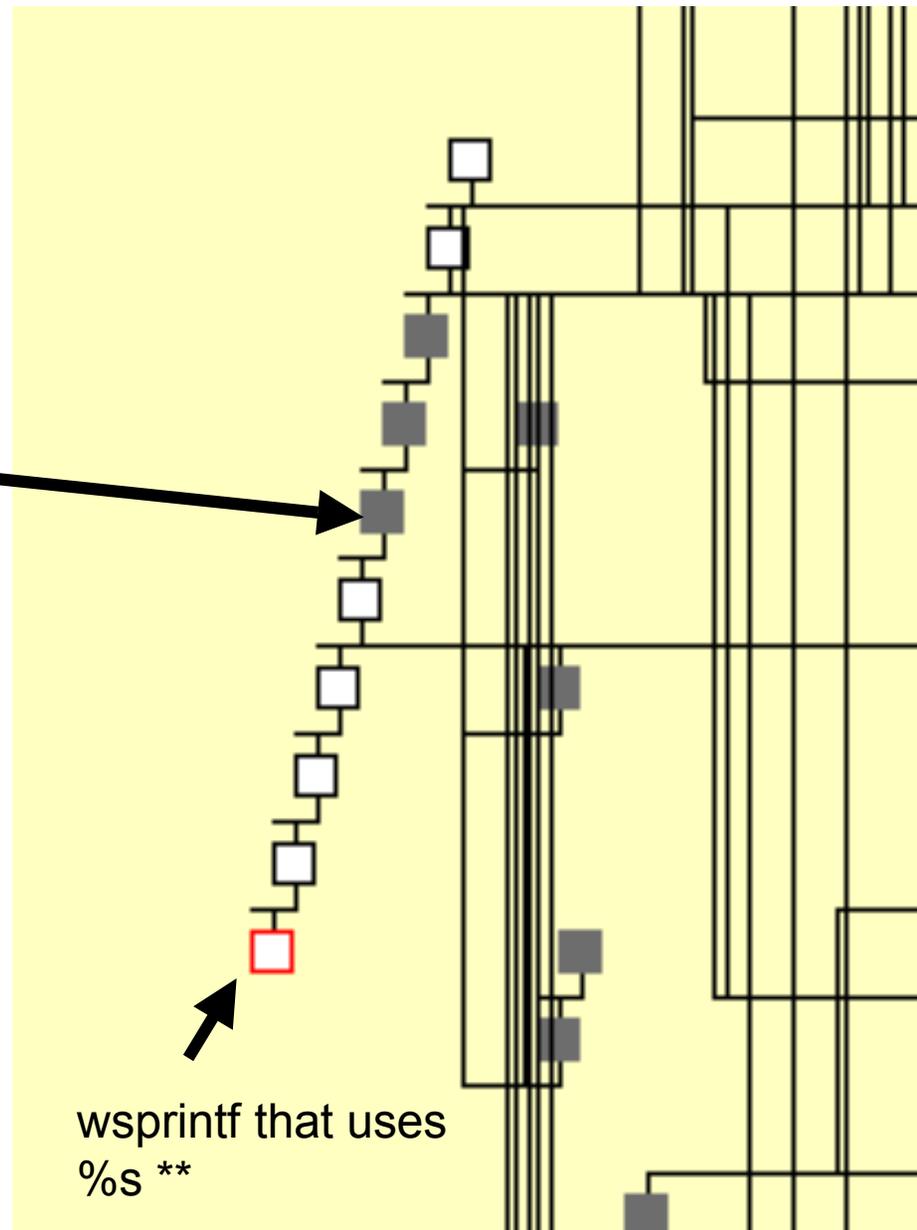
- BP call location 0x00412542 (sprintf)
- xB call location 0x00414DFC (sprintf)
- BP call location 0x0042C0E2 (sprintf)
- xB call location 0x00430750 (sprintf)
- BP SQL query 00444CF1
- BP call location 0x00478D49 (sprintf)
- BP call location 0x00478D49 (sprintf)

```
%d but also handles a SELECT statement which looks mildly interesting.  
0012F7E8 004A2070 ASCII "UPDATE  
SERVERSETTINGS SET  
SITE_SERVER_STATE=%s WHERE ID=1"
```



This location is the nearest fly-by. To solve the problem we must visit this location and determine what data is being used to make the branching decision.

In most cases, the value is not directly controlled by the fuzzer. This means that we must trace back further to determine if the value is calculated from user input. This is both tedious and time consuming.



** this graph generated from commercial proxy server (vendor not revealed)

Conclusion

- There exists a process to connect user-input to potential vulnerabilities
- By tracing data and control flow at runtime, a fuzzer can be tuned to target a location
- Only a certain percentage of those bugs identified statically will be exploitable

Closing Remarks

BugScan is a commercial product that can
be obtained from

www.bugscaninc.com

Closing Remarks

Spike is free and can be obtained from

www.immunitysec.com

Hailstorm is not free, and can be obtained
from

www.cenzic.com

Closing Remarks

- The Tempest debugging system is used internally by HBGary, LLC and is not a commercial product
- Many components of the tempest system are open source and can be obtained for study

www.hbgary.com

Thank You

Greg Hoglund
HBGary, LLC.

hoglund@rootkit.com

www.hbgary.com