



# Host-based, Run-time Win32 Bot Detection

Liz Stinson  
John Mitchell



## bot : definition

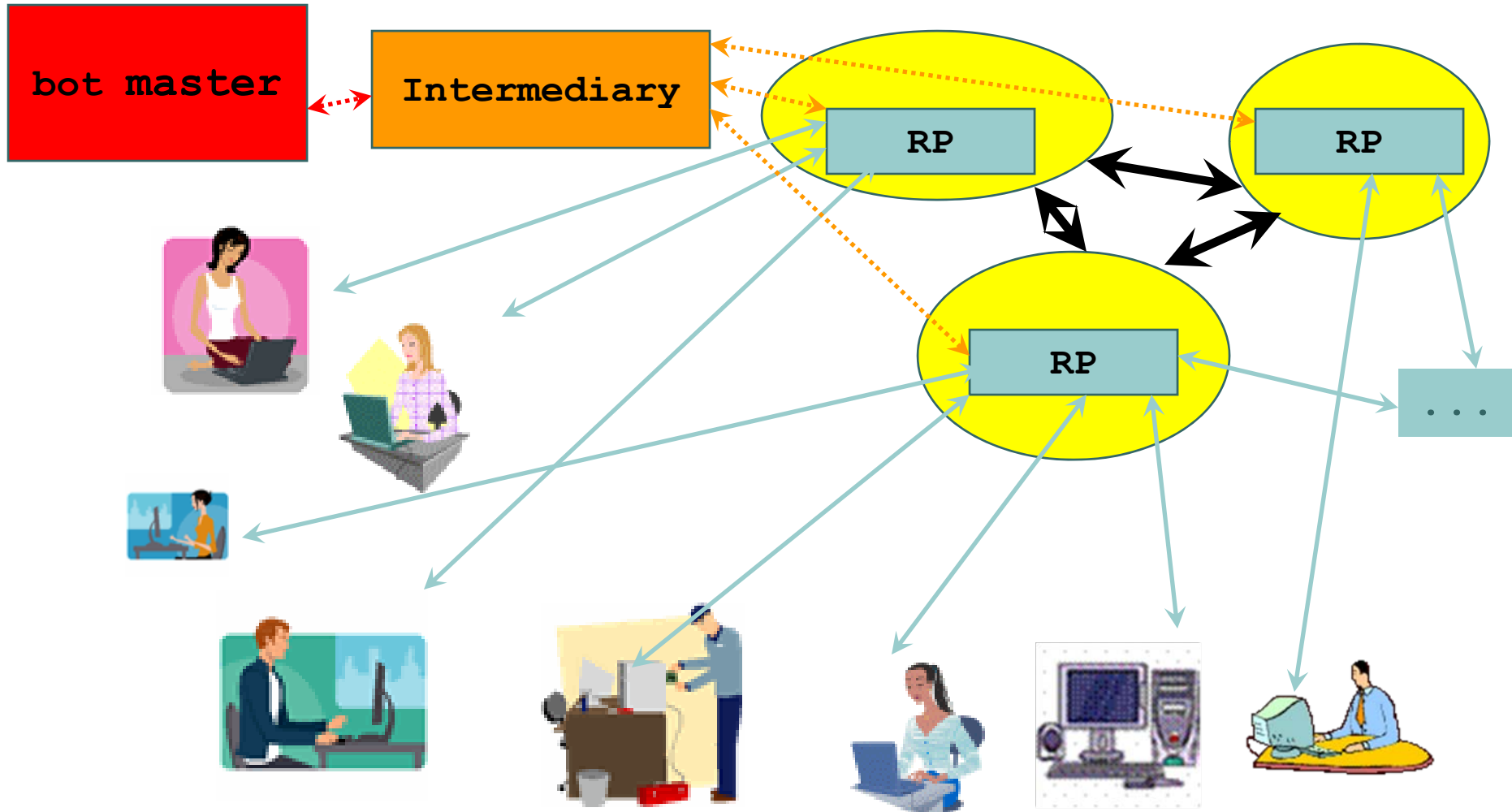
- *automated program; simulates human activity*
- We're interested in malicious ones
- Under real-time, fine-grained control by an entity that simultaneously manages other bots
- Bots mostly useful in the aggregate and proportionally so – i.e. as a *botnet*
- Applications:
  - DDoS extortion, Google AdSense abuse, identity theft, spamming, phishing-site hosting



# bots : characteristics

- General, multi-purpose, extensible
  - Download arbitrary files
  - Execute arbitrary programs
  - Connect to arbitrary IPs
- Ongoing control via a command-and-control (C&C) network :
  - C&C protocol
  - *rendez-vous* point

# bots : C&C network





# bots : functionality

- Every bot we looked at provided a way to :
  - Download a file and optionally execute it
    - Most offered just via HTTP; some also via FTP
    - Implies update functionality
    - Also visiting URLs and optionally specifying a referrer
    - Including HTTP server software so can host phishing site
  - Participate in DDoS
    - Some have quite elaborate attack vectors
  - Create port redirects (HTTP{S}, SOCKS, ... proxies)
- Most bots provided a means to clone and spy
- Some bots :
  - Spamming (accounting for 70% of all spam, Message Labs, '04)
  - Scanning, Spreading
  - Key logging ...



# Distinction from precursor malware : ongoing C&C

## ○ How to exploit this :

- Network-based approaches : e.g. develop a signature for bot traffic then flag
  - Problem: perturb bot traffic to be indistinguishable from legitimate traffic (or encrypt it)
  - Assume: *can't rely on particular protocol, port, payload*
    - Motivates against content-based filtering
- Host-based : ours. Have more info at host level. Since the bot is controlled externally, use this meta-level behavioral signature as basis of detection

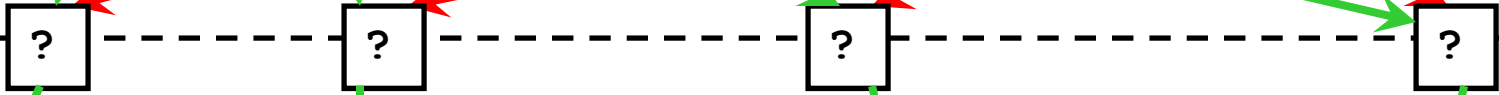
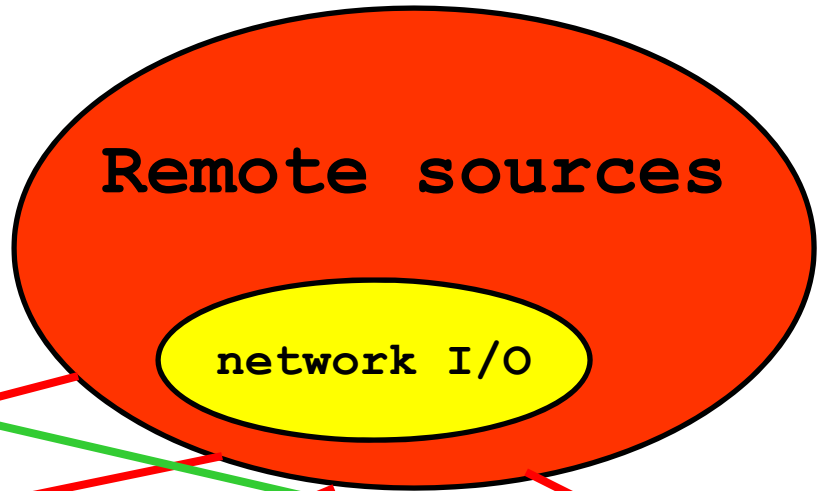
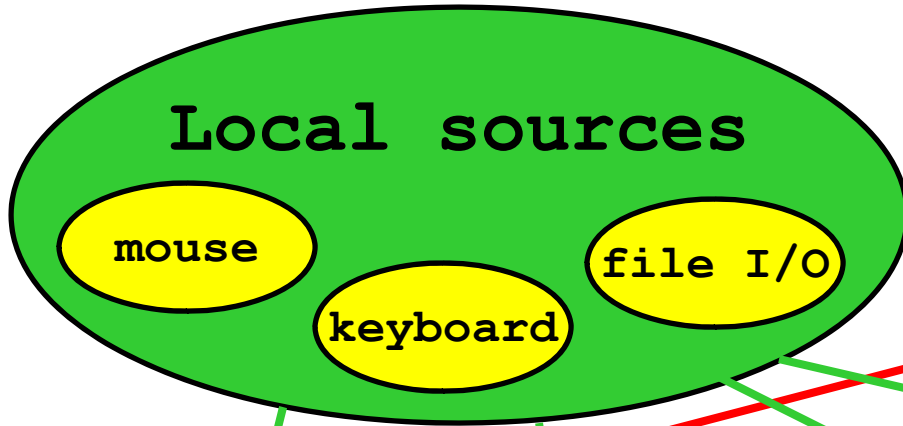


# Our approach : identify instances of *external control*

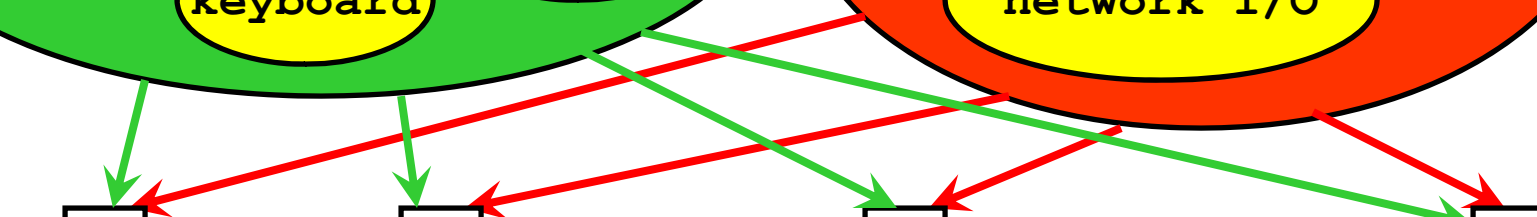
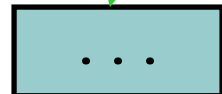
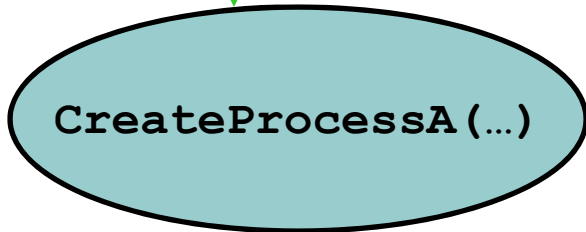
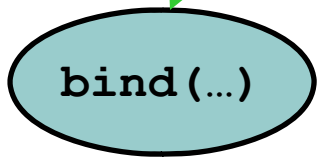
- Look at the “system calls” made by a program
  - In particular at certain of the *args* to these syscalls
    - We’ll call these chosen arguments our *sinks*
- Possible sources for these sinks can be categorized :
  - local : { mouse, keyboard, file I/O, ... }
  - remote : { network I/O }
- An instance of **external control** occurs when *data from a remote source reaches a sink*
  - Implies no value judgment a/b the nature of that control

# Big picture schema

S  
O  
U  
R  
C  
E  
S



S  
I  
N  
K  
S







# outline

- I. Bots
- II. Our approach to detection
  - A. Motivating abstraction
  - **B. Platform**
  - C. Design / Implementation
- III. Results
  - A. Bot testing
  - B. Benign program testing
- IV. Evading detection
- V. Future directions

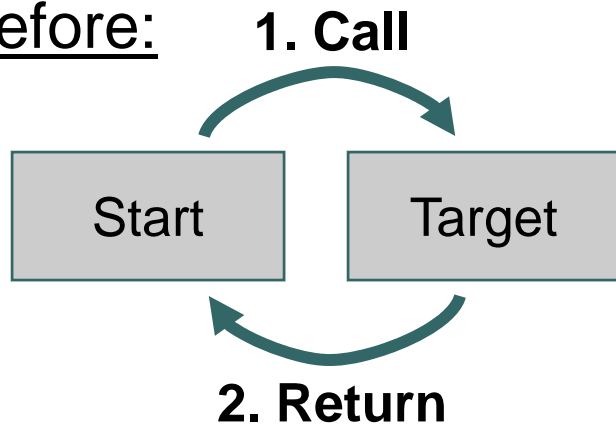


# Platform : detours library

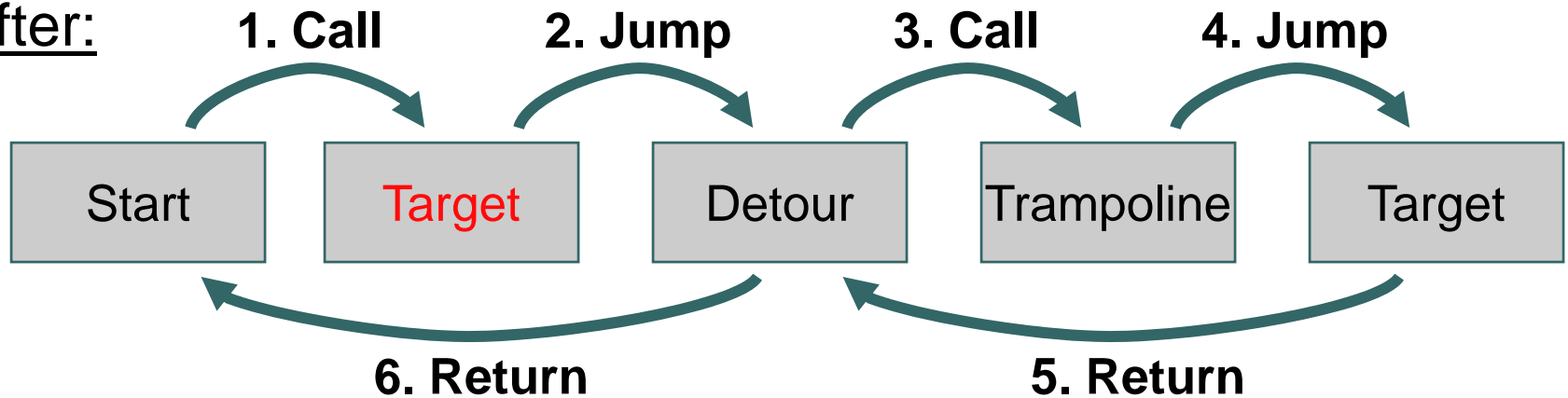
- API interposition; can hook any function as long as it's exported (by name) by a DLL (don't need \* .lib files)
- Overwrite first 5 bytes of *memory image* of targeted function with an uncond jmp to your replacement fxn
  - DLLs copy-on-write for NT family
    - User-land hooking; Not system-wide
- Don't need src; just inject our DLL into target process
- No changes to target binary
- Applies to calls made by calls
- Link time irrelevant
- Hunt/Brubacher, MSR
  - <http://research.microsoft.com/sn/detours/>

# Pictorially – c/o detours folks

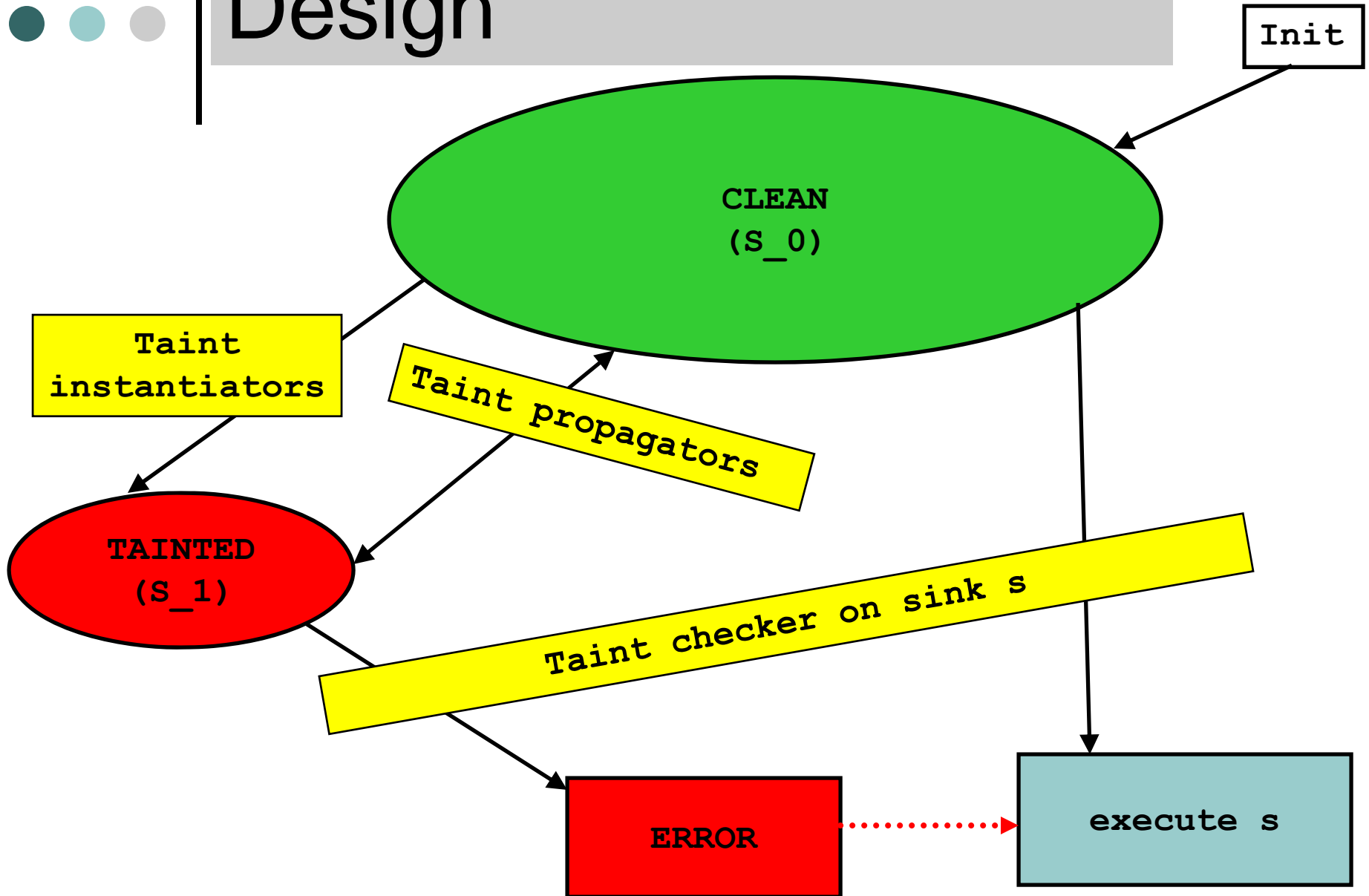
Before:



After:



# Design





# Two types of tainted things

1. **Memory regions** : [addy, addy + len) tuples
  - Parameterized by receive offset, local offset, amount of dirty data, ancestor receive buffer, type (ANSI or Unicode), ...
    - **Why all this data?** False positive mitigation
2. **Values** : strings, integers
  - **Why strings?** Provide some resilience against OOB copies (cf. spybot)
  - **Integers** : port numbers, IP addresses, ...

# Taint instantiation : network receive functions

```
recv( addy ) → len
```

for every  $w_i \in [addy, addy + len)$

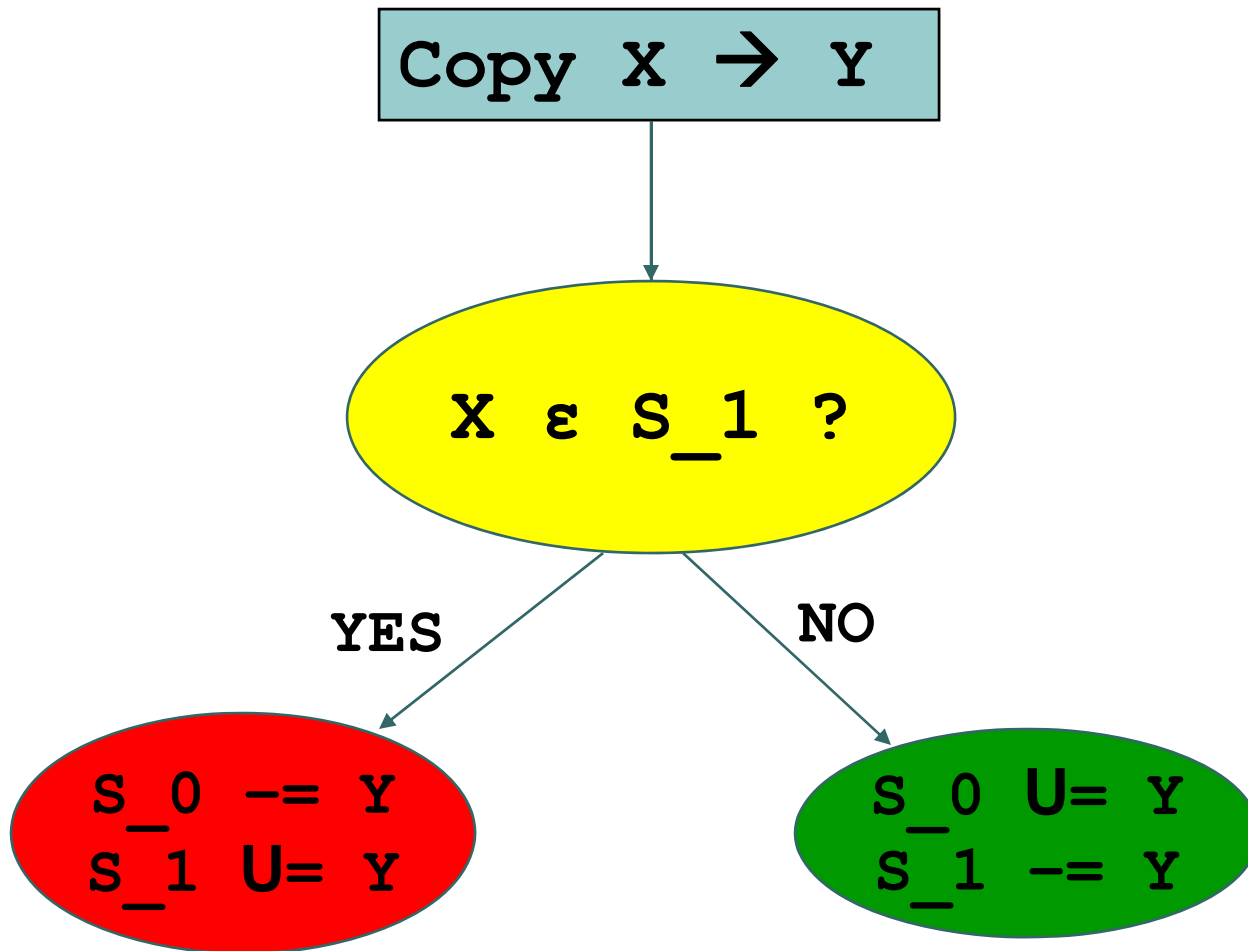
$S_0 \text{ --} w_i$

$S_1 \text{ U} w_i$

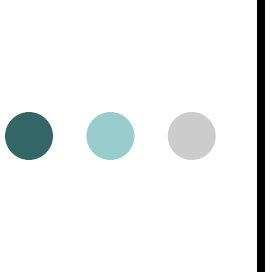
# Taint instantiators : network receive functions

- `recv(...)`, `recvfrom(...)`, `WSARecv(...)`,  
`WSARecvFrom(...)`,  
`WSARecvDisconnect(...)`, ...
- `NtDeviceIoControlFile` : non-blocking
- Do two things here:
  1. Create cached copy of this `recv` buffer (used in false positive mitigation – later)
  2. Add this `[base, base + bounds)` pair to tainted addys

# Taint propagation







# Taint propagators : memory regions

## ○ Obvious

- C library functions : string-copying, string concatenation, mem-copying/moving, buffer formatting (sprintf,...), ...
  - Where? From every C run-time libe to be thorough plus ntdll.dll
- Win32 versions of these : lstrcpy{A,W}, StrCpyN{A,W}, wsprintf{A,W}, ... plus safe versions of each
  - Where? Spread across a dizzying number and variety of DLLs
- Conversion functions : multi-byte to wide-char and vice versa

## ○ Less obvious

- `realloc(...)`
  - `SearchPath{A,W}(...)`
  - `InternetCrackUrl{A,W}(...), ...`
- In general: any function that takes an ANSI or Unicode buffer and outputs same where the output is a sub or superset of the input



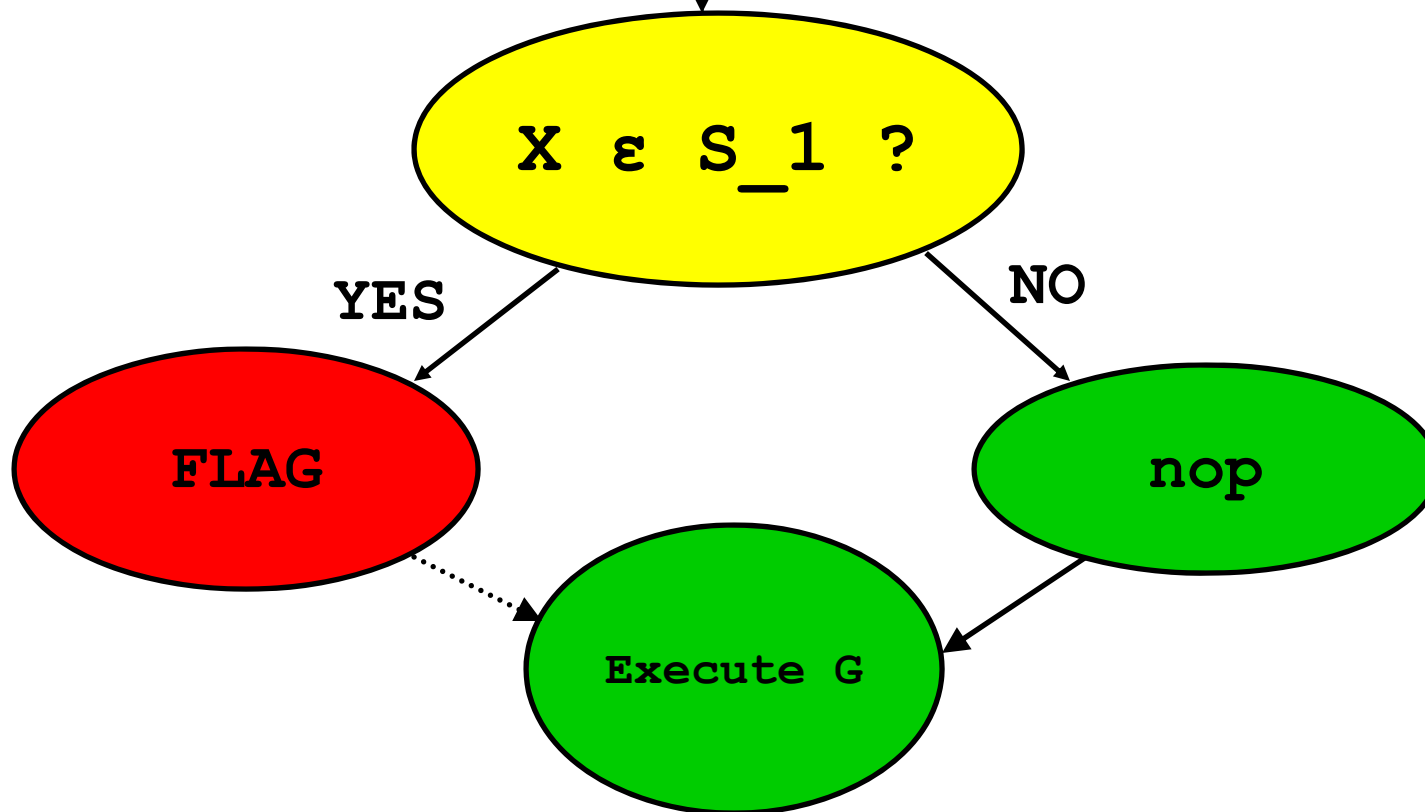
# Taint propagators : values and other

## Two other types of taint propagators:

1. Certain fxns that take a character buffer input, and output a value :
    - E.g. `atoi(...)` → { port #s, PIDs, ... },  
`gethostbyname(...)`, `inet_ntoa(...)`, ...
  2. Any fxn that takes a char buffer input, we check:
    1. Is this a tainted memory region?
      - a) If so, make sure the string therein is a dirty string
      - b) If not, is this a dirty string?
        - i. If so, transitively taint its memory region
- [Symbiotic r/p b/n tainted addys, dirty strings]

# Taint checking

on arg X to gate G





# Gate functions

- Process management : dirty filenames, PIDs
  - `CreateProcess{A,W}`, `WinExec`
  - `NtTerminateProcess`
- File management : dirty filenames
  - `NtOpenFile`
  - `NtCreateFile`
- Network interaction : dirty IPs, port #s; tainted send
  - `NtDeviceIoControlFile` (`send`, `sendto`, `bind`, ...)
  - `connect`, `WSAConnect`
  - `sendto`, `WSASendTo`
  - `HttpSendRequest{A,W}`
  - `SSL_write` (calls into `send(...)` w/encrypted output buf)
  - `IcmpSendEcho`

# Behaviors : ideally disjoint; check at lowest possible level

#	Name
1	tainted NtOpenFile
2	tainted NtCreateFile
3	dirty program execution
4	dirty process termination
5	bind dirty IP
6	bind dirty port
7	connect to dirty IP
8	connect to dirty port
9	dirty send
10	derived send
11	sendto dirty IP
12	sendto dirty port
13	dirty HttpSendRequest
14	dirty IcmpSendEcho

MoveFile{Ex}{A,W},  
MoveFileWithProgress{Ex}{A,W},  
DeleteFile{A,W}, ReplaceFile{A,W},  
Win32DeleteFile, ...

CreateFile{A,W}, OpenFile,  
CopyFile{Ex}{A,W}, fopen,  
\_open, \_lopen, \_lcreat, ...

ShellExecute{Ex}{A,W},  
CreateProcess{A,W}, WinExec

send, sendto, WSASend, WSASendTo



# Results

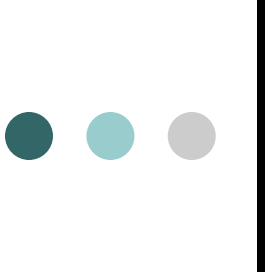
- Looked at 5 bots : agobot, dsnxbot, g-sysbot, sdbot, spybot
- At least three of these have totally independent code bases : agobot, dsnxbot, sdbot
- Sdbot, g-sysbot, spybot : shared ancestry but spybot differs in a non-trivial manner and even sd/gsys don't export same command interface
- In general *overall approach* to implementing functionality *X* may be different and almost certainly code to do same differs (greatly) from bot to bot











## Resulting observation : bots implicitly amp the S in the SNR

- Our detection platform treats all network receive buffers the same (regardless of their contents: HTTP, IRC, FTP, SMTP, ...)
- We let the bot tell us (implicitly) what's interesting : which recv bufs, what parts
- Do otherwise and system becomes very fragile (e.g. bot writer changes delimiter) and open to exploitation (white-listed words)

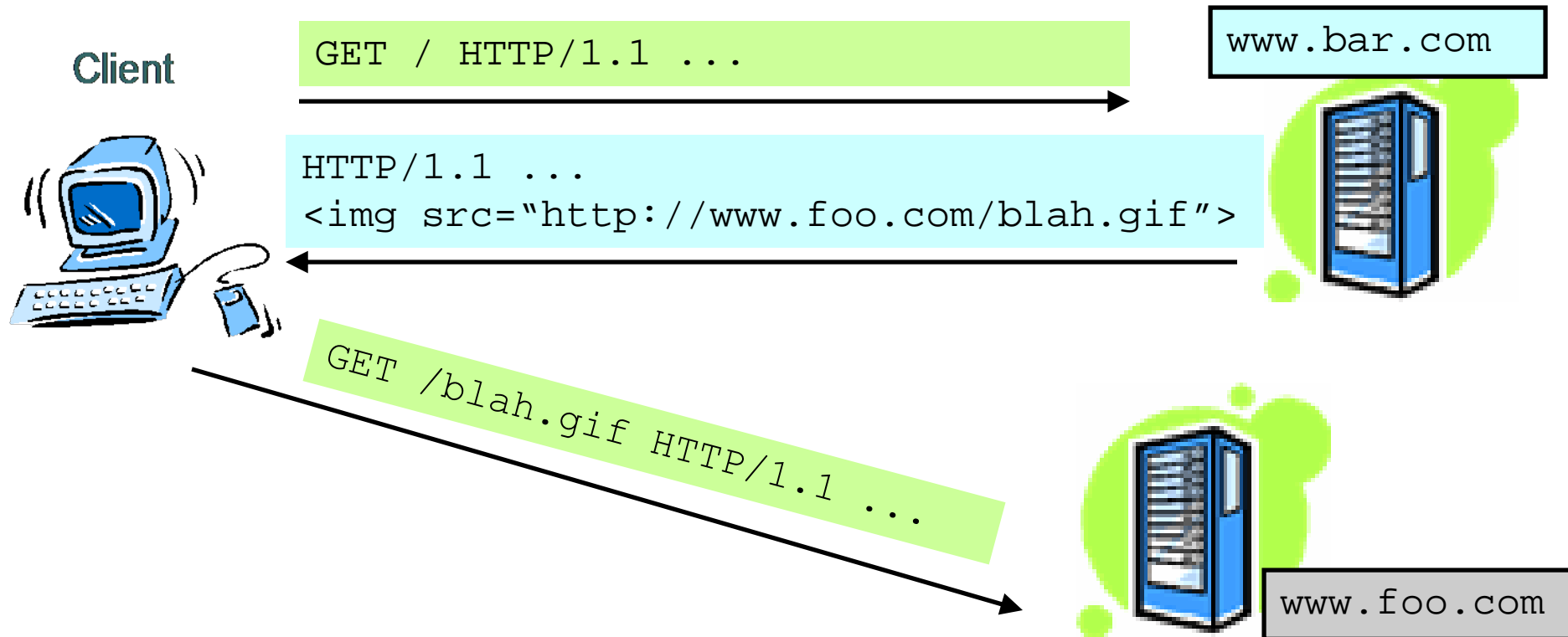


# False positives : technical

1. Values : since no visibility into value assignment
  - Some value is obtained from a tainted memory region (via call to `:atoi`, `gethostbyname`, ...)
  - That same value is later used in a sink
  - But no cause/effect relationship actually present
2. Strings : since transitively taint mem region if its contents match a known dirty string
  - E.g. “.execute C:\WINDOWS\system32\notepad.exe {0,1}” followed at some later point by any command (e.g. “open somefile.txt”) which involves executing `notepad.exe`
    - Will flag later command’s call to `NtOpenFile/NtCreateFile` on `notepad.exe` as dirty even though strictly speaking it’s not

# False positives : contextual

- Flagging as malicious widely accepted behavior
- E.g. Web browsers, server programs





# False negatives

- `HttpSendRequest` : calls into `connect(...)` – usually – and `send(...)`
  - Private `StringBuffer` implementation in `wininet.dll` used to craft actual HTTP message
- Spybot downloading file
  - Tokenizes URL into hostname and filepath byte-by-byte via '='
- Scanning agobot : input is an IP range `192.168.100/24` e.g. – fourth octet obtained via `rand(...)`



# Benign program testing

- Have performed some preliminary testing against benign programs that interact with the network
  - IE, Firefox, Outlook, IRC clients
- *These programs generated no flagged behaviors*
- Not good news w.r.t. the browsers; likely a code coverage issue
- More work to do here (breadth + depth)



# Evading detection

1. Don't do anything : dormant bots not detected
2. Convert parameterized bot commands to take no parameters
  - Presumably this occurs at cost of granularity of control
3. Statically link in C library functions
  - Maybe we can convert statically-linked executable to one that uses C run-time libraries; but in an adversarial environment? Harder.
4. Write own versions of mem-copying, tokenizing, ... fxns
5. Easier : encrypt using private encryption functions
  - If use any mechanism to encrypt that we have visibility into (e.g. OpenSSL), we can still detect
6. Get out of the detours sandbox
  - phrack 0x62, section 0x05



# Future directions

- Look at other malware, e.g. worms
- Move tainting to lower level
- More/other gate functions?
- Identify high-level behaviors from sequences of component behaviors; e.g. “port redirect”