The Hacker Strategy

Dave Aitel
dave@immunityinc.com
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

- Obtain Product
- Protocol Analysis
  - Manual Network Vulnerability Analysis
  - Fuzzing
  - Source/Binary Analysis
    - Open Source Research
- Private Source Research
- Exploit Development
The unseen step: Picking your targets

- Target: Bob's Network Server
  - Steve's ISAPI Filter
  - Carl's Backup
  - Yusef's AntiVirus
  - Dan's Software Management
Third party software is often the problem

<table>
<thead>
<tr>
<th>Target Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLeay</td>
</tr>
<tr>
<td>OpenLDAP</td>
</tr>
</tbody>
</table>

This you may not even know is being used

This you think you understand

Platform API's (Win32/Posix/etc)
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 against 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 soldering 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

```
0x004771de: POP EDI
            EDI=00000000
            Stack [00DFFDB0]=00000000

0x004771df: MOV EAX,ESI
            EAX=00000034 (Packet Length)
            ESI=00000034 (Packet Length)

0x004771e1: POP ESI
            ESI=00000034 (Packet Length)

0x004771e2: POP EBX
            EBX=00000000
            Stack [00DFFDB0]=00DDFF10 (00DDFF10)

0x004771e3: POP EBX
            EBX=00C20960
            Stack [00DFFDB0]=00C20960 (00C20960)

0x004771e4: ADD ESP,10C
            ESP=00DFFDC0

0x004771ea: RETN 0C
            Return to 0047575B (p+0.0047575B)

0x004771eb: TEST EAX,EAX
            EAX=0000003A (Packet Length)
```
Hackers always create custom client protocol libraries

- Custom Client
  - Exploits
  - Manual Analysis
  - Fuzzers
Manual Security Analysis

- Recon
  - Authentication
  - Overflows
  - Crypto
  - Backdoors
- Other
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
**Not The Ideal Coding Style**

---

<table>
<thead>
<tr>
<th>Address</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001737</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>000017AF</td>
<td>CALL DWORD PTR DS:[&lt;&amp;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>00001887</td>
<td>CALL EDI</td>
</tr>
<tr>
<td>00001892</td>
<td>CALL DWORD PTR DS:[&lt;&amp;USER32.wsprintf&gt;] (initial CPU selection)</td>
</tr>
<tr>
<td>00001898</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>000018E0</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>0000191F</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>00001972</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>000019C0</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
<tr>
<td>00001A80</td>
<td>CALL DWORD PTR DS:[&lt;USER32.wsprintf&gt;] USER32.wsprintfA</td>
</tr>
</tbody>
</table>

---

```
F75 08 PUSH DWORD PTR SS:[EBP+8]                      
F75 24 PUSH DWORD PTR SS:[EBP+24]                     
88 B480F000 PUSH NiprSetu.000F0084                   
90 PUSH EAX                                        
F15 2CB50F00 CALL DWORD PTR DS:[<USER32.wsprintf>]    
  USER32.wsprintfA                                 
3C4 10 ADD ESP,10                                   
37D 0C 00 CMP DWORD PTR SS:[EBP+C] 0                 
44 20 JE SHORT NiprSetu.000F0001                    
B35 0CB00F00 MOV ESI,DWORD PTR DS:[<KERNEL32.1strcat>]
  KERNEL32.1strcatA                                
D85 00FF00FF LEA EAX,DWORD PTR SS:[EBP-100]         
88 B800F00 PUSH NiprSetu.000F0080                   
90 PUSH EAX                                        
FD6 CALL ESI                                       
F75 0C PUSH DWORD PTR SS:[EBP+C]                     
D85 00FF00FF LEA EAX,DWORD PTR SS:[EBP-100]         
90 PUSH EAX                                        
F6D CALL ESI                                       
D45 08 LEA EAX,DWORD PTR SS:[EBP+8]                 
90 PUSH EAX                                        
88 19002000 PUSH 20019                              
D85 00FF00FF LEA EAX,DWORD PTR SS:[EBP-100]         
90 PUSH 0                                          
90 PUSH EAX                                        
F75 20 PUSH DWORD PTR SS:[EBP+20]                    
F15 34AC0F00 CALL DWORD PTR DS:[<ADVAPI32.RegOpenKe]
  RegOpenKeyExA                                    
5C0  TEST EAX,EAX                                   
```
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!

The Venn Diagram usually looks like this
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->AAAAAAAA...AAAAAAA
  - Syntax patterns
    - \texttt{GET \langle fuzz string\rangle 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

Security Research Team