Instrumenting Point-of-Sale Malware

A Case Study in Communicating Malware Analysis More Effectively

Wesley McGrew
Assistant Research Professor
Mississippi State University
Department of Computer Science & Engineering
Distributed Analytics and Security Institute
Introduction

• The pragmatic and unapologetic offensive security guy

• Breaking things

• Reversing things

• Mississippi State University - NSA CAE Cyber Ops

• Enjoying my fourth year speaking at DEF CON
The Plan

• In general:
  • Adopt better practices in describing and demonstrating malware capabilities
  • Proposal to supplement written analyses with illustration that uses the malware itself

• What we’ll spend a good chunk of today’s session doing:
  • Showing off some cool instrumented POS malware
  • Talk about how you can do the same
Scientific Method
(the really important bits)

- Reproducibility

- Reasons:
  - Verifying results
  - Starting new analysis where old analysis left off
  - Education of new reverse engineering specialists
  - IOC consumers vs. fellow analysts as an audience
What’s often missing?

- Sample info
- Hashes
- Availability
- Procedure
- Subverting malware-specific countermeasures
- Context
- Redacted info on compromised hosts and C2 hosts
- Internal points of reference
- Addresses of functionality/data being discussed
Devil’s Advocate: Why it’s not there…

• Fellow analysts and students are not the target audience of many published analyses
  • We’re left to “pick” through for technically useful info

• Added effort - It’s a lot of work to get your internal notes and tools fit for outside consumption

• Analysis-consumer safety - preventing the reader for inadvertently infecting

• Client confidentiality - Compelling. May be client-specific data in targeted malware

• Competitive advantage - public relations, advertising services, showcase of technical ability
  • Perhaps not in our best interest to allow someone to further it, do it better, or worse: prove it wrong.
What’s Being Done Elsewhere?

• Reproducibility and verifiability are a big deal in any academic/scientific endeavor

• Peer review is \textit{supposed} to act as the filter here
  \begin{itemize}
  \item (Though maybe we aren’t as rigorous as we ought to be with it in computer science/engineering)
  \end{itemize}

• Software, environment, data, documented to the point that someone can recreate the experiment

• \textit{Executable/interactive research paper}
  \begin{itemize}
  \item Embedded algorithms and data,
  \item (Doesn’t that sound a bit scary re: Malware? :) )
  \end{itemize}
Recommendations

• Beyond sandbox output…

• Sample availability (!!!!!!!!!)
  
  • virusshare.com is the best positive example of the right direction here

• Host environment documentation

• Target data - give it something to exfiltrate

• Network environment - give it what it wants to talk to

  • **Instrumentation** - programmatic, running commentary
    
    • Scriptable debugging (winappdbg!)
    
    • Isolate functionality, document points of interest, put it all into a big picture
Case Study: JackPOS
Ackowledgements

• Samples - @xylit0l - http://cybercrime-tracker.net

• Prior-to-now-but-post-this-work analyses
  • http://blog.spiderlabs.com/2014/02/jackpos-the-house-always-wins.html
  • http://blog.malwaremustdie.org/2014/02/cyber-intelligence-jackpos-behind-screen.html

• Please check the white paper citations for tools, executable paper prior work, etc.

(to make sure I get these in before we geek-out on the demo)
Why JackPOS?

• Current concern surrounding POS malware

• C2 availability - Ability to demonstrate a complete environment
  • From card-swipe to command-and-control

• C++ strings, STL - runtime objects make static analysis with IDA Pro a bit more awkward

• Good use case for harnesses
  • Independent memory-search functionality
Harness Design

- WinAppDbg - Python scriptable debugging
  - Really fun library - Well-documented, lots of examples, easy to use
  - Callbacks for breakpoints

```python
breakpoints = [(0x00401B38, patch_cnc),
              (0x004048F8, patch_install_check),
              (0x0040B796, shell_execute_blocker),
              (0x00402F84, open_url),
              (0x0040320A, cnc_online),
              (0x00403321, cnc_online_end),
              (0x004035BC, cnc_send),
              (0x00403627, cnc_recv),
              (0x00409388, search_process),
              (0x004099FE, process_kill),
              (0x00408DAC, kill_block_registry_cleanup),
              (0x004095E6, process_update),
              (0x00409CA7, process_exec),]

def patch_cnc(event):
    """
    Breakpoint: 0x00401B38
    Patch the CnC to ours
    """
    process, thread, context = get_state(event)
    original_cnc = process.peek_string(0x004339BC)
    process.write(0x004339BC, debug_cnc+"\x00")
    print_modification('Modified CnC from %s to %s' % (original_cnc, debug_cnc))
    esp = context['Esp']
    process.write_dword(esp+0x04, len(debug_cnc))
    print_modification('Patched length to %i' % (len(debug_cnc)))
    return
```
JackPOS

- Example sample - SHA1
  9fa9364add245ce873552aced7b4a757dceceb9e

- Available on virusshare (and mcgrewerksecurity.com)

- This is the only part *not* on the DEF CON DVD.

- Command and Control

- PHP, Yii Framework
Command and Control

- Data model - bots, cards, commands, dumps, ranges, tracks, users
Back to the sample

• UPX (thankfully not an unpacking talk/tutorial)

• Unpacked version crashes due the stack cookie seed address not relocating

• Easy fix: disable ASLR (also makes our analysis easier), unset:

  • IMAGE_NT_HEADERS > IMAGE_OPTIONAL_HEADER > IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE
Setup

- String setup - c2, executable filenames, process names for memory search
- Installation (copying self)/persistence (registry)
- Harness patches -
  - Command and control
  - Installation check
  - Prevents watchdog process (and anything else from ShellExecute’ing)
Communication

• Command and Control Check-in
  • Checks C2 for http://[c2]/post_echo
    • (PostController.php responds “up”)
  • Prevents simple sandbox from getting much
  • If there’s track data, base64 it and send it
    • Harness configured to display data sent
  • Check command queue
  • Hosts uniquely identify by MAC
Commands

• Credit card track theft happens without having to be commanded to do so

• Remainder of command set is simple:
  • kill
  • update - (replace current install with latest from /post/download)
  • exec <url>
Scraping Memory

• Get a list of functions
  • No 64-bit process
  • No processes matching internal table (system, etc)

• Iterate and search for card data using two regular-expression-esque functions
  • ISO/IEC 7813 (we can generate and instrument this)

• Harness identifies search process

• Another harness can be used to instrument the code to scan arbitrary PIDs
Demo

• Sample MD5 - aa9686c3161242ba61b779aa325e9d24

• Harnesses
  • jackpos_harness.py - Instruments all operation
  • search_proc_harness.py - Skips to and illustrates track-data capture

• Track data generator - Generate and hold card swipes in memory

• PHP source for (actual) C2
  • (recreated DB schema (uh it works))
Wrapping up

• Addressing reproducibility/verifiability, potential benefits
  • Effective illustration for lay audiences, students
  • Base to work from (not “from scratch”) for other analysts
• Illustration using the resources malware “wants”, vs. generic sandbox
• Potential for publishing instrumented analysis in virtual/cloud environments for others to work with more immediately
Contact Info

wesley@mcgrewwsecurity.com
@McGrewSecurity