The Hacker Strategy

Dave Aitel
dave@immunityinc.com

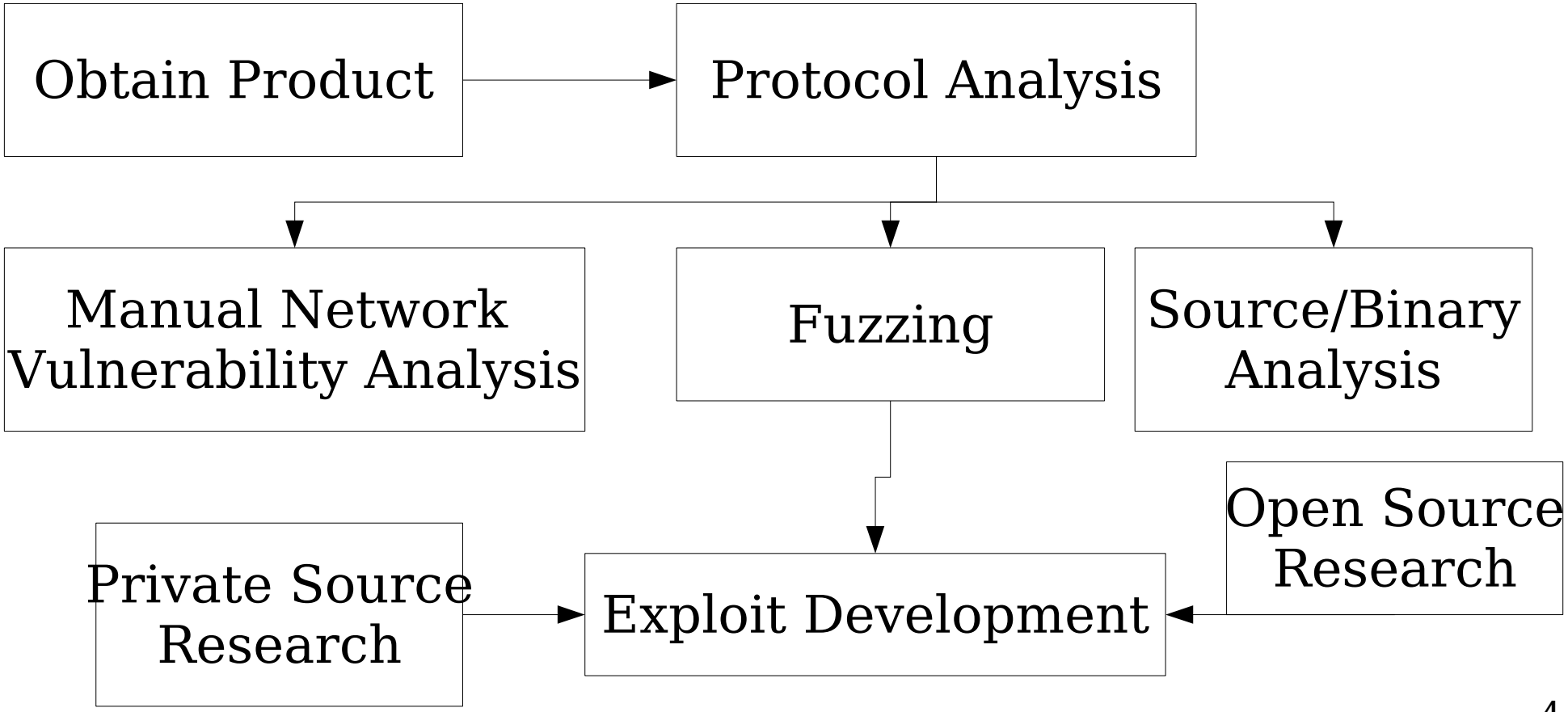
IMMUNITY Security Research

Who am I?

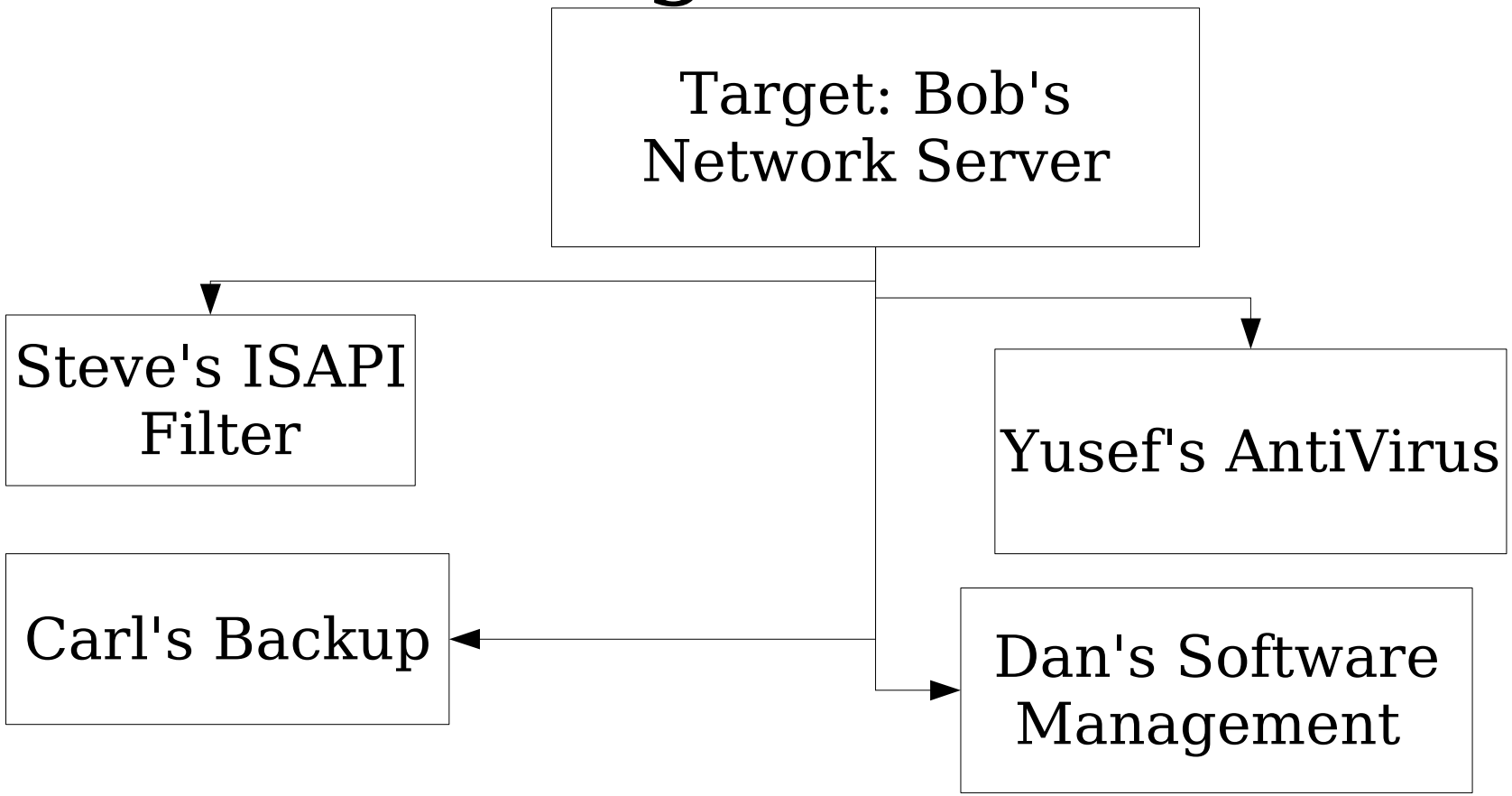
- CTO, Immunity Inc.
- History:
  - NSA->@stake -> Immunity
- Responsible for new product development
  - Vulnerability Sharing Club
  - Immunity CANVAS
  - Immunity Debugger
  - SILICA

Hackers use People, Processes and Technology to obtain a singular goal: Information dominance

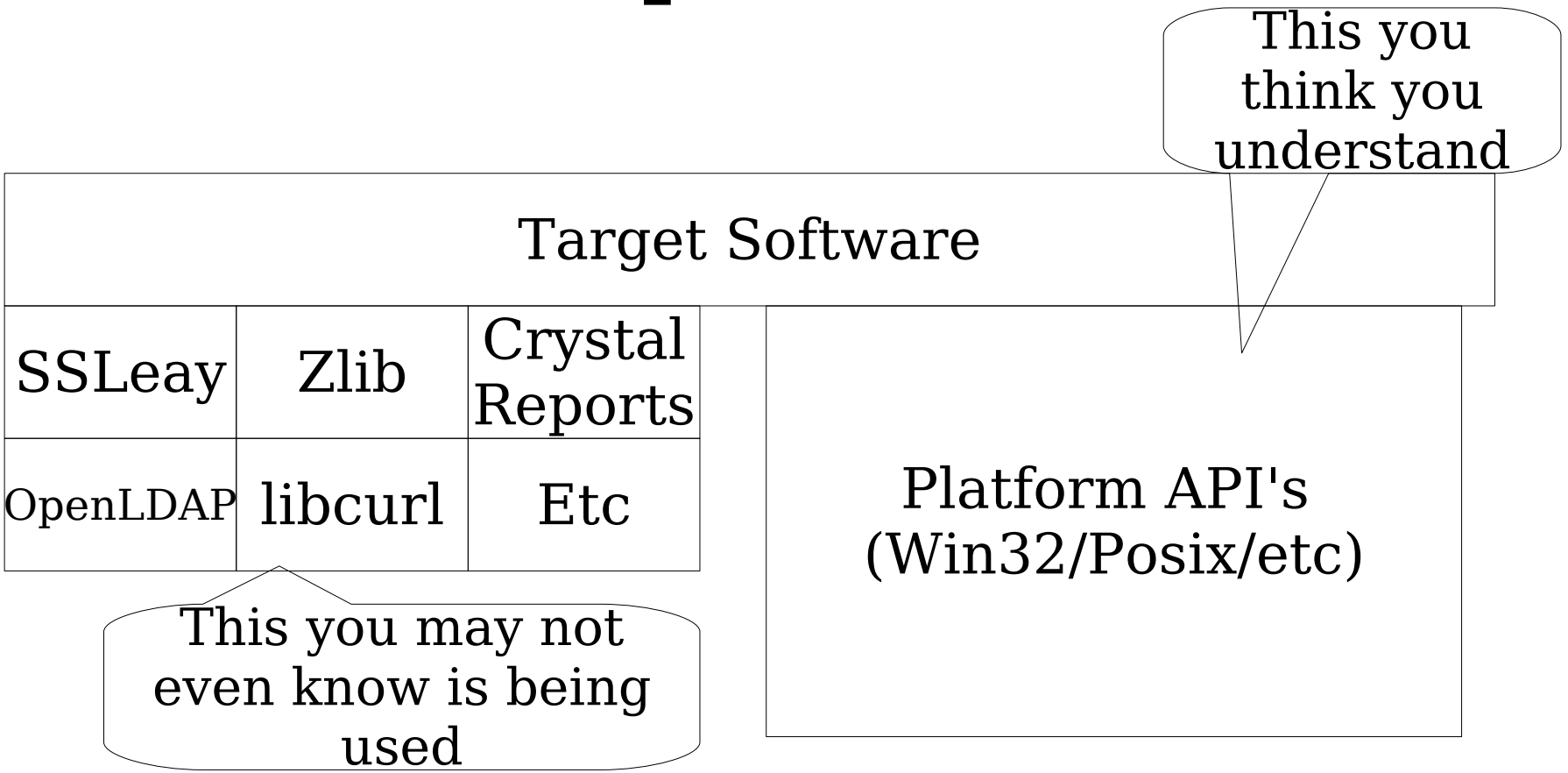
Take a sample product X and attack it remotely



The unseen step: Picking your targets



Third party software is often the problem



Obtaining hardware and software is the hardest step

Subject: [Full-disclosure] scada/plc gear
From: gmaggro <gmaggro@rogers.com>
Sender: full-disclosure-bounces@lists.grok.org.
Date: 01/05/08 14:01
To: Full Disclosure

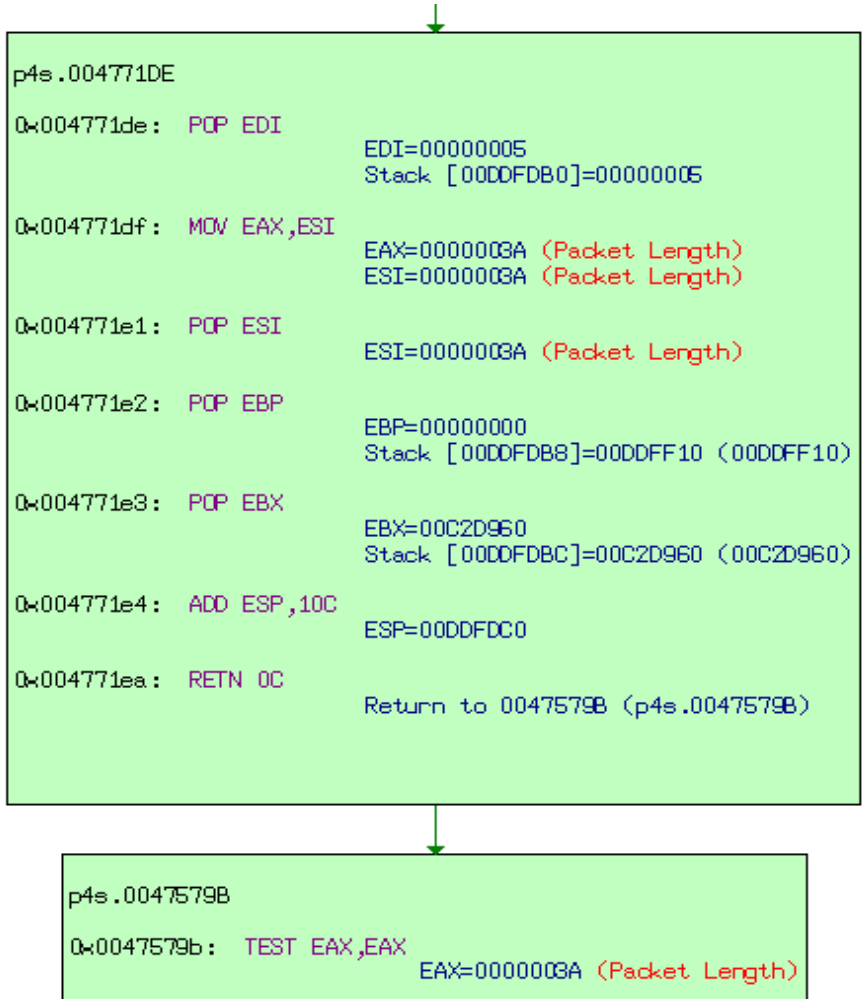
OK, having done some digging a decent little chunk of industrial automation gear has started coming my way; 1 of 6 pieces. All totaled roughly under $1000. Small standalone stuff for now; the shipping on populated PLC chassis like SLC-500 stuff is problematic.

If people have specific technical questions, want a script run agains piece of gear or a custom protocol capture done I will entertain such requests. I am also willing to open the cases and pick up the solderi iron, attempt rom/firmware dumps, etc.

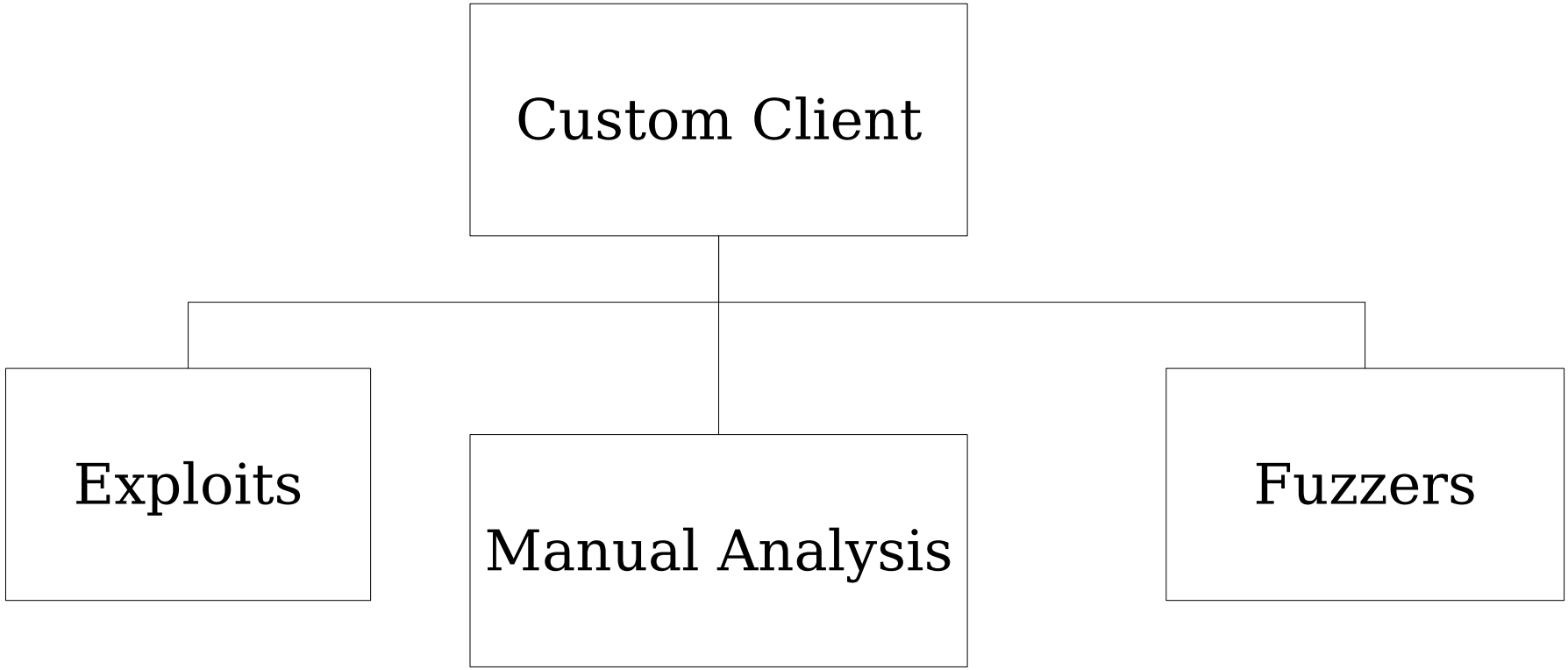
Are there any particular tests or tools someone would like me to work into my routine right from the start?

Hardware piece #1 is a Kohler Power Systems modbus/ethernet converter, pn# GM40165.

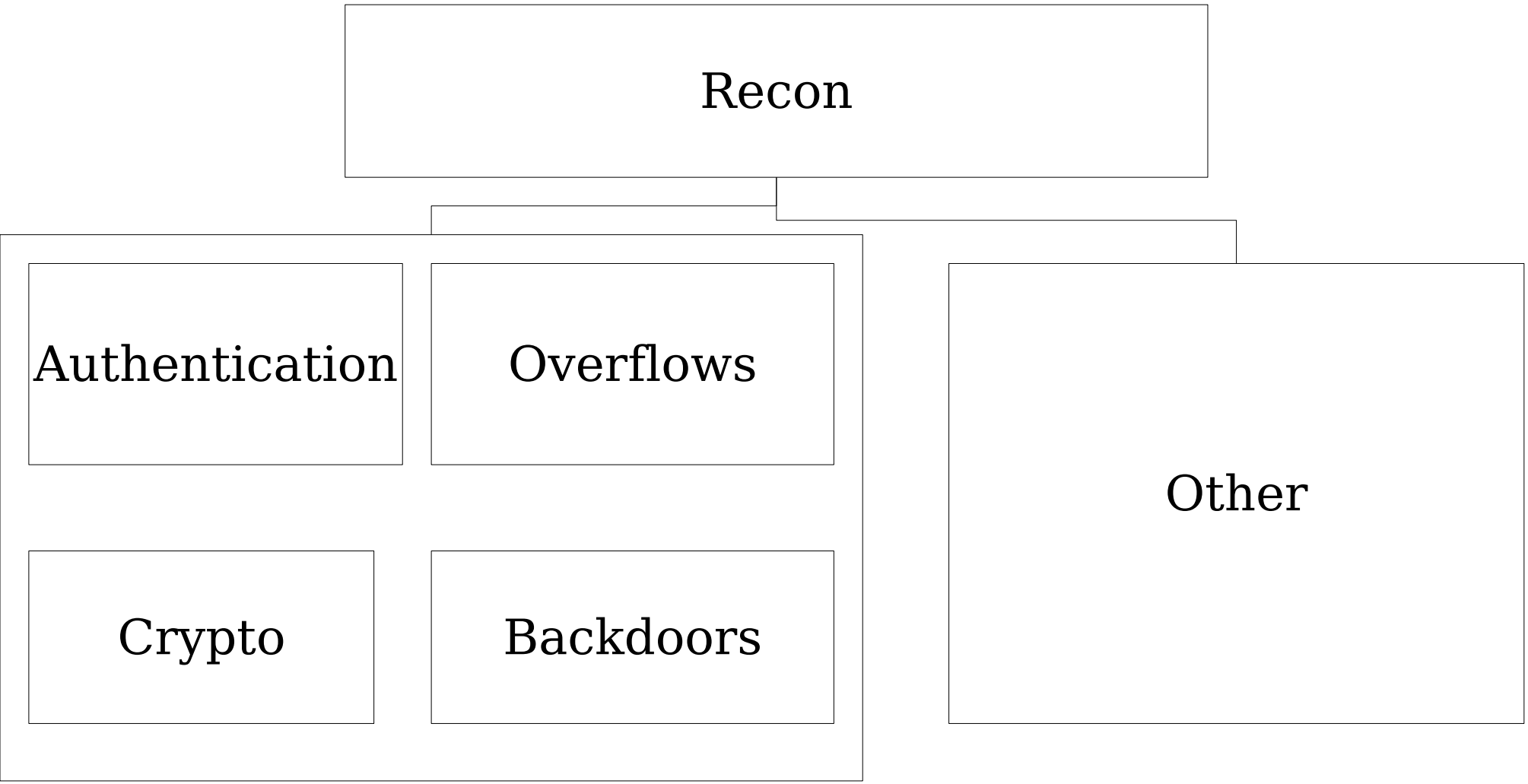
Protocol Analysis is often quite easy



Hackers always create custom client protocol libraries

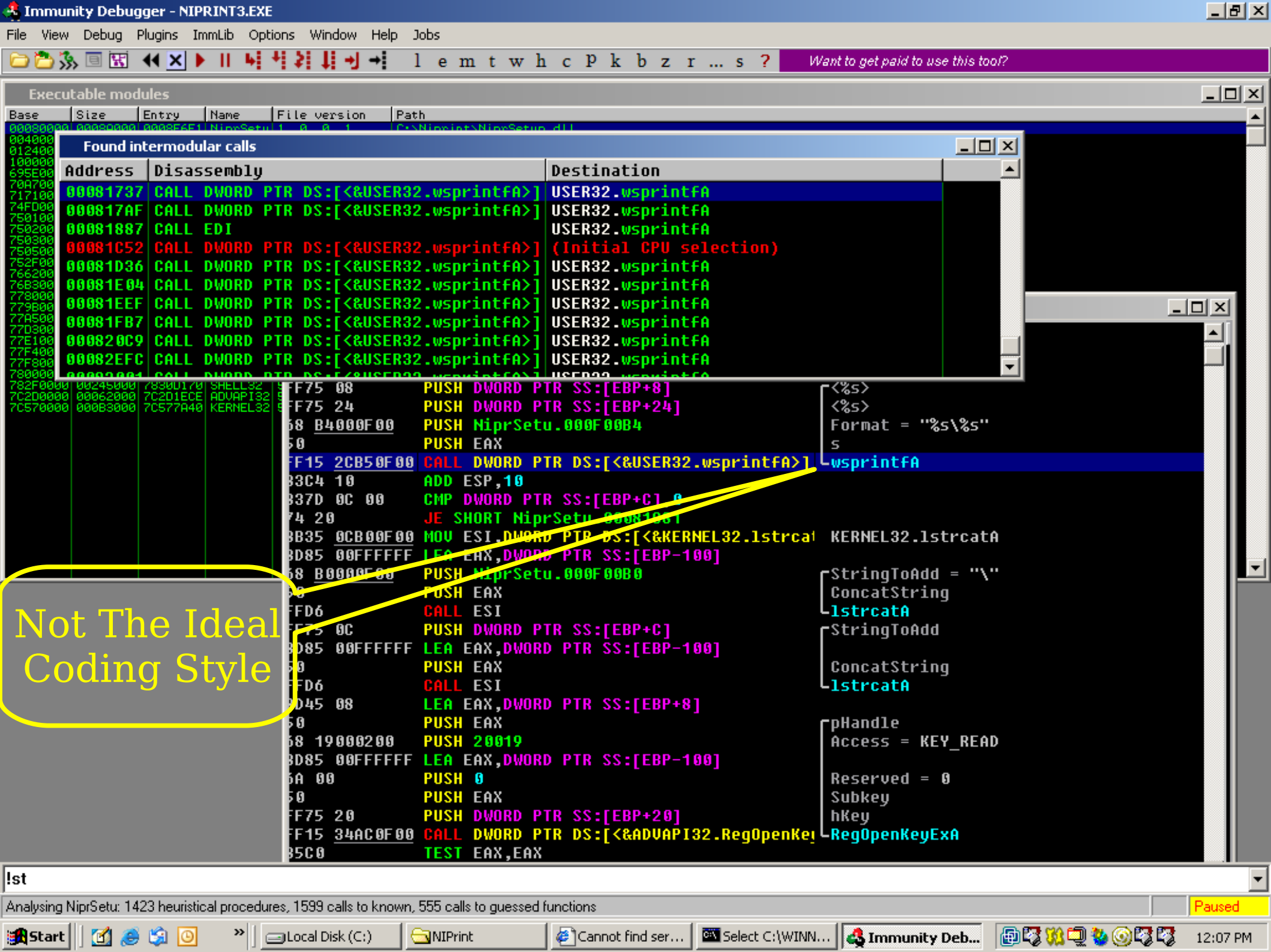


Manual Security Analysis



Basic Binary Analysis For Fun and Profit

- Look at all DLL's loaded by the application and all exposed API's and text strings
- Trace from all incoming packets to get a feel for the structure of the application
- Look for dangerous code patterns
- Conduct code coverage review


Immunity Debugger - NIPRINT3.EXE
File View Debug Plugins ImmLib Options Window Help Jobs
Want to get paid to use this tool?
Executable modules
Base Size Entry Name File version Path
Found intermodular calls
Address Disassembly Destination
00081737 CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
000817AF CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
00081887 CALL EDI USER32.wsprintfA
00081C52 CALL DWORD PTR DS:[<&USER32.wsprintfA>] (Initial CPU selection)
00081D36 CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
00081E04 CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
00081EEF CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
00081FB7 CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
000820C9 CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
00082EFC CALL DWORD PTR DS:[<&USER32.wsprintfA>] USER32.wsprintfA
PUSH DWORD PTR SS:[EBP+8] <%s>
PUSH DWORD PTR SS:[EBP+24] <%s>
PUSH NiprSetu.000F00B4 Format = "%s\%s"
PUSH EAX s
CALL DWORD PTR DS:[<&USER32.wsprintfA>] wsprintfA
ADD ESP,10
CMP DWORD PTR SS:[EBP+C],0
JE SHORT NiprSetu.000B1061
MOV ESI,DWORD PTR DS:[<&KERNEL32.lstrca KERNEL32.lstrcatA
LEA EAX,DWORD PTR SS:[EBP-100]
PUSH NiprSetu.000F00B0 StringToAdd = "\"
PUSH EAX ConcatString
CALL ESI lstrcatA
PUSH DWORD PTR SS:[EBP+C] StringToAdd
LEA EAX,DWORD PTR SS:[EBP-100]
PUSH EAX ConcatString
CALL ESI lstrcatA
LEA EAX,DWORD PTR SS:[EBP+8]
PUSH EAX pHandle
PUSH 20019 Access = KEY_READ
LEA EAX,DWORD PTR SS:[EBP-100]
PUSH 0 Reserved = 0
PUSH EAX Subkey
PUSH DWORD PTR SS:[EBP+20] hKey
CALL DWORD PTR DS:[<&ADVAPI32.RegOpenKe RegOpenKeyExA
TEST EAX,EAX
Not The Ideal Coding Style
!st
Analysing NiprSetu: 1423 heuristical procedures, 1599 calls to known, 555 calls to guessed functions
Paused
Start
Local Disk (C:)
NIPrint
Cannot find ser...
Select C:\WINN...
Immunity Deb...
12:07 PM

What you can find in 1 hour of binary analysis

- Basic data flow from the network
- Coding style (the use of bad API's, f.e.)
- Simple backdoors (“DEBUG” string in command list, etc)
- Potential vulnerabilities

One Week of Binary Analysis should get you at least one good vulnerability

- But will probably get you several exploitable bugs, and potentially an exploit as well
- Real binary analysis is almost never just static analysis
  - Which is why automated static analyzers are at a severe disadvantage from a human
- This data will feed quite well into your fuzzer

One month of binary analysis will get you a vulnerability no one else will ever find

- Defeating the automated systems such as Prefix/Prefast, the SDL and SafeSEH+NX +ASDL may require this amount of effort
- A lot of what you will do is build custom binary analysis scripts and protocol libraries
- Vulnerabilities no one else will ever have are extremely useful

What binary analysis is and is not

- In its most advanced form, you transform the program into another kind of program or equation and "solve" it to find vulnerabilities
- Most people scan for code patterns or have code scanning for code patterns
- Finding some bug classes is insanely hard this way

Source Code Analysis

- Not as hard as you think from a hacker's perspective
  - Auditing entire Solaris source tree for one bug can be done in a morning
  - Doing intense study of some part of the Linux kernel can take several weeks
- http://taossa.com/

Hackers do have the source code

- Maintaining global information dominance means that source code to almost every product is available to a skilled hacker group
- This puts them at an immediate advantage over security teams
- They also have a tendency to work at software vendors

Automated source code analyzers don't solve the problem

- High false positive rate
- No ability to read and understand comments
  - Can't prioritize
  - Can't follow unstated data flow
- Only find the simple bug-classes, such as strcpy()
- Microsoft has the world's best source code analyzers – it helps, but it's no solution

On Tools

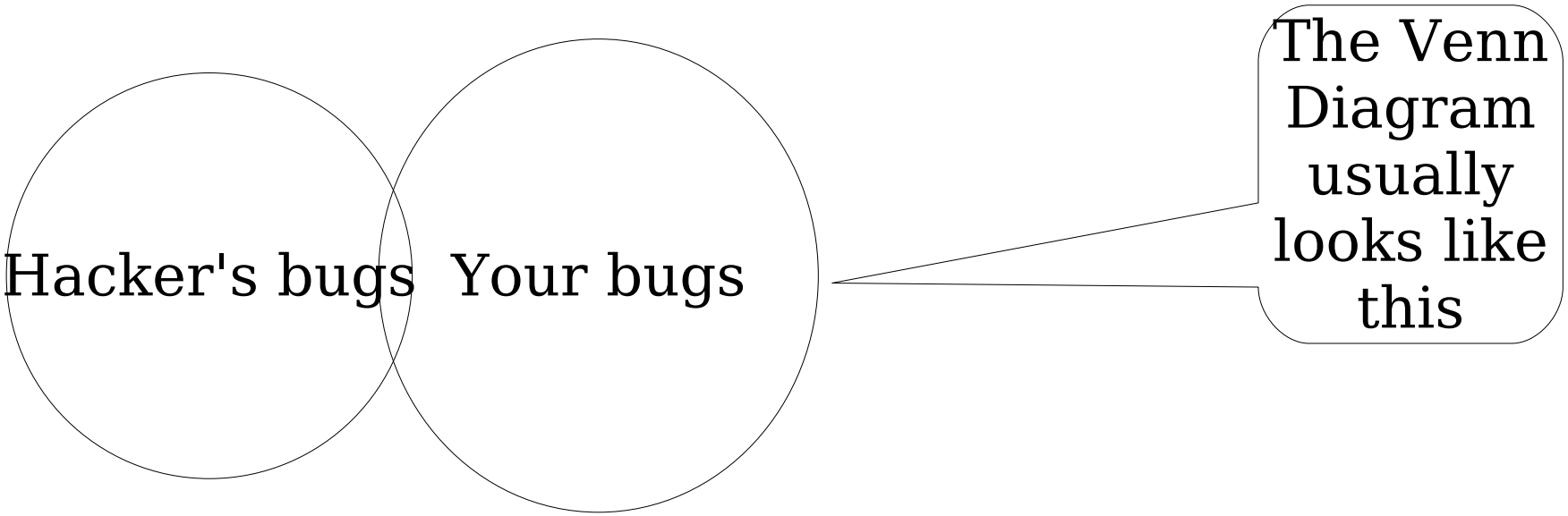
Tools are very useful, we build a lot of tools, and use them all the time here at Microsoft. Some of those tools have found their way into our SDKs and Visual Studio so our customers can use them too. But I would never claim that these tools make code "free of security defects." - Michael Howard (Microsoft SWI)

The defensive side

- Manual analysis
  - Burns out programmers quickly
- Secure Software Design Programs such as Microsoft's threat modeling work to some degree
- Moving to a more secure platform provides the largest benefit

How to build a fuzzer that finds bugs you care about

Your fuzzer and another hacker's fuzzer will not find all the same bugs!



What kind of fuzzer to write?

- I prefer block based
  - Use Python (everyone does)
  - Sulley is a good option
  - SPIKE 3.0
  - Peach
  - etc

Fuzzing is a many year process

- For each vulnerability that comes out, make sure your fuzzer can find it, then abstract it a bit more
- There's two basic things you need to add
  - Transformations
    - A->AAAAAAA…AAAAAA
  - Syntax patterns
    - GET /<fuzz string> HTTP/1.0\r\n\r\n

Myth: Fuzzing only catches low hanging fruit

- Fuzzing can catch many vulnerabilities that are hard to see from the naked eye or from static analysis
  - DTLogin Arbitrary Free, is one example
- Off by ones
- Race conditions

Looking at emergent behaviours in the hacker community from small to large

Things you can't see that no doubt exist

- “Vendor Management” Teams
- X.25 Attack Research was very popular in the late 90's and remains so to this day
- SCADA is certainly on everyone's radar

Hackers maintain a pipeline of things:

- What protocols are most buggy that no one else is looking at
- Bug classes that are hard to scan for by automated technologies
- Bugs themselves
- Exploitation techniques

0days are a hacker obsession

- An 0day is a vulnerability that is not publicly known
  - IDS/IPS cannot find them
  - Can your forensics team figure them out?
- Modern 0days often combine multiple attack vectors and vulnerabilities into one exploit
  - Many of these are used only once on high value targets

As of June 16 2007:

Real-world 0day Statistics

Average 0day lifetime: 348 days

Shortest life: 99 days

Longest life: 1080 (3 years)

The Market Always Wins: 0day is for sale. Deal with it.

- Tippingpoint
- Eeye
- Gleg.net
- Dsquare
- Idefense
- Digital Armaments
- WabiSabiLabi
- Breakingpoint
- etc

Classes of Vulnerabilities

- The classic example is the format string bug
  - printf(user_supplied_string,args);
  - Easy to scan for with automatic tools or compiler options
  - Commonly available in code in 2000
  - Now an extinct species

Vulnerability Classes you know about

- Stack/Heap overflows
- Format Strings
- Race conditions
- Uninitialized variable problems
- Integer overflows and indexing problems

Vulnerability classes you don't know about

- Race conditions
- Sophisticated timing attacks
- Extremely complex multi-vector overflows
- Kernel attacks
- Lots of vulnerabilities in hardware you've never seen attacked publicly

Now: Defeating Patching, IDS, Anti-Virus, etc.

- Be faster to attack than the defender can deploy patches
  - Attack frameworks, better debuggers
- Attack with vulnerabilities that are unknown (0days)
  - New bug classes, better debuggers, new exploit techniques

The Future

- Look for more embedded system attacks
- Look for more interesting bug classes
- Vista/Windows 7 – not the answer
- Hacker Teamwork

Thank you for your time

Contact us at:

admin@immunityinc.com

IMMUNITY

Security Research Team