Runtime Decompilation

The 'GreyBox' process for Exploiting Software

By: Greg Hoglund

hoglund@hbgary.com

Introduction

- This presentation is an overview of the lab process used by HBGary™ to locate and exploit software bugs
- The value is reduced time investment and formalizing a 'black art'

The Deconstructionist

- Taking a system apart is about uncovering mysteries
- Having secret knowledge attracts the human psyche at the deepest level
- There is an age old battle between those that create systems and those who take them apart
 - i.e, crytoanalysis

Why Exploit Software?

- Exploits are worth money
 - A vulnerability can be worth over \$100K
 - The vendor costs are huge for a public vulnerability
 - An exploit costs less than physical 'bugs'
- Exploits are worth lives
 - An exploit is safer than physical penetration
- Exploits are strategic
 - Disable or control the information systems of your enemies decision cycle

Survivability

- Exploits have a lifetime
- Every use of an exploit has the potential to compromise the asset
- Exploits depend on bugs and your enemy may also find the same bug
- The public may find the same bug
- Once public, many exploits can be protected against or detected via an IDS

In order to maintain a battle advantage, your offensive information capability must include a lab process for finding and exploiting new software bugs

Chapter One

The Bugs!

The Bugs

- Buffer Overflows
 - Lack of bounds checking
 - Arithmetic errors
- Parsing Problems
 - Input filters and normalization
- General State Corruption
- 4. Race Conditions

Buffer Overflows

- Old News, but still most common today
 - Because of speed, most server software is still developed in c/c++
- Will remain common until old compiler technologies are abandoned
 - Strongly typed languages, such as C#, eliminate simple string overflows

Parsing Problems

- Not solved by better compilers
- Solved only by good algorithms
- To eliminate parsing problems requires standardized algorithms
 - Similar to peer review on crypto systems
- This will never happen
 - Parsing problems here to stay

General State Exploits

- States control decisions
- Users can cause state transitions
- Some states are insecure by nature
- State exploits are found by sending commands in the exact order and context to arrive at the insecure state
- Only solved by provably correct systems
 - Humans are never going to build provably correct systems

Race Conditions

- State problems are going to be difficult to measure and control
- When state is managed over many nodes, the problem becomes even harder
- When state must be synchronized among nodes, we have race conditions
 - The problem is compounded greatly
- This is the 'buffer overflow' of the future

Chapter Two

What is the 'GreyBox' process?

GreyBox:

Combining both static analysis and runtime fault injection to maximize coverage of a software programs' state-space. Typically used to detect and isolate fault states in a software system.

White Box

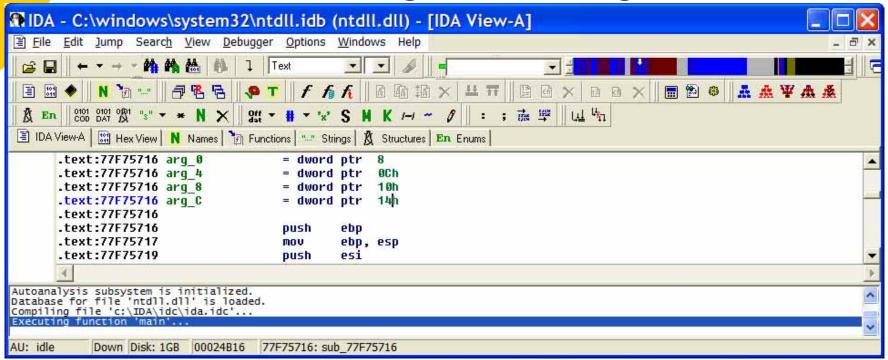
- In theory, operating with full knowledge about the inner workings of the system
 - At best, we only have an approximate understanding of the builder's intent
- White box analysis involves "deadlistings" static disassemblies of the binary
 - Source code is an added advantage, like having really good documentation for the deadlisting
- The software is *not* being executed

First Pass

- When confronted with a new binary, the HBGary team fires the binary through BugScan™
- We obtain a report within minutes to assess whether the programmers use secure coding practices
 - Typically they do not
- We use the BugScan report to prioritize which binaries will be analyzed first
 - Binaries with bad reports are hit first

Manage the Deadlisting

 IDA-Pro allows you to manage and comment a large deadlisting



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 tommorow?
- 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?

Automatic Bug Detection

- The bug must have a defined pattern, it must be schematic in nature
- 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

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, the emulation becomes computationally equivalent to running the program in the first place. How much emulation is enough?

Backtraces reach dead ends

- We backtrace up to 64 steps from a vulnerable function
 - Every branch is exercised
- Back traced cross references can be used to connect input with a code location
 - For example, does a previous function take input from the network?
- Many times a static backtrace deadends
 - Windows message handler

Black Box

 All we see are the outputs from the software – no inner workings

Requires deep protocol knowledge

- o 'Fuzzers':
 - Hailstorm and Spike

Black Box is not stand alone

- Black box testers take FOREVER to complete their input sequences.
- If the program is slow, this compounds the problem
- Amounts to 'brute forcing'
- Finding bugs with pure brute force is mostly luck

Blackbox State

- Typical network software is highly stateful
- A client must be able to maintain a complex state in order to communicate effectively with the target
- Modeling highly stateful clients from scratch is very time consuming and prone to error

Instrumented Clients

- Using a real client program
 eliminates most of the state issues
- You don't need to rebuild the wheel
- Fault injection is inserted *in-transit* by modifying the code within the client program
- The client program becomes a hostile mutant

Fault Injection Clients

- If the protocol is proprietary you have two choices
 - Modify in the middle the packets
 - Only works is protocol is not overly complex and not encrypted – beware of authentication/encryption
 - Instrument proprietary client
 - Requires difficult call-hooking, time consuming

Hooking clients

- Find location where pointer is held in register
- Put breakpoint on this location and modify the given string in memory
 - Cannot BO string w/o corruption
- Or, replace the pointer with another pointer
 - May cause state problems in some clients
- Using 'debugger' technology makes this whole process easier – no EXE patching required

GreyBox

- Combine black-box injection with code analysis
- If you use a program debugger, your 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

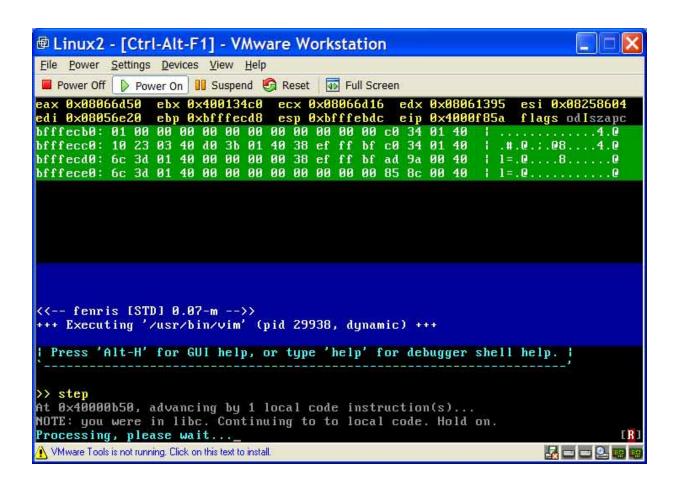
Interactive

- Grey-box testing is an interactive process between a skilled engineer and the target program
- Tools used include SoftIce, OllyDbg, Aegir, Fenris, GDB, Tempest, and the MS-Visual C debugger*

*IDA-Pro has an integrated debugger, but the current version is not evolved enough for industrial level work

Fenris

www.bindview.com



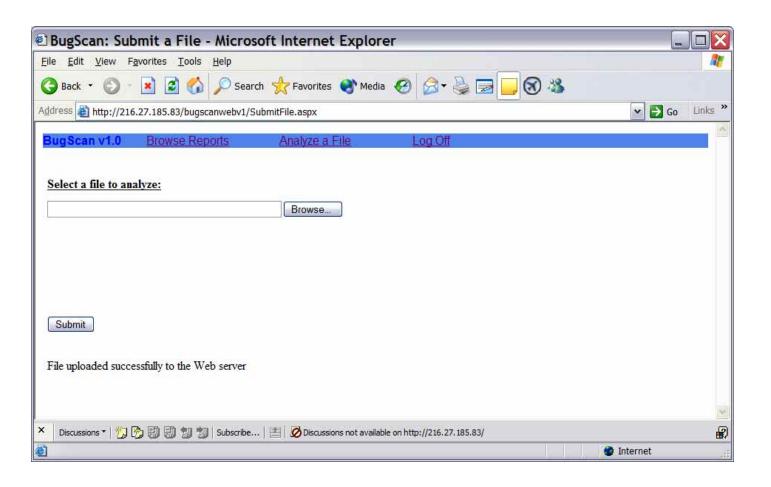
Chapter Three

How to use Bug Scan™

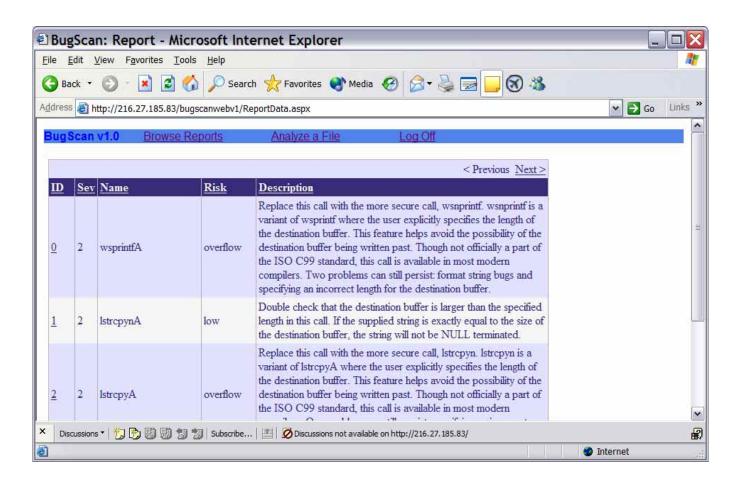
Easy Stuff – Introducing BugScan!

- BugScan is extremely simple to use
- Submit binary and get report
- Report cannot verify is 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



Latest BugScan Reports from the Field

TO BE REVEALED AT CONFERENCE

FREE BUGSCAN for BLACKHAT

 Use this logon to scan any binary, free for blackhat attendees for the next 60 days

HTTP://www.hbgary.com/freeblackhat

More on bugscan

www.bugscaninc.com

info@bugscaninc.com

310-654-8745

Chapter Four

How HBGary Uses GreyBox

Hard Stuff

- Designed for experts
 - Not a product!
- Requires reverse engineering skills not limited to:
 - Runtime debugging
 - Assembly code
 - Protocols
 - Technical knowledge of programming bugs

Introducing TEMPEST

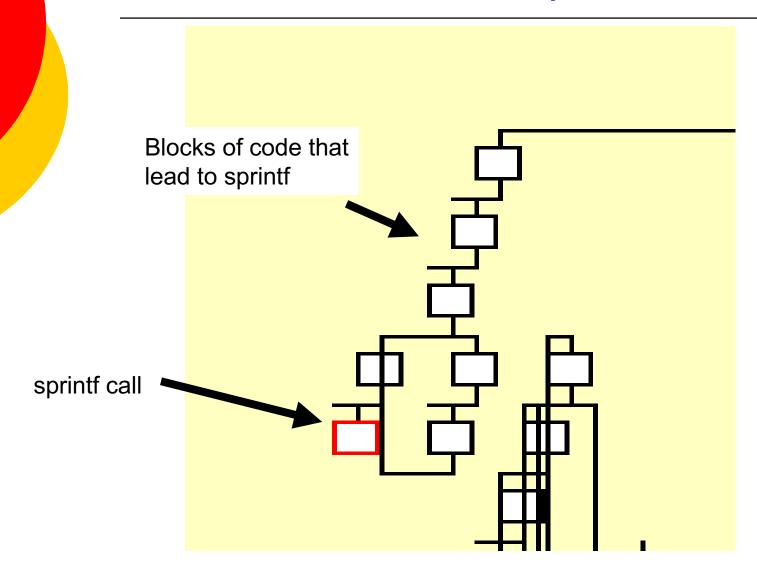


Free technology available for download from www.hbgary.com

TEMPEST

- Connect the inputs with the bugs
- Verify the exploit
- Build a working exploit
- Offensive stance find working injection vectors
- Defines a WORKFLOW

Static backtrace from suspect locations

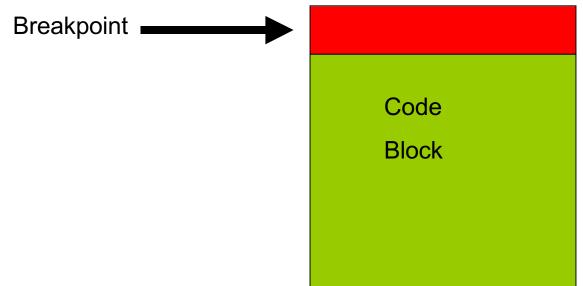


'Self Learning Coverage'

- We start with user-supplied data
- We can detect when decisions are calculated from user input
- We can freeze and restore the program at any point and test new values
- Thus, we can map how usercontrolled values influence state transitions

Location Coverage

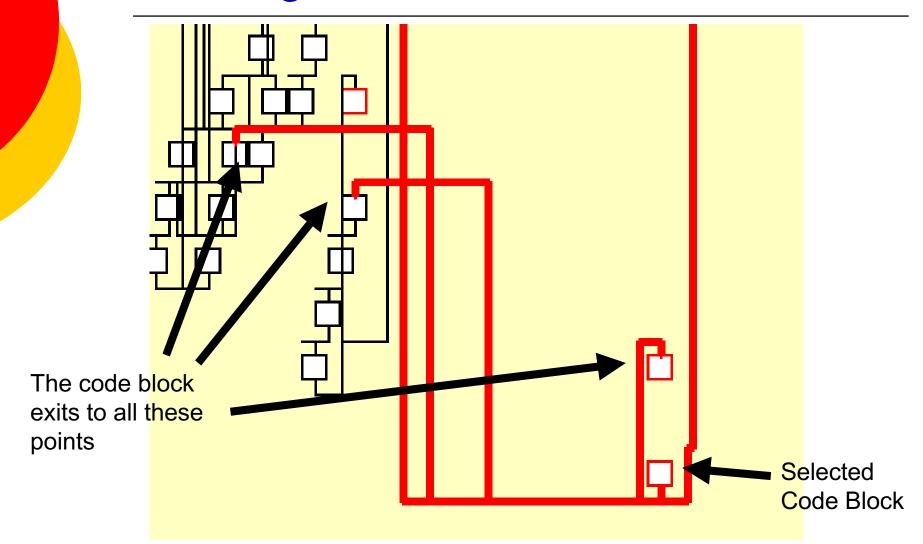
 As program is used, if a code block is visited it will be highlighted 'grey'*



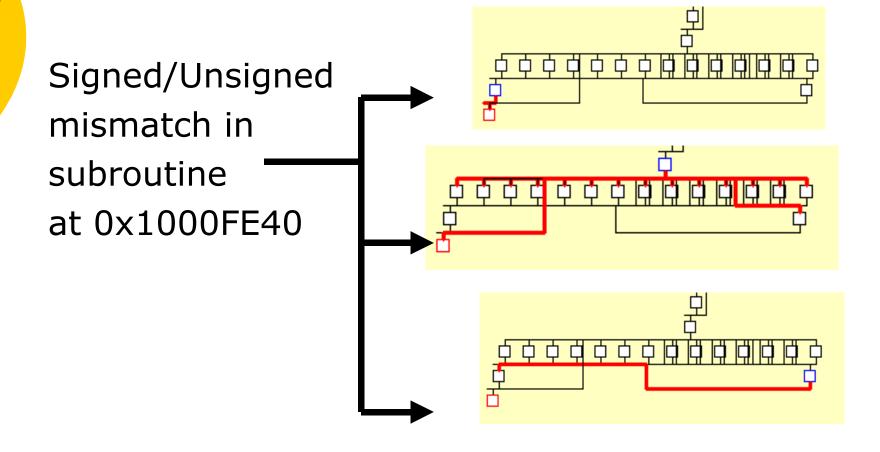
Fly-By's & Drill Downs

- 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

Tracing



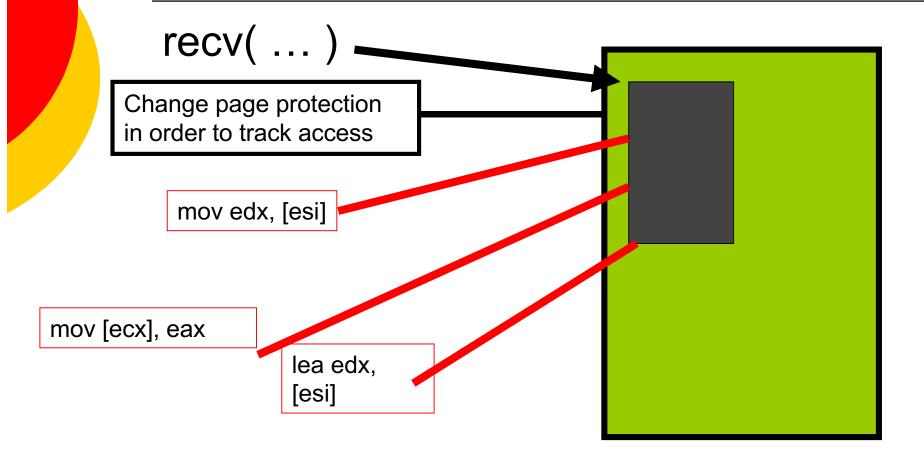
Trillian IRC DLL



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

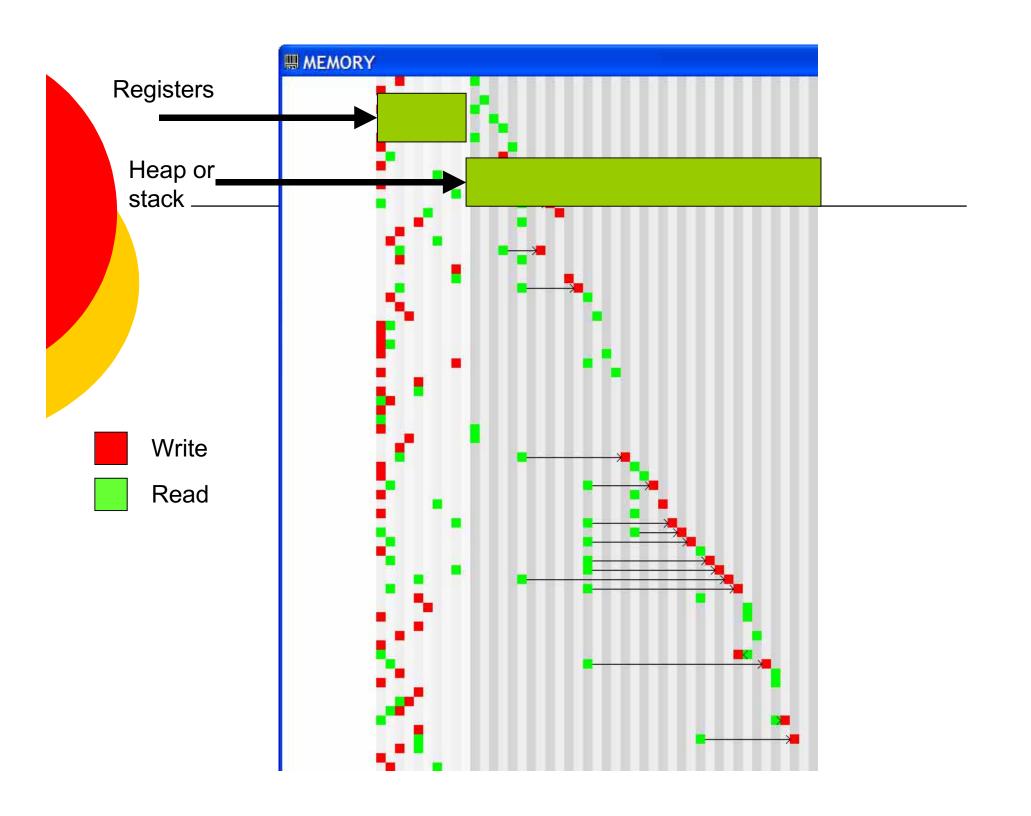
Leap Frogging



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 **STACK** EDI EAX HEAP **STACK** HEAP

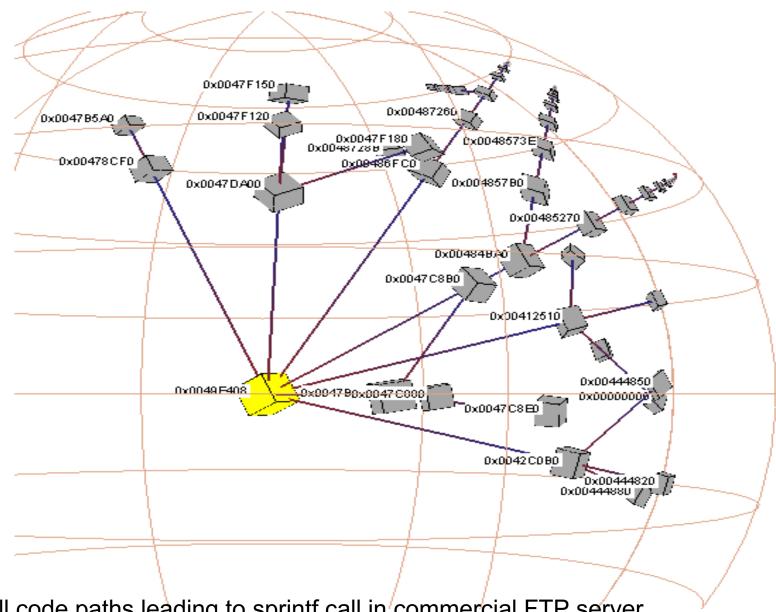


Graphing Problems

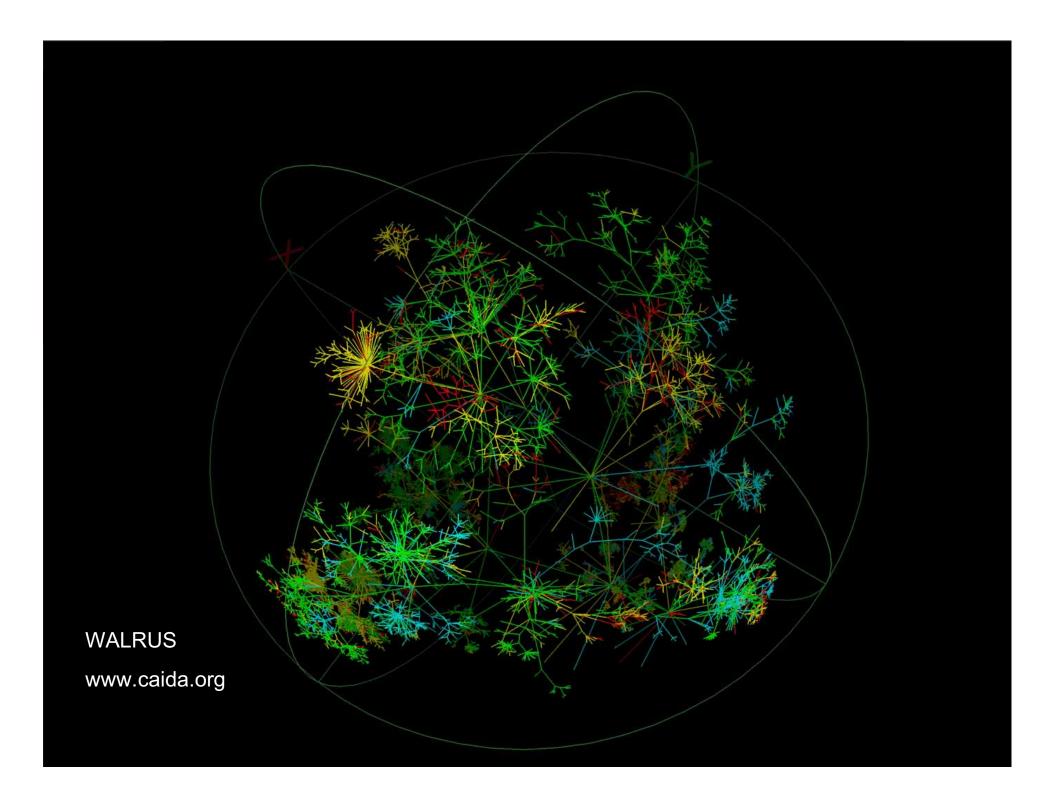
- Graph complexity increases with the number of back traces
- Using tempest on more than a few target points at a time results in a huge, unwieldy graph

Advanced Graphing

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

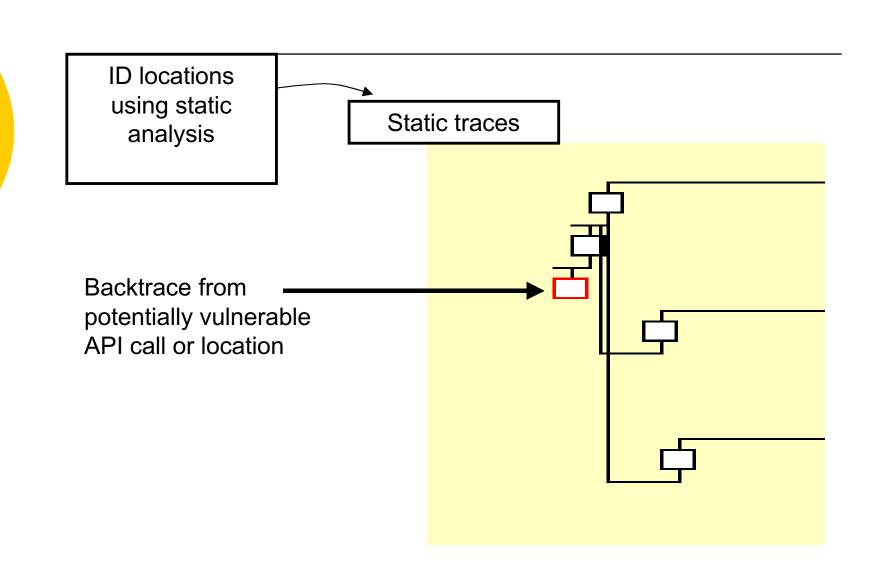


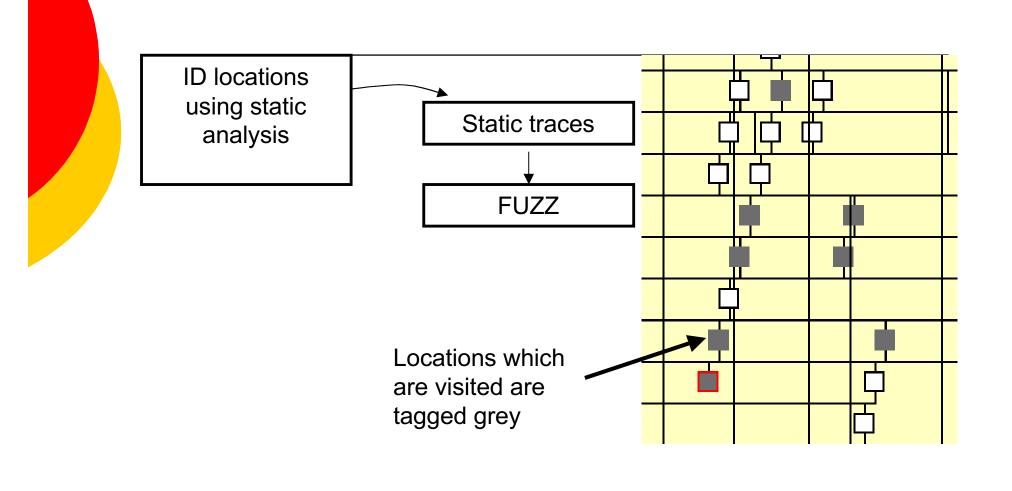
All code paths leading to sprintf call in commercial FTP server, information obtained statically

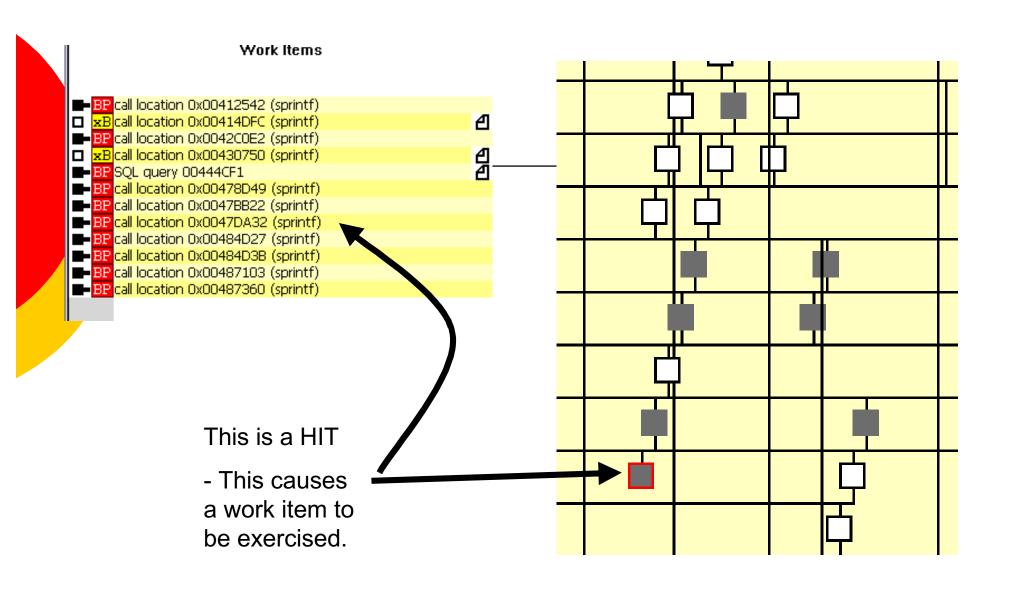


Filtering the set

- 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

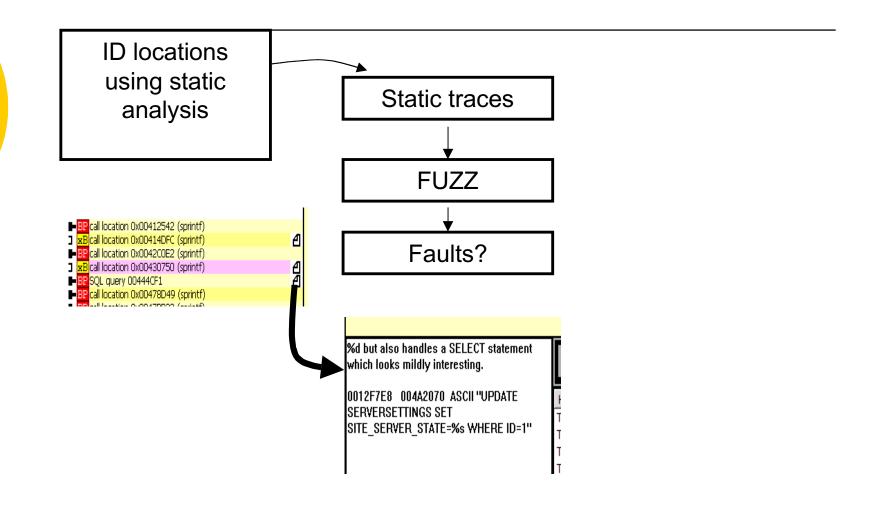


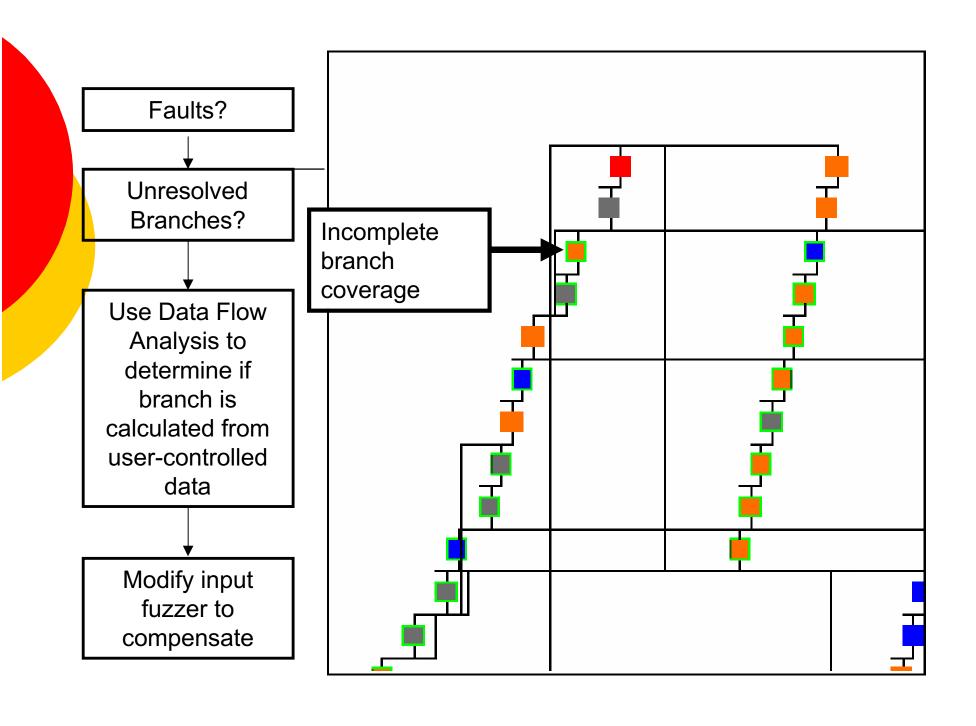


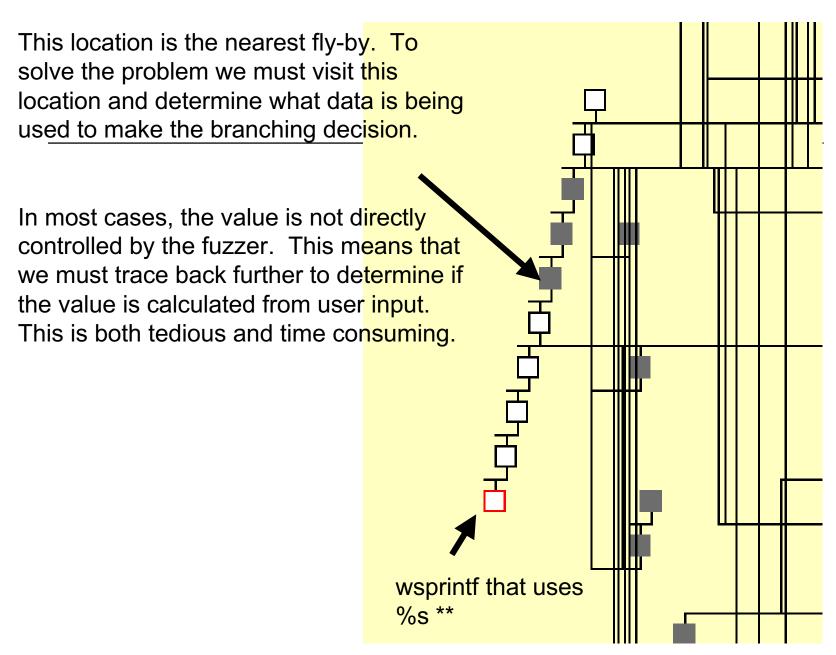


Is user-supplied data used in the suspect call?

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







^{**} 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.hbgary.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 and is not a commerical 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@hbgary.com