

A Simpler Way of Finding 0day

Robert Graham
David Maynor
Errata Security

Abstract: Instead of reverse engineering vulnerabilities to find 0day, hackers can now reverse security products. More and more companies are buying and commercializing 0day vulnerabilities and exploits. This includes offensive hacking toolkits, and also defensive products like vulnerability assessment appliances, intrusion detection systems, and intrusion-prevention systems. In this paper, we will demonstrate that it's possible to crack open a defensive product in order to get its 0day information. While we focus on one particular example here, the techniques are directly applicable to most other security products.

0Day – Early Protection and Competitive Differentiation

Buying and selling vulnerability information is not new in the underground, but it's new in the legitimate corporate world. Companies who specialize in vulnerability brokering have emerged from the shadows and are rapidly becoming a legitimate part of a good security strategy. Well known security vendors like IBM-ISS¹, Symantec², 3Com-TippingPoint, and eEye³ all have groups of researchers who find 0day⁴ flaws in the hopes that this edge will help sell products or services. Vulnerabilities are digital munitions and their brokers share a lot in common with arms dealers.

Undisclosed vulnerabilities have benevolent and malevolent value in a number of ways:

Benevolent

- Companies who buy vulnerabilities can show their product protecting against a vulnerability longer than their competitors
- Companies who discover vulnerabilities can show that their expertise is as good as any hacker by the quality of the vulnerabilities they find
- Companies can use their vulnerabilities in advanced penetration tests to demonstrate what a well-funded, determined attacker can do
 - Some companies like ImmunitySec and CORE sell offensive toolkits for use in “cyber warfare⁵” by “cyberoperators⁶” for reconnaissance or to attack and perhaps cripple critical infrastructure components of their adversary.
- Customers who subscribe to 0day information can find other ways of mitigating the threats without relying upon vendor solutions

Malevolent:

- Vulnerabilities can be used to spread malware, help increase botnets numbers, and to steal information

The 0Day Risk

However, making 0day information available for a price can result in a serious problem for vulnerability brokers. How do you tell a secret without letting the wrong people know? In the case of vulnerability commercialization it is almost impossible.

Take for example an Intrusion Prevention System (IPS) vendor buying 0day vulnerabilities to produce signatures. Those signatures contain information about the vulnerability and/or exploit. If these signatures are “open” to customers, the customers could potentially use those signatures to reverse engineer the 0day vulnerability and create their own 0day weaponized exploits.

Therefore, vulnerability brokers must do their best to hide the information, such as by encrypting signatures or giving vague research reports. However, as the DRM industry has shown, anything that a company does to encrypt information can be reverse engineered. When companies ship encrypted rules to their customers, hackers can intercept and decrypt that information, find the vulnerability, and develop weaponized exploits.

Further, hackers can develop new exploits based upon a signature that mutate that exploit so the signature doesn't catch it. Thus, shipping 0day signatures can endanger both the market as a whole, as well as a company's own customers.

Should we eliminate 0day in products?

Not necessarily. Everything that helps the defender has the potential of helping the attacker, 0day in defensive products is no different.

While it's theoretically impossible to ever protect secrets in products, vendors can do more to protect their information. The DRM vendors provide a good example. While DRM in both Windows and Apple products has been broken, their latest version *haven't* been broken recently. Unfortunately, most security products we have looked at have not taken this level of effort and 0day is easily retrieved from products.

Product with 0day

Many network security products claim to provide protection against 0day vulnerabilities. These vendors make attractive targets for hackers parties who to find 0day with as little effort as possible. Some of these vendors are ISS-IBM⁷, eEye, 3Com-TippingPoint, McAfee, and Symantec. 3com is one of the best know vendors to target because of their Zero Day Initiative⁸ (ZDI). This public program purchase vulnerability information from a larger community of external researchers.

Examination of advisories published by ZDI show there are often significant periods of time between their protection and a vendor's patch. Below is a random sample of advisories produced by the ZDI in 2007 that illustrate how long an hacker would have to make use of information reverse engineered from TippingPoint:

- 3Com-TippingPoint ZDI-07-038 Microsoft Internet Explorer Prototype Dereference Code Execution Vulnerability
 - Digital Vaccine released: October 10th, 2006
 - Patch issued by vendor: June 12th, 2007
 - Exposure time (roughly): 7 months
- ZDI-07-010 Apple Quicktime UDTA Parsing Heap Overflow Vulnerability
 - Digital Vaccine released: May 23rd, 2006
 - Patch issued by vendor: March 7th, 2007
 - Exposure time (roughly): 8 and a half months
- ZDI-07-013 Kaspersky AntiVirus Engine ARJ Archive Parsing Heap Overflow Vulnerability
 - Digital Vaccine released: December 12th, 2006
 - Patch issued by vendor: April 5th, 2007
 - Exposure time (roughly): 4 months

Because the ZDI is one of the largest buyers of Oday, and there is a long lag before vendors ship patches, they are the most attractive target for harvesting Oday information. In addition to being able to reproduce the vulnerability, seeing the signatures will tell the hacker how to evade the IPS.

Intrusion-prevention products have a wide-range of architectures. Some products, such as the one we worked on at ISS-IBM, use binary state-machine protocol-parsers. Other products, such as Snort, store regex patterns within text files. A text file containing patterns is much easier to read than binary state machines.

3Com-TippingPoint's rules are stored in a flat XML based file. All we need to do in order to retrieve the rules is to reverse engineer the encryption protecting that file.

TippingPoint is therefore an attractive target. The ZDI program buys a lot of Oday, there is a long lag before vendors fix the Oday, and their technology presents the Oday information in an easy-to-read format.

Analyzing the Target

We start by viewing the rule file in a hex editor. Normally TippingPoint rule files are named with a number scheme so it is easy to tell what version the rule set is. The scheme looks like SIG_2.2.0_5733.pkg or SIG_1.4.2_1522.pkg. Our example file has been named `tp_rules.sig` so that exact versioning cannot be identified. Below is a screenshot of `tp_rules.pkg` loaded in a hex editor⁹.

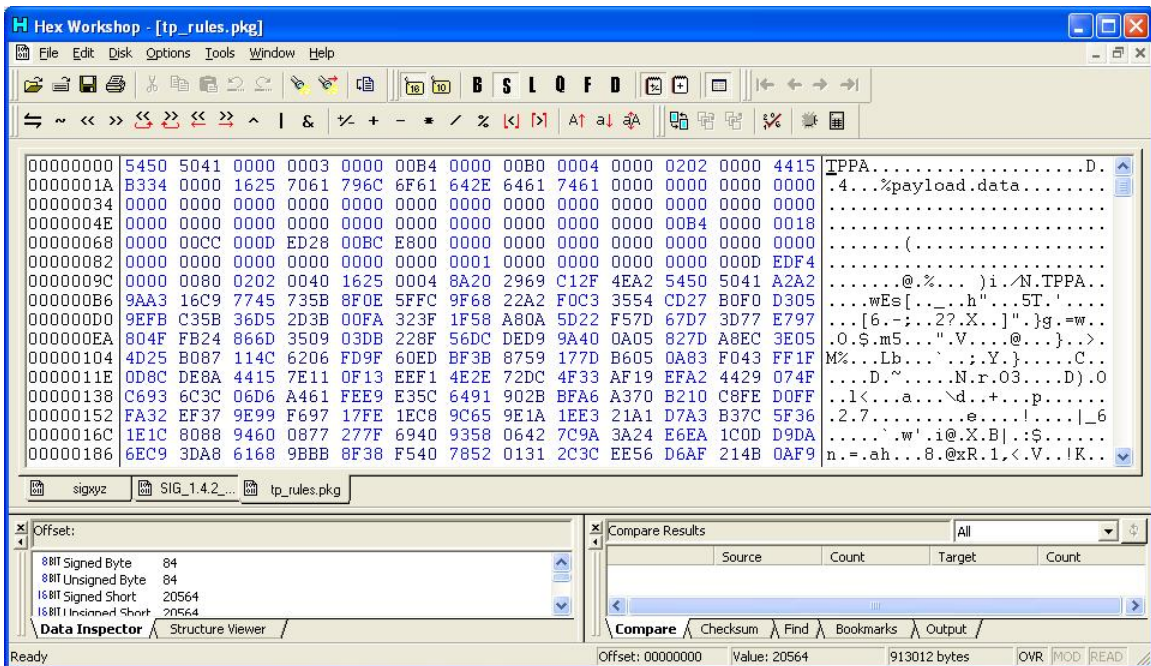


Figure 1 Hex editing the TippingPoint signature rule update file.

Several things are noticeable in this file. It is not a regular compressed archive and after offset 0xb4 the byte distribution looks random enough to suggest it is encrypted. There is what looks like two tags in the file, TPPA. Since one of the tags is the first 4 bytes of the file it is safe to assume this is some sort of magic number. There is also a string `payload.data` which may indicate that a process reads the signature file in, parses some type of file header, extracts a certain amount of the file and decrypts the rest of the data (that contains the signatures). The size of the file can also give clues. This file is 891KB, which could be a few different things. It could mean that the file contains the entire rule set and is compressed or that TippingPoint only does incremental updates by sending just the rules that have been added or changed. The file extension is generic enough, `.pkg`, that it does not actually help in the evaluation.

Since it is unlikely any more information can be gained from just the rule file moving to an evaluation of the actual TippingPoint hardware and file system is the next step. Since a TippingPoint box is pretty much a basic PC with a custom network processing board attached, no complex methods will be required to read the disk image. This means that the OS is installed on a standard IDE hard drive that can be cloned or analyzed by any forensics tool. At this stage a copy of the TippingPoint hard drive is made using a Linux machine and the “`dd`” command.

The TippingPoint IPS runs on VxWorks¹⁰, an embedded operating system, but the file system can still be read by most Linux distributions and Mac OS X. (Many products, such as McAfee’s Intruvert, is similarly based on VxWorks, while other product, such as ISS-IBM Proventia, are based on Linux). After the copy 4 partitions are discovered each containing somewhat familiar looks files and directories. The partitions are `/boot`, `/opt`, `/usr`, and `/log`. The most obvious thing to do at this point is to search the

partitions for keywords that could reveal where the signatures are kept. Words like “signature”, “rules”, “*.pkg”, “message”, and “logs”. Most of these searches will yield nothing but a message file is found on the 4th partition and the rule file is found on the 3rd partition. The rule file is found in /usr/vaccine/dv/ but it is still encrypted. On the same partition a likely location is found, /usr/usdm/ver2/base, that contains files that would support the signature file but none actually contain any signatures.

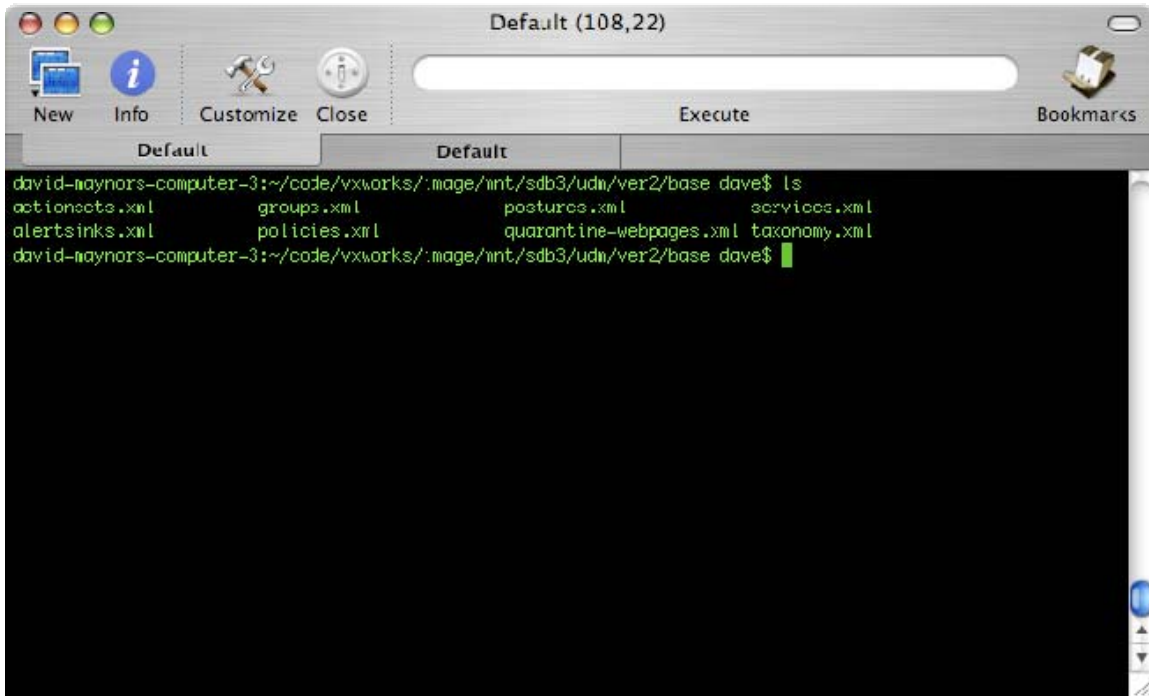


Figure 2 Possible location of signature file

Analysis of the log file reveals more information that will be used in disassembly later. It shows the logging information for an update including the extraction of payload.data, its verification, processing of the rules, and deleting of the intermediate files. There are plenty of logging strings that can be used to step through a binary and trace what is being done to the file at what point. VxWorks has a single binary that contains all of the executable code for the operating system and applications. This binary can be found in the /boot directory in the current version directory and called vxWorks. This file needs to be disassembled and analyzed. IDA Pro¹¹ is the tool used for this as it can support many different processors and binary types. IDA Pro does not have a specific target for VxWorks so a lot of the automated analysis will not be performed so a lot of the binary will have to be analyzed by hand.

```
15098      1142432042 0349999986 <DBG3> [HTP] 0000 calling install with --sig_... .pkg--
15099      1142432042 0399999984 <INFO> [UP ] 2004 Begin Package Install using /opt/tmp/download/i
image/sig_
.pkg
15100      1142432042 0399999984 <DBG0> [UP ] 0000 Changing UpdateError to OK - No Error [1]
15101      1142432042 0399999984 <DBG0> [UP ] 0000 Changing UpdateState to Updating [2]
15102      1142432042 0416666650 <DBG0> [UP ] 0000 Changing UpdateStateQualifier to OK [1]
15103      1142432042 0433333316 <INFO> [UP ] 2015 DV update to
15104      1142432042 0533333312 <INFO> [UP ] 2021 Extracting payload /opt/update/sec/
/p
payload.data from pkg /opt/tmp/download/image/
.pkg
15105      1142432048 0083333330 <INFO> [UP ] 4006 Replacing saved Digital Vaccine pkg with: /opt/
tmp/download/image/
.pkg
15106      1142432048 0083333330 <DBG0> [UP ] 0000 upManifestRemoveFiles(/usr, /usr/dv-manifest)
15107      1142432048 0133333328 <DBG0> [UP ] 0000 upVaccinePkgRemove(dv)
15108      1142432049 0016666666 <DBG0> [UP ] 0000 upVaccineProcess(dv, /opt/update/sec/
, /opt/update/sec/
/payload.data
)
15109      1142432049 0099999996 <INFO> [UP ] 2801 Extracting /opt/update/sec/
/payload.d
ata to /usr
15110      1142432053 0733333304 <DBG0> [UP ] 2701 Validating dir: /usr, using manifest: /usr/mani
fest
15111      1142432054 0783333302 <INFO> [UP ] 2803 upTEaV: Successful extraction/validation of /op
t/update/sec/
/payload.data
```

Figure 3 /log/sys/message.log

When the vxworks file is initially loaded a choice is given between loading it as a binary file or an a.out file. Loading it as an a.out file will get a lot of analysis but it will also get a lot of things wrong because a lot of guesses are being made. In order to help IDA analyze as much as possible the correct loading offset for the VxWorks image must be found. The offset can be found by examining the file in a hex editor and looking for a file header.

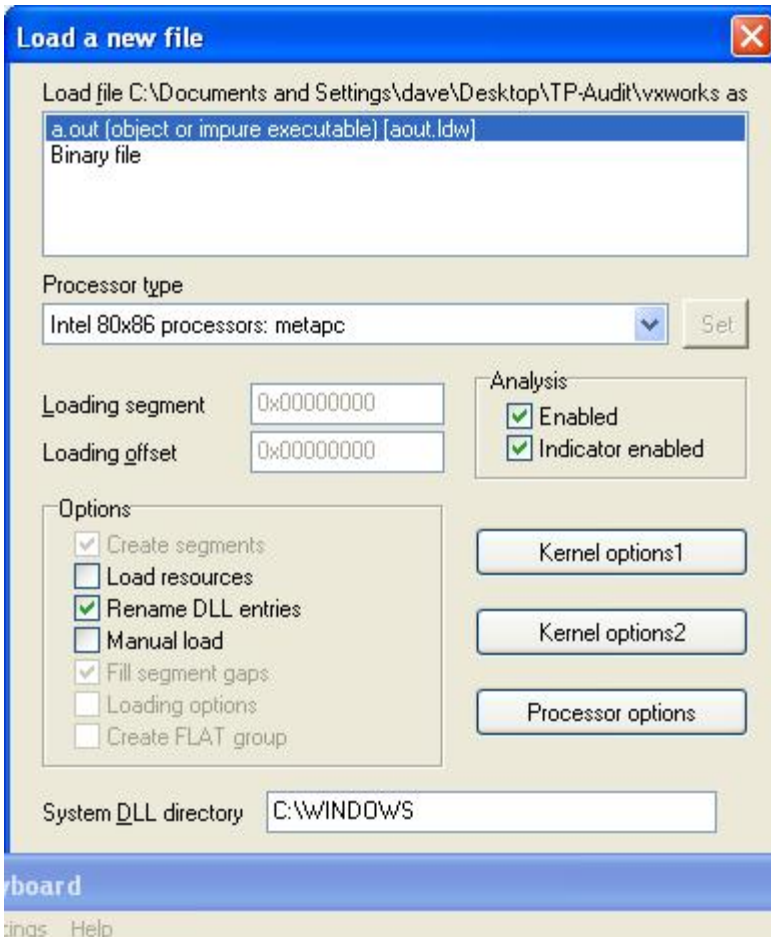


Figure 4 IDA Pro binary load screen

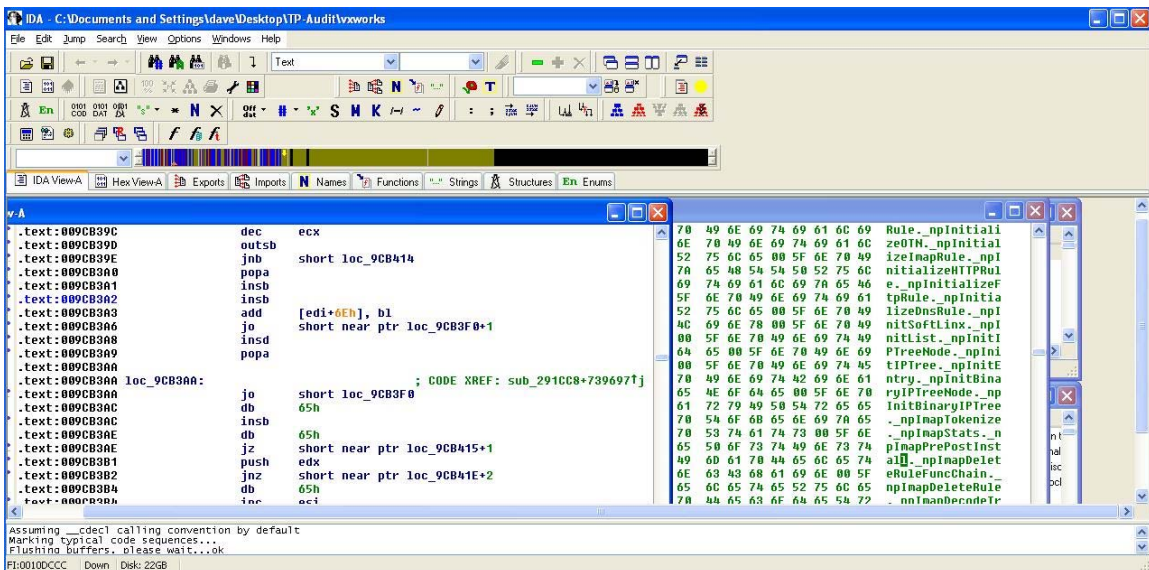
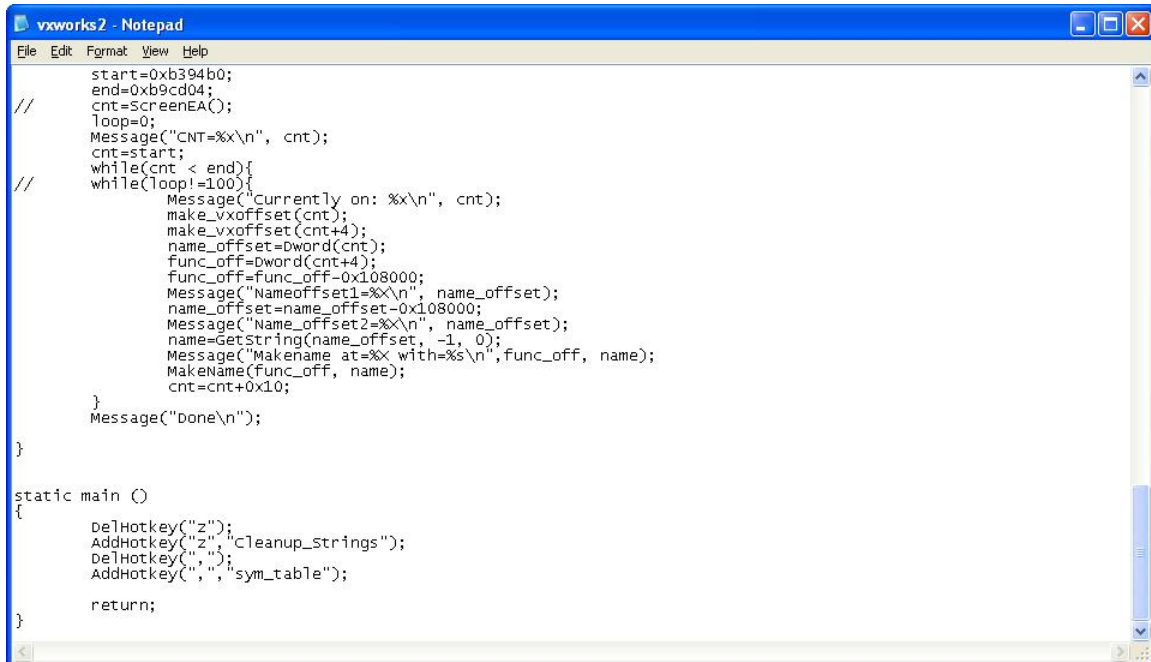


Figure 5 IDA Pro incorrectly disassembling strings and assembly

After examining the boot loader, called bootrom in /boot/<current version>/rom the loading offset appears to be read from offset 0x14 of the VxWorks file. The value found

there is 0x108000. This is used to rebase the image in IDA Pro and start another analysis of the binary with increased accuracy in the results. The results are still far from optimal with problems like strings not being detected correctly. Searching through the binary will reveal strings for standard C programming function names, located adjacently. After these strings is what looks like a symbol table. IDA Pro is flexible enough to support a scripting language called IDC which can be used to automate a lot of the cleanup tasks.



```
start=0xb394b0;
end=0xb9cd04;
cnt=ScreenEA();
//
loop=0;
Message("CNT=%x\n", cnt);
cnt=start;
//
while(cnt < end){
while(loop!=100){
    Message("Currently on: %x\n", cnt);
    make_vxoffset(cnt);
    make_vxoffset(cnt+4);
    name_offset=dword(cnt);
    func_off=dword(cnt+4);
    func_off=func_off-0x108000;
    Message("Nameoffset1=%x\n", name_offset);
    name_offset=name_offset-0x108000;
    Message("Name_offset2=%x\n", name_offset);
    name=GetString(name_offset, -1, 0);
    Message("Makename at=%x with=%s\n",func_off, name);
    MakeName(func_off, name);
    cnt=cnt+0x10;
}
    Message("done\n");
}

static main ()
{
    DelHotkey("z");
    AddHotkey("z", "cleanup_strings");
    DelHotkey(",");
    AddHotkey(",", "sym_table");

    return;
}
```

Figure 6 IDC script to fix strings and symbol table

In addition to the symbol table the auto-analysis misses detection of numerous functions. Since this VxWorks software was designed to run on a Intel x86 processor it has a standard function prologue of 0x55/0x89/0xE5 in hex or push ebp, mov ebp, esp in assembly. Creating another IDC script that can recognize the prologue and mark functions accordingly will shorten the analysis time. The scripts created for this demonstration are simple with hardcoded offsets for the strings and symbol table. These hardcoded offsets need to be changed for new versions of the image.

When the VxWorks image has undergone as much analysis and cleanup as possible locating the strings from the message.log file will be possible allowing for a trace of the update process. The first string that looks like it is exclusively part of the installer is "calling install with <file name>--". This string is used in the function _upPkgTask. Understanding what this function does with the rule file, how the information is processed, and what values are important in the file header is essential in decrypting the rule set.

A screenshot of a Notepad window titled 'pkg_hdr.hsl - Notepad'. The window contains a C structure definition for 'pkg_hdr'. The code is as follows:

```
#pragma byteorder(big_endian)
struct pkg_hdr
{
    DWORD Magic;
    DWORD version;
    DWORD size;
    DWORD unknw1;
    DWORD unknw2;
    DWORD unknw3;
    DWORD unknw4;
    DWORD unknw4_m;
    int name[16];
    DWORD key_offset;
    DWORD key_len;
    DWORD payload_data_start;
    DWORD unknw6;
    DWORD unknw7;
    DWORD unknw8;
    DWORD unknw9;
    DWORD unknw10;
    DWORD unknw11;
    DWORD unknw12;
    DWORD unknw13;
    DWORD unknw14;
    DWORD unknw15;
    DWORD unknw16;
    DWORD unknw17;
    DWORD unknw18;
    DWORD unknw19;
    DWORD unknw20;
    DWORD unknw21;
    DWORD unknw22;
    DWORD Magic_tail;
};
```

Figure 7 A structure definition of the TippingPoint rule file header for Hex Workshop

Using a new found understanding of the rule file header a type library can be built for a hex editor that shows what variables have what values. This is similar to defining a structure in C programming. Some of the interesting things about the file header include the `payload.data` string being used as the name of the output file. The three important static offsets are at fixed locations in the file. These offsets are for the position of the encryption key, its length, and the position of payload. The actual encryption/decryption is handled by OpenSSL and requires a certificate called `signer.pem` which can be found in the `/boot/ssl` directory of the disk image.

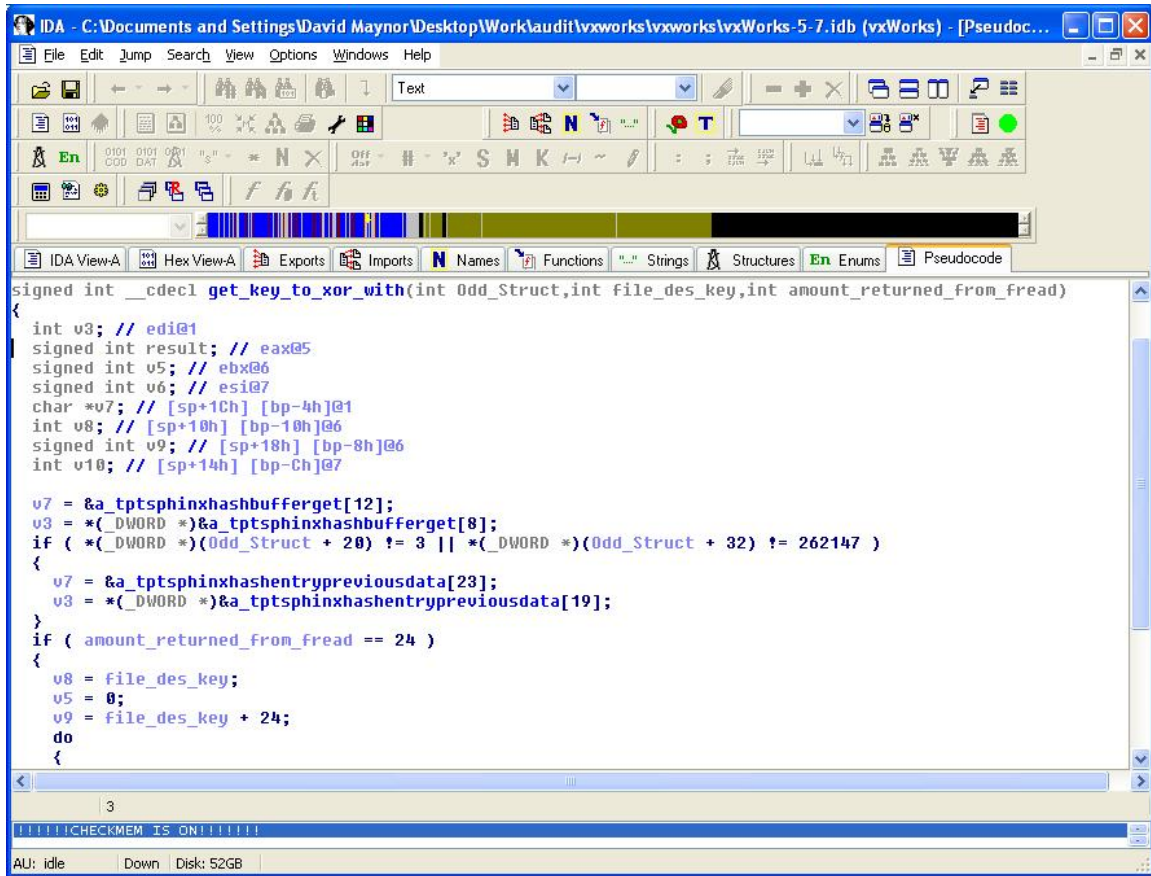


Figure 9 A decompilation of reading a key from the signature update file using Hex Rays¹²

The actual extraction, decryption, and decompression happens in a specific function. Calling this function from a standalone program will yield the ability to decrypt and untar the resulting archive. The archive contains a directory structure of `usdm/ver2/base` that contains a number of files, one of them is `signatures.xml`.


```
-/tp-audit
[dave@frontend tp-audit]$ ./tp-decrypt -o tp.pkg
ZDI Rule Decrypt      Errata Security
Opening file: tp-rules.pkg
Verifying package header.....GOOD
Get key offset: b4
Get key length: 18
Get payload.data offset: cc
Set up EVP_Digest for md5..done
Looping until decrypted: Done

You can now untar the file
[dave@frontend tp-audit]$ tar xvf tp.tar
manifest
udm/ver2/base/actionsets.xml
udm/ver2/base/alertsinks.xml
udm/ver2/base/policies.xml
udm/ver2/base/signatures.xml
udm/ver2/base/services.xml
udm/ver2/base/postures.xml
udm/ver2/base/groups.xml
udm/ver2/base/taxonomy.xml
udm/ver2/base/quarantine-webpages.xml
[dave@frontend tp-audit]$
```

Figure 11 decrypting the rules and extraction

```
dave@frontend: ~/tp-audit/udm/ver2/base
<header>
  <tptsig/>
  <number>      </number>
  <description>
This filter provides protection against exploitation of a zero-day vulnerability
.

This vulnerability was either discovered internally by the TippingPoint Security
Research Team (TSRT) or disclosed through the Zero Day Initiative (ZDI). The ZD
I aims to responsibly surface currently undisclosed (zero-day) vulnerabilities.
The vendor has been notified of the issue and we are working together to coordin
ate a public disclosure and patch release date. Until that time, the details of
this issue will remain confidential in accordance with our vulnerability disclos
ure policy.

For more information on the Zero Day Initiative please visit:

http://www.zerodayinitiative.com
  </description>
  <author>TippingPoint Technologies</author>
  <message>      : ZDI-CAN-      : Zero Day Initiative Vulnerability</message>
  <category>5</category>
  <cve/>
244753,48      88%
```

Figure 12 A canned description for an undisclosed ZDI rule.

TippingPoint triggered on a number of interesting items. Two strings in particular were “QTPointerRef” and “getSize”.

Googling these strings leads us to a lot of explanations about where the vulnerability is. One interesting page is this one from Apple. Notice all the buffer copy operations. http://developer.apple.com/documentation/Java/Reference/1.4.1/Java141API_QTJ/quicktime/util/QTPointerRef.html

What we see here in the vulnerability is that QuickTime has a Java wrapper around internal QuickTime functions written in C. While Java protects against overflows in its own buffers, it cannot protect against overflows in the C functions that it calls. The underlying functions that it calls provide little or no protection against buffer overflows.

Google also gives us a number of code fragments to work from. We can take those code fragments. The TippingPoint signature also gives us the value 0x7fffffff, which is also a strong hint to us that we should be creating integer overflows in values sent into copyToArray.

Some curious things come out of this. The first is that the TippingPoint signature is specific to the exploit. Changing one character in the exploit, for example, will bypass detection. Second, it appears that this signature could easily trigger on a lot of QuickTime objects that contain Java code. That’s probably a good thing, though, because, it appears that the Java interface is too powerful. Indeed, Apple appears to have shipped multiple patches after the CanSecWest contest.

Other 0day nexuses

We’ve had a chance to look at many defensive security products.

Some products are “flat” pattern-match systems (they do little decoding of protocols other than the TCP/IP stack), and others are based on protocol-parsers. They still often look for regex patterns, but those patterns are tightly restricted to a certain context. In other words, whereas Snort has the “uricontent” for searching for a pattern in the context of an HTTP URL, these products will have potentially hundreds of contexts within which to search for patterns.

The Intruvert product from McAfee is itself interesting. Their protocol parsers are expressed in an XML-based state-machine language. This gives more accurate detection than a pattern-matching product using regex, but at the same time, it more accurately tells the hacker how to build a packet that will trigger a vulnerability.

This work has interesting implications for an open-source product like Snort. A long standing criticism of Snort was that vendors could provide “private” content that hackers could not see, in advance of official announcement of vulnerabilities. Instead, since all vendors can have their signatures cracked, the openness of Snort’s signatures is less of a disadvantage.

We have also spent some time with anti-virus vendors. Most provide their signature updates either as an unencrypted file, or a file that is simply obfuscated. Unfortunately, we haven't found anything useful to do with that information. AV vendors trigger on patterns found within malware, unfortunately, those patterns can't be used for much of anything other than identifying that malware.

There are three popular hacking toolkits: CORE Impact¹³, ImmunitySec CANVAS¹⁴, and HD Moore's Metasploit¹⁵. The first two are commercial, and last is open-source. The commercial vendors develop and buy 0day exploits. Other people sell their own 0day exploits as plug-ins for these tools.

These toolkits are quite large, including a lot of library code (such as RPC stacks) as well as interpreted languages like Ruby and Python. They are too large for malware hackers, such as those who create botnets. However, we have found cases where botnet writers have extracted exploits from the toolkits and rewritten them in a small C module (such as capturing the packets, then blindly replaying them over TCP).

How to protect your signatures

We feel a good technique for protecting signatures is to “pre-compile” them.

Many products, such as the one shown above, send PCRE regular expressions to the customer. The customer's product then compiles them. Instead, the vendor can run the PCRE compilation step at the factory, then send the binary “blob” to the customer. Techniques for doing this are published on <http://www.pcre.org>.

Again, hackers could defeat this in theory, but it would be much, much harder.

Conclusion

In this presentation we have shown that 0day exploit information can easily be harvested from security products. Many defensive vendors include 0day information as part of their offerings. While security vendors attempt to encrypt or obfuscate such information, standard reverse engineering tools can be used to decrypt it. Once decrypted, that information can often be turned into weaponized exploits. While we use TippingPoint as an example, any vendor who ships 0day has the same type of risk. With broader knowledge of this problem, we hope that all vendors will take additional steps to prevent disclosure

.

¹ IBM Internet Security Systems - <http://xforce.iss.net/>

² Symantec

http://www.symantec.com/enterprise/security_response/weblog/security_response_blog/vulnerabilities_exploits/

³ eEye - <http://research.eeye.com/>

⁴ 0day - A undisclosed or unpatched flaw in software that can cause unintended consequences like additional levels of access or higher privileges

⁵ Cyber warfare - <http://en.wikipedia.org/wiki/Cyber-warfare>

⁶ Cyberoperator - <http://taosecurity.blogspot.com/2007/06/hope-for-air-force-cyberoperators.html>

⁷ Disclosure: We used to work for ISS.

⁸ 3com Zero Day Initiative - <http://www.zerodayinitiative.com/>

⁹ Hex Workshop - <http://www.hexworkshop.com/>

¹⁰ VxWorks - <http://www.windriver.com/vxworks/>

¹¹ IDA Pro - <http://www.datarescue.com/idabase/>

¹² Hex Rays - http://hexblog.com/2007/04/decompilation_gets_real.html#more

¹³ Core Security - <http://www.coresecurity.com/>

¹⁴ Immunity Sec - <http://www.immunitysec.com/products-canvas.shtml>

¹⁵ Metasploit - <http://www.metasploit.com/>