virus BULLETIN Covering the global threat landscape

INVADING THE CORE: IWORM'S INFECTION VECTOR AND PERSISTENCE MECHANISM

Patrick Wardle
Synack, USA

iWorm is a recently discovered *OS X* backdoor that affords an attacker complete control of an infected host. Initial reports provided a fairly thorough overview of the malware's functionality, however iWorm's infection vector was not identified, and its persistence mechanism was not discussed sufficiently. These reporting shortcomings were (at least partially) remedied by subsequent reports that revealed how the malware both gained and retained access on a host.

This paper builds upon the latest analyses, and provides a more comprehensive technical analysis of iWorm's infection vector and persistence mechanism. Armed with this information, users will be able to detect existing iWorm infections and prevent future infections. This is essential, as users may not be able to count on *Apple*'s anti-malware mitigations (*XProtect*, *Gatekeeper*, etc.) to protect them from this threat. More importantly, this analysis reiterates several well established security mantras and provides information that will both educate and safeguard users, even against future threats.

IWORM APPEARS AS A RIDDLE

iWorm was first reported on 29 September 2014 by the Russian anti-virus company *Doctor Web* [1]. The researchers' initial analyses provided a decent overview of this 'multi-purpose backdoor' [1], briefly describing its capabilities and its unique *Reddit*-based C&C server location mechanism. Unfortunately, the report left out several pertinent facts, such as the malware's infection vector, and did not provide specifics as to its persistence mechanisms. Therefore, the remainder of this paper will provide a comprehensive analysis of both iWorm's initial infection vector and its persistence mechanism. Moreover, suggestions for both the detection and prevention of iWorm infection will be discussed and an open-source tool will be detailed that can enumerate and display persistent *OS X* binaries generically, including iWorm. This short paper should give readers a comprehensive understanding of iWorm's infection and persistence mechanisms, and of how to both detect and prevent the malware. Moreover, by covering various best practices, analytic techniques, and a generic detection tool, it is hoped that users will remain secure against other *OS X* malware as well.

INITIAL INFECTION VECTOR

A few days after *Doctor Web*'s initial reports on iWorm, the owner of Mac security website *The Safe Mac* reported that he had received an email tipping him off as to the malware's infection vector [2]. The email described how pirated versions of desirable *OS X* applications (such as *Adobe Photoshop* and *Microsoft Office*) were infected with the malware. In other words, iWorm is 'a classic trojan – a program which tricks you into installing malware, usually bundled with legitimate software' [3]. Magnet links to these infected applications were uploaded to *Pirate Bay* by the user 'acerprog' in order to

Type	Name (Order by: Uploaded, Size, ULed by, SE, LE)	SE	LE
Applications (Mac)	Adobe Photoshop CS6 for Mac OSX Uploaded 07-26 23:11, Size 988.02 MiB, ULed by aceprog	80	3
Applications (Mac)	Parallels Desktop 9 Mac OSX Uploaded 07-31 00:19, Size 418.43 MiB, ULed by aceprog	39	3
Applications (Mac)	Microsoft Office 2011 Mac OSX Uploaded 07-20 19:04, Size 910.84 MiB, ULed by aceprog	421	9
Applications (Mac)	Adobe Photoshop CS6 Mac OSX Uploaded 07-26 23:18, Size 988.02 MiB, ULed by aceprog	261	13
Applications (Mac)	Adobe Photoshop CC 2014 Mac OSX Uploaded 07-29 05:19, Size 801.44 MiB, ULed by aceprog	371	13
Applications (Mac)	Adobe Illustrator CS6 Mac OSX Uploaded 07-31 05:19, Size 1.42 GiB, ULed by aceprog	213	21

Figure 1: Links to iWorm-infected torrents on Pirate Bay.

reach a global audience. Although the links have since been removed, the applications were downloaded prior to their removal.

INSTALLATION: PART 1

As described in the anonymous write-up [3], the installers for the pirated applications contained malicious code that installed the iWorm malware persistently. The write-up continued by analysing these applications in order to provide an illustrative overview of the actual infection technique. Here, a few extra (technical) details will be added to paint a comprehensive picture of how iWorm infects a target computer. Specifically, the infected '*Adobe Photoshop CC 2014 Mac OS X*' torrent will be dissected to reveal iWorm's installation (infection) and persistence mechanisms.

Whenever a user launches an application, *OS X* consults the application's Info.plist file for the app's binary. As shown in Figure 2, the Info.plist file for the infected *Adobe Photoshop* application specified that the 'Install' binary should be executed whenever the application was launched.

The 'Install' binary existed in the usual Contents/MacOS folder within the infected application. Oddly, two other binaries (named '0' and '1') were also present within this directory. Their purpose will be described shortly.

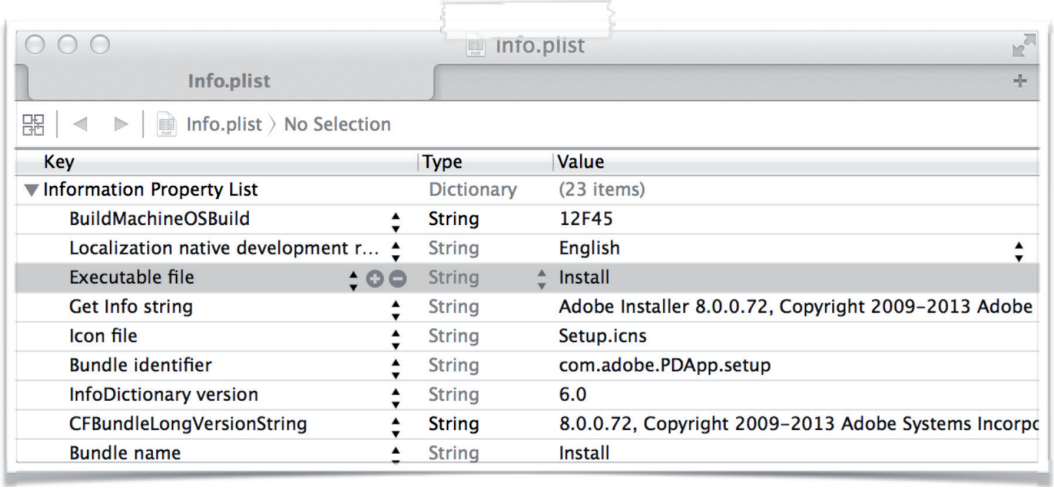

Figure 2: Infected application's Info.plist file.

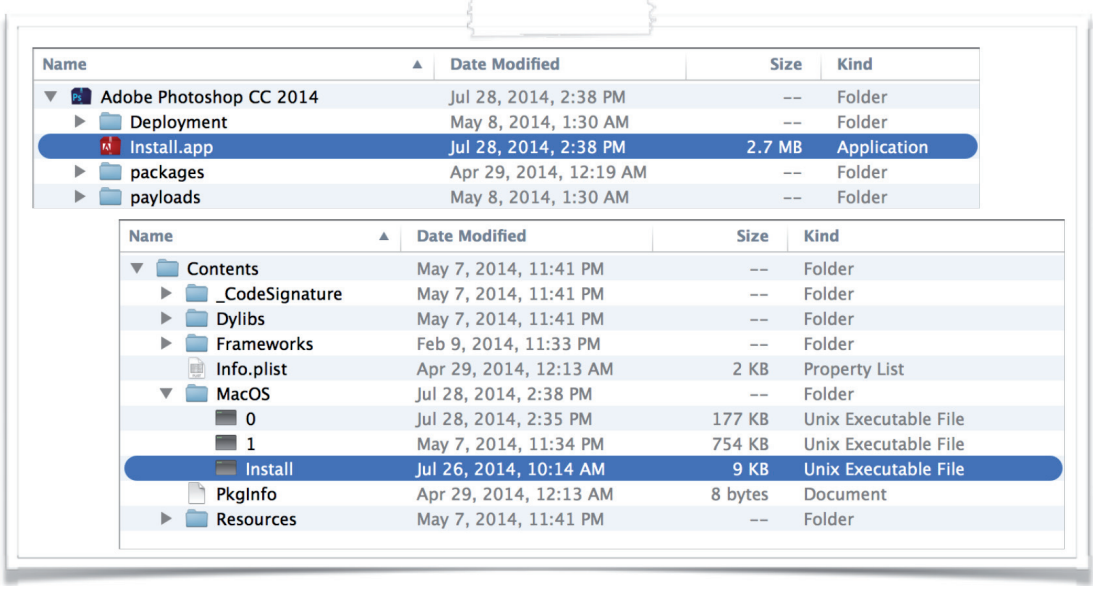

Figure 3: Infected installer application.

Whenever a user launches the Install.app file (within the 'Adobe Photoshop CC 2014' folder), the 'Install' binary will be executed. Disassembling this 32-bit Mach-O executable provided an insight into its actions. In short, the 'Install' binary simply launched the '0' binary with elevated privileges, then executed the '1' binary before exiting. The former action is illustrated in the disassembly and pseudo-code shown in Figures 4 and 5.

```
;build path to '0' binary
mov     [esp+4], esi
lea     edi, [ebp+path]
mov     [esp], edi
call    _strcpy
mov     [esp], edi
call    _strlen
mov     [ebp+eax+var_406], 0
mov     word ptr [ebp+eax+path], '0/'

;spawn '0' with auth prompt
lea     eax, [ebp+authorizationRef]
mov     [esp+0Ch], eax
mov     dword ptr [esp+8], 0
mov     dword ptr [esp+4], 0
mov     dword ptr [esp], 0
call    _AuthorizationCreate
mov     eax, [ebp+authorizationRef]
mov     [esp+4], edi
mov     [esp], eax
mov     dword ptr [esp+10h], 0
mov     dword ptr [esp+0Ch], 0
mov     dword ptr [esp+8], 0
call    _AuthorizationExecuteWithPrivileges
```

Figure 4: iWorm installer's disassembly (IDA).

```
//get path to self ('Install')
appRef = CFBundleCopyBundleURL(CFBundleGetMainBundle());
pathRef = CFURLCopyFileSystemPath(appRef, kCFURLPOSIXPathStyle);
path  = CFStringGetCStringPtr(pathRef, CFStringGetSystemEncoding());

//replace 'Install' with '/0'
// ->this is the binary to executed
*(strrchr(path, '/')) = 0x0;
path += "/0"

//spawn '0' with auth prompt
AuthorizationCreate(NULL, kAuthorizationEmptyEnvironment,
                    kAuthorizationFlagDefaults, &authRef);

AuthorizationExecuteWithPrivileges(authRef, path,
                    kAuthorizationFlagDefaults, NULL, NULL);
```

Figure 5: iWorm installer's pseudo-code.

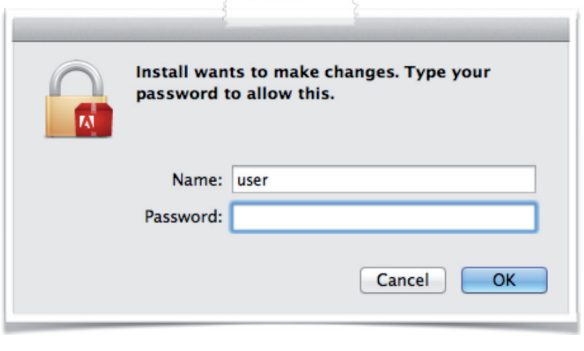

Figure 6: iWorm installer's authorization prompt.

The use of the AuthorizationCreate() and AuthorizationExecuteWithPrivileges() API functions generated an authorization dialog that would be presented to the user, as shown in Figure 6.

If the user entered their password and clicked 'OK', the '0' binary would be executed with elevated privileges. Interestingly, if the user clicked 'Cancel', the binary would not be executed, as there is no code within the 'Install' binary to handle this scenario. As the '0' binary is malicious, clicking 'Cancel' would actually prevent infection.

Analysis of the two binaries revealed that the '0' binary was a second-stage installer (dropper) for the persistent malware component, while '1' was the legitimate installer for the pirated application (*Adobe Photoshop*). Spawning the malware dropper with elevated permissions makes sense, as in *OS X*, unprivileged binaries are fairly constrained and limited in what actions they can perform. Obviously, malware authors prefer for their creations to have free reign over an infected host. If a privilege escalation vulnerability is not used, simply asking the user for their password via an authorization prompt may achieve this privileged state. While the malware dropper is executing, it makes sense to execute the legitimate installer application in parallel. This ensures that the user will not notice that anything is amiss.

Figure 7 summarizes iWorm's initial installation phase, showing how the malware can find its way onto a user's system and gain privileged execution.

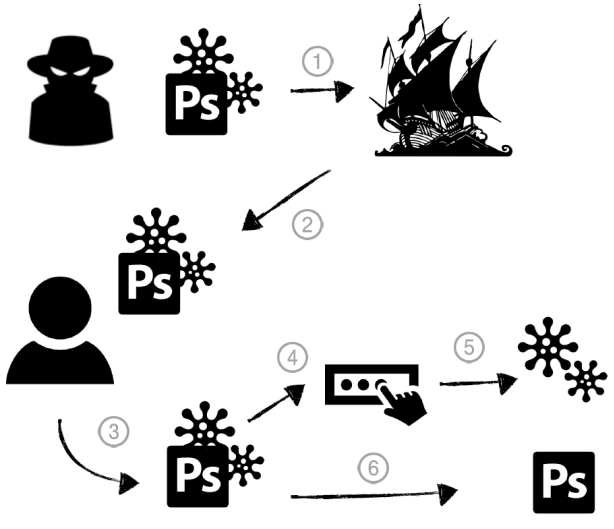

Figure 7: Overview of iWorm infection.

First, the malware author uploads an infected application to the popular torrent site *Pirate Bay*. Any user who downloads

and runs the infected application will become infected. Of course, the (fully functional) pirated application will also be installed – although that is quite a small consolation for turning over complete control to an unknown adversary!

INSTALLATION: PART 2

The last section described how the pirated *Adobe Photoshop* installer application invoked the 'Install' binary, which in turn would execute the '0' binary with elevated privileges. The '0' binary turned out to be a basic malware dropper. First, this dropper created the /Library/Application Support/JavaW/ directory, then it saved 0x29000 bytes from offset 0x00002050 (the start of its _data segment) to a file named JavaW, as shown in Figure 8.

```
# fs_usage -w -f filesys

20:26:15.386840  mkdir   /Library/Application Support/JavaW
20:26:15.387031  open    /Library/Application Support/JavaW/JavaW
20:28:28.727328  write   B=0x1000
20:28:28.727336  write   B=0x1000
...
20:28:28.727645  chmod   <rwxrwxrwx>
                         /Library/Application Support/JavaW/JavaW
20:28:28.727797  close
```

Figure 8: iWorm dropper writing to JavaW.

Single-stepping the malware through a debugger provided an easy way to examine the bytes as the dropper was writing them to disk, as shown in Figure 9.

```
(lldb) Process 726 stopped
* thread #1: tid = 0x56dd, 0x000019ef
  queue = 'com.apple.main-thread'
-> 0x19ef:  calll  0x1d14         ; symbol stub for: fwrite$UNIX2003

(lldb) x/16xw *(int*)$esp
0x00002050: 0xfeedface 0x00000007 0x00000003 0x00000002
0x00002060: 0x00000005 0x000001b8 0x00000005 0x00000001
0x00002070: 0x00000038 0x41505f5f 0x455a4547 0x00004f52
0x00002080: 0x00000000 0x00000000 0x00001000 0x00000000
```

Figure 9: iWorm dropper (within a debugger), writing bytes to JavaW.

Ah, good old 0xfeedface, the 'magic' number indicating an *Intel* Mach-O binary – the dropper was saving an embedded binary to disk. Once the embedded binary ('JavaW') had been saved, the dropper created a property list within the /Library/LaunchDaemons directory.

```
# fs_usage -w -f filesys

20:28:28.727871  open     /Library/LaunchDaemons/com.JavaW.plist
20:28:28.727890  write    B=0x16b
20:28:28.727937  close
```

Figure 10: iWorm dropper writing out a launch daemon plist.

The contents of this plist file are shown in Figure 11.

On *OS X*, there are many ways to ensure that a binary is executed automatically by the OS every time the computer is restarted [4]. For 'n00bie' *OS X* malware writers, launch items (daemons and agents) are the preferred method of persistence. To persist a binary as a launch daemon, one simply has to create a property list ('plist) within one of the launch daemon directories (e.g. /Library/LaunchDaemons). This plist should contain a dictionary of various key value pairs including the path to the persistent binary and flag (such as RunAtLoad), indicating how and when the binary should be started by the OS. Since the dropper created a launch daemon plist with the RunAtLoad key set to true, the persistent component of the malware (/Library/Application Support/JavaW/JavaW) would automatically be started by the OS on each reboot – persistence achieved!

Once the dropper had installed the malware persistently as a launch daemon, it executed the malware directly via two calls to the launchctl utility (Figure 12). This native OS utility interfaces with launchd, the 'system wide and per-user daemon/agent manager' [5]. The calls to launchctl and arguments passed were observed passively via a simple dtrace script [6].

The output of the dtrace script showed the dropper first loading, then starting the malicious launch daemon. It should be noted that, because the RunAtLoad key was set to true, the start command was surplus to requirements; the malware was

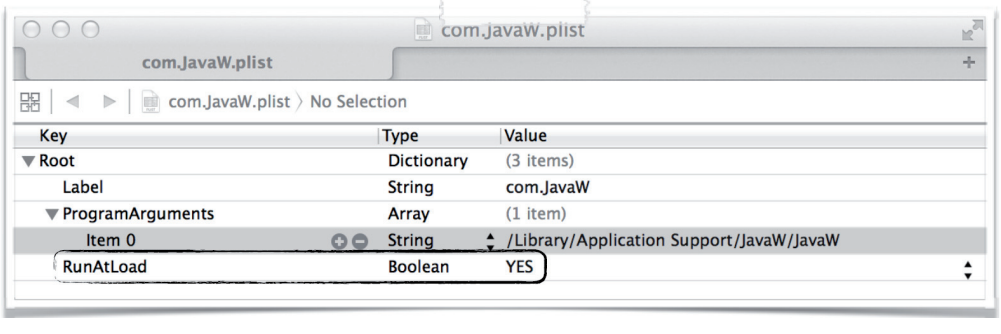

Figure 11: iWorm's launch daemon plist.

```
# ./watchProc

launchctl load /Library/LaunchDaemons/com.JavaW.plist
launchctl start com.JavaW
```

Figure 12: iWorm dropper launching iWorm via launchctl.

```
# launchctl load /Library/LaunchDaemons/com.JavaW.plist

# ps aux | grep -i javaw
root   1289   /Library/Application Support/JavaW/JavaW
```

Figure 13: Simplifying the launch of iWorm.

both loaded and started via the first load command [7] (see Figure 13).

With iWorm installed persistently (as a launch daemon) and started manually for the first time, the dropper exited.

IWORM PROPER

Various online analyses of iWorm, such as [1] and [8], provide a fairly thorough overview of the malware's capabilities and features. While the goal of this paper is to focus on the infection and persistence of iWorm, several of the more interesting components of the malware will briefly be discussed as well.

The iWorm binary ('JavaW') was packed with UPX. While packers are commonly used by *Windows*-based malware, amongst *OS X* malware specimens, this is a somewhat uncommon feature. Unpacking the binary (upx -d JavaW) decompressed it and allowed for analysis to commence.

As mentioned in the initial report from *Doctor Web* [1], the malware appeared to have been written in C++. While binaries on *OS X* are often written in Objective-C, it's likely that the background of the malware author(s) was *Linux*-based, and thus C++ was a more familiar language.

iWorm provides basic backdoor functionality, and contains no worm-like (i.e. self-spreading) capabilities. However, it does have a few tricks up its sleeve. First, in order to locate its command and control (C&C) servers, iWorm queried reddit.com. This query returned 'a page containing the list of botnet C&C servers and ports published by criminals in comments to the post minecraftserverlists under the account vtnhiaovyd' [8]. Unfortunately, although several of the subreddits that contained iWorm's C&C servers (e.g. minecraftserverlists) were banned, others such as 'ilikedota2' remained (and remain) online. Since these are still accessible, new iWorm infections are still able to resolve addresses for remote tasking. It should be noted that (other than using *Reddit*) this is not a novel technique. Other malware, including OS X Flashback, has used online services (such as *Twitter*) both for determining the location C&C servers and for direct command and control [9].

Another interesting feature of iWorm is its support for Lua [8]. The malware contained an embedded Lua interpreter that enabled it to execute Lua scripts directly. Such a feature allows the malware author(s) to dynamically (though not persistently) extend the core functionality of the malware by uploading and executing any scripts they desire.

```
;embedded lua tokens ('luaX_tokens')
__const:0005282C              dd offset aAnd        ; "and"
__const:00052830              dd offset aBreak      ; "break"
__const:00052834              dd offset aDo         ; "do"
__const:00052838              dd offset aElse       ; "else"
__const:0005283C              dd offset aElseif     ; "elseif"
__const:00052840              dd offset aEnd        ; "end"
__const:00052844              dd offset aFalse      ; "false"
__const:00052848              dd offset aFor        ; "for"
```

Figure 14: Snippet of iWorm's embedded Lua interpreter.

Interested readers are encouraged to read [8] to learn more about iWorm's capabilities (e.g. supported backdoor commands) and features (e.g. use of encryption).

GOODBYEWORM

Having gained a comprehensive understanding of iWorm's infection vector and persistence mechanism (coupled with a high-level overview of its features), it's time to discuss detection, prevention, and several security 'best practices'. In this case, the discussion is quite pertinent as *Apple*'s anti-malware mechanisms did little to protect unwitting users from iWorm infections. Of course, *Apple* zealots may point out that there is 'no patch for human stupidity', and that if users are downloading and running malware manually, the OS doesn't stand a chance. However, one would hope that the OS's anti-malware mechanisms would at least provide some level of protection. Unfortunately, *Apple*'s may provide none.

According to *Apple*, the *Gatekeeper* security feature helps protect Macs from malicious applications that are downloaded and installed from the Internet [10]. Aiming to be the first line of defence, it checks whether downloaded files are digitally signed, and may either warn the user or simply block a downloaded file from executing if it comes from an untrusted source. Contrary to popular belief, *Gatekeeper* (like *XProtect*) is fairly limited in the attacks it can prevent. This is due to the fact that *Gatekeeper* will only examine binaries that contain a quarantine attribute named 'com.apple.quarantine'. Interestingly, it is the responsibility of the downloading application (e.g. *Safari*, the torrent client, etc.) to set this quarantine attribute. If the downloading application does not set this, *Gatekeeper* will remain out of the loop. Unfortunately, many of the torrent clients that are likely to be used to download the infected applications may not set this attribute. For example, *uTorrent* (a popular *OS*

```
# xattr -p com.apple.quarantine Adobe\ Photoshop\ CC\ 2014.dmg
xattr: Adobe Photoshop CC 2014.dmg: No such xattr: com.apple.quarantine
```

Figure 15: No com.apple.quarantine.

```
# xattr -w -r com.apple.quarantine "0000;00000000;something;" Install.app/
```

Figure 16: With the quarantine attribute set, Gatekeeper displays an alert.

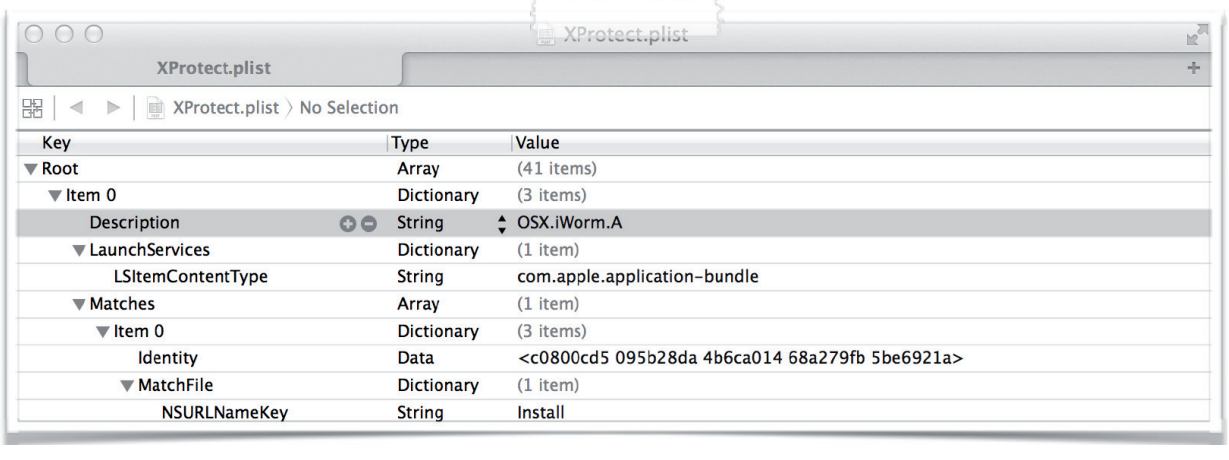

Figure 17: XProtect's iWorm signature.

X torrent client) did not set it, and thus when the infected installer was executed, no *Gatekeeper* prompt appeared. The malicious file was allowed to run in an uninhibited manner.

Of course, had the com.apple.quarantine attribute been set, a *Gatekeeper* alert would have been raised (since the iWorm installer was not signed) and the malware would have been blocked. This was confirmed by setting the attribute manually, then attempting to re-run the installer application. As expected, this (finally) resulted in *Gatekeeper* blocking the malicious installer (see Figure 16).

XProtect is *Apple*'s attempt at an anti-virus product. Implemented within the CoreServicesUIAgent, it uses signatures from /System/Library/CoreServices/CoreTypes.bundle/Contents/Resources/XProtect.plist to detect *OS X* malware. Since it is a static, signature-based AV product, it cannot detect (and thus prevent) new malware samples. Thus iWorm was initially free to ravage a user's system. To give *Apple* some credit though, although *XProtect* could not detect iWorm initially, once the malware had been reported, the company released several detection signatures.

Interestingly, the signature (shown in Figure 17) will only match (and thus block) the malware's installer application ('Install', sha1: c0800cd5095b28da4b6ca01468a279fb5be6921a). Although two other iWorm signatures were released,

```
# xattr -w -r com.apple.quarantine "0000;00000000;something;" Install.app/
```

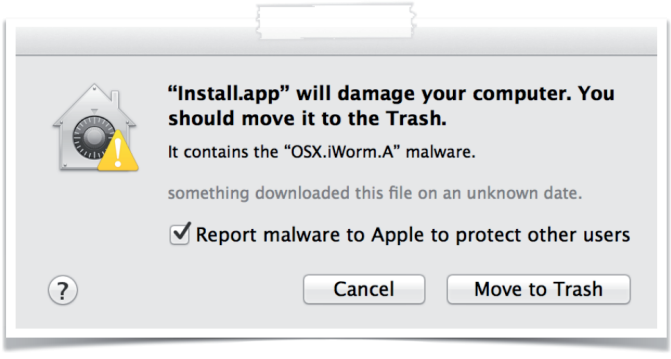

Figure 18: With the quarantine attribute set, XProtect detects the malware.

these also only detect the installer (variants 'B' and 'C'). This means that existing iWorm infections (e.g. the JavaW binary) will not be detected. Worse yet, like *Gatekeeper*, *XProtect* only scans files that have the quarantine attribute set. Thus, if the downloading application (e.g. *uTorrent*) does not set this attribute, the malware (including its installer) will still be able to execute freely and infect the user's system.

Since *Apple* seems unable to protect its users or detect the infection, how can users remain safe? First, the obvious: downloading pirated and cracked applications from untrusted sources is a bad idea (at least from a security point of view). As iWorm clearly illustrates, malware authors may use such applications to distribute their malware. Also, applications from untrusted sources that request elevated privileges should be treated with care. If the source cannot be verified, such applications should be avoided. If the application must be run, executing it within a virtual machine under the watchful eye of various profiling tools may be a possible solution.

Detecting the iWorm infection is actually fairly trivial, as the malware does not employ any rootkit or self-defence mechanisms. Several infection indicators are detailed in Table 1.

Removing the malware from an infected host is trivial as well. The steps shown in Figure 19 (run with elevated privileges) should remove all traces of the infection. Of course, if the malware has installed additional components, other steps may be necessary as well.

```
1) stop the malware
   # launchctl unload /Library/LaunchDaemons/com.JavaW.plist

2) remove the malware's persistence mechanism
   # rm /Library/LaunchDaemons/com.JavaW.plist

3) remove the malware's directory (and the malware binary, etc.)
   # rm -rf "/Library/Application Support/JavaW/"
```

Figure 19: Steps to remove iWorm.

Infection indicator type	Infection indicator	Description
Process	JavaW	The persistent iWorm component is named JavaW and runs as a launch daemon.
Directory	/Library/ApplicationSupport/JavaW/	The malware installer creates this directory to contain things such as the malware's binary (JavaW).
File	/Library/ApplicationSupport/JavaW/JavaW	The persistent iWorm component is named JavaW and is installed into the /Library/ApplicationSupport/JavaW/ directory.
File	/Library/LaunchDaemons/com.JavaW.plist	In order to persist, the malware installer creates this plist file.

Table 1: iWorm infection indicators.

As *Mac OS X* continues to increase in popularity, persistent *OS X* malware such as iWorm is becoming more common than ever. There are many locations on *OS X* that may be abused for persistence (such as launch daemons), and *Apple*'s anti-malware mitigations may not protect end-users.

In order to detect persistent *OS X* malware generically, a new tool has recently been developed that can enumerate and display persistent *OS X* binaries. Named *KnockKnock*, the goal of the tool is simple: to tell you who's there! Open source [11], and based on an extensible plug-in architecture, it can easily evolve as new methods of persistence are uncovered. *KnockKnock* can readily detect the presence of iWorm via the malware's persistence technique.

```
# python knockknock.py
WHO'S THERE:

[Launch Daemons]

JavaW
 path: /Library/Application Support/JavaW/JavaW
 plist: /Library/LaunchDaemons/com.JavaW.plist
 hash: 7c86f720cbfabb7d376493a2ca12adcd
```

Figure 20: Using KnockKnock to detect iWorm generically.

CONCLUSION

iWorm is a recent *OS X* backdoor that allows an attacker complete remote control over infected hosts. As detailed in this paper, iWorm was distributed via infected pirated applications that were hosted on *Pirate Bay*. When run, it was persistently installed as a launch daemon, which ensured that it would automatically be executed each time the infected computer was rebooted.

Although users cannot rely on *Apple*'s anti-malware mechanisms for protection from iWorm, refraining from using pirated applications should keep them safe in this case. More generically, armed with a tool such as *KnockKnock*, users can detect both current and future persistent *OS X* threats.

REFERENCES

[1] New Mac OS X botnet discovered. http://news.drweb.com/show/?i=5976.

[2] iWorm method of infection found! http://www.thesafemac.com/iworm-method-of-infection-found/.

[3] Mac.BackDoor.iWorm. https://docs.google.com/document/d/1YOfXRUQJgMjJSLBSoLiUaSZfiaS_vU3aG4Bvjmz6Dxs/edit.

[4] Methods of malware persistence on Mac OS X. https://www.virusbtn.com/virusbulletin/archive/2014/10/vb201410-malware-persistence-MacOSX.

[5] Man page for launchd. https://developer.apple.com/library/mac/documentation/Darwin/Reference/Manpages/man8/launchd.8.html.

[6] D-trace script. https://gist.github.com/viroos/1242279.

[7] What is launchd? http://launchd.info/.

[8] The Mac.BackDoor.iWorm threat in detail. http://news.drweb.com/show/?i=5977&lng=en.

[9] Flashback Mac Malware Uses Twitter as Command and Control Center. http://www.intego.com/mac-security-blog/flashback-mac-malware-uses-twitter-as-command-and-control-center/.

[10] OS X: About Gatekeeper. http://support.apple.com/kb/ht5290.

[11] KnockKnock. https://github.com/synack/knockknock.

Editor: Martijn Grooten

Chief of Operations: John Hawes

Security Test Engineers: Scott James, Tony Oliveira, Adrian Luca

Sales Executive: Allison Sketchley

Editorial Assistant: Helen Martin

Perl Developer: Tom Gracey

Consultant Technical Editors: Dr Morton Swimmer, Ian Whalley

Tel: +44 (0)1235 555139. Fax: +44 (0)1865 543153

Email: editorial@virusbtn.com

Web: http://www.virusbtn.com/