Malware-Focused Network Signatures

CYBR473 – Malware and Reverse Engineering (2024/T1)

Lecturers: Arman Khouzani, Alvin Valera

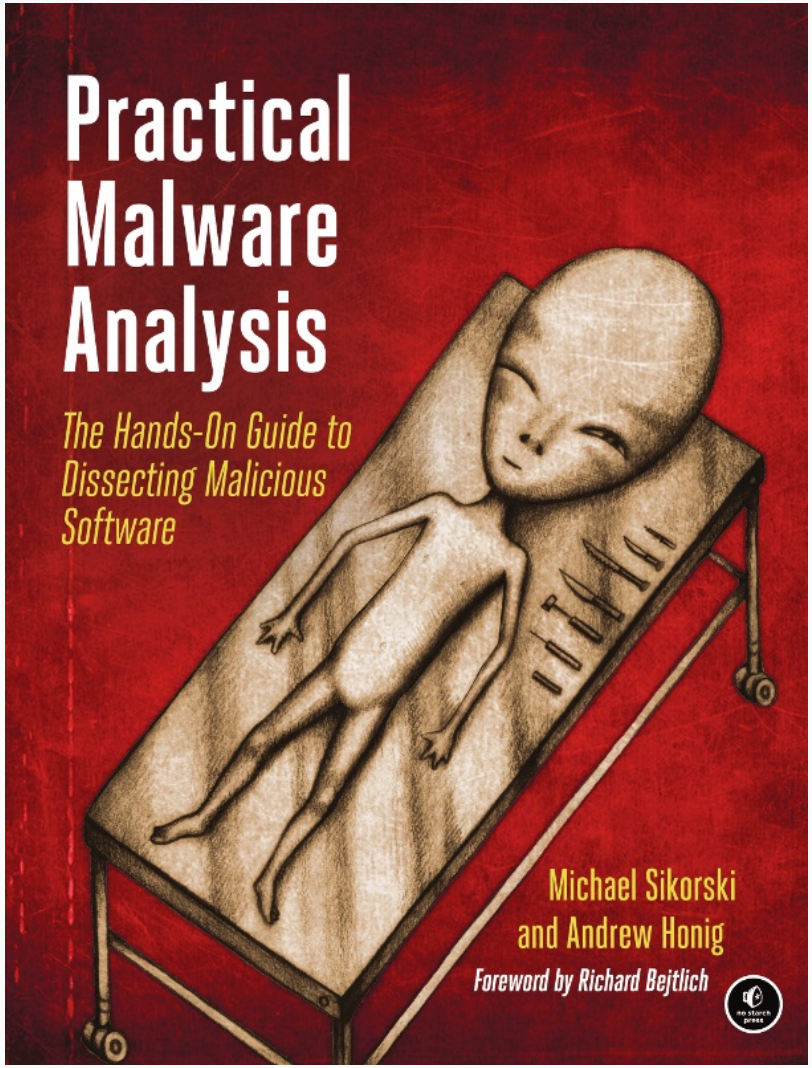
Victoria University of Wellington – School of Engineering and Computer Science

Table of contents

Reading Material

- Part IV: Malware Functionality
  - Ch.14: Malware-Focused Network Signatures

“Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software”, Michael Sikorski and Andrew Honig, 2012



Objectives

Malware makes use of network connectivity.

- Weak point for malware if we control ingress and egress

Countermeasures

- Detect or prevent malicious activity
- Need to understand malware use of networks
- Challenges faced by malware authors
- How these can be used

Network Countermeasures

Network Countermeasures

Safely Investigate an Attacker Online

Content-Based Network Countermeasures

Combining Dynamic and Static Analysis Techniques

Understanding the Attacker’s Perspective

Malware Use of Network

Malware uses network for:

- downloading additional malware
- exfiltration of stolen data
- *command and control* – a.k.a. *C2* – communication: retrieval or sending of instructions to trigger specific functions
- lateral movement (exploring the network to progressively find targets and subsequently gaining access to it once having an initial foothold)

Network Countermeasures

Based on basic network indicators (first to be analysed):

- Firewalls and routers can be used to restrict access to a network based on IP addresses and ports.
- DNS servers can be configured to reroute known malicious domains to an internal host, known as a *sinkhole*.
- Proxy servers can be configured to detect or prevent access to specific domains.

Network Countermeasures

Content-based countermeasures, enabled e.g. by intrusion detection systems (IDS), intrusion prevention systems (IPS), email/web proxies.

Difference between IDS/IPS:

- IDS is designed to merely *detect* the malicious traffic
- IPS is designed to *detect* malicious traffic and *prevent* it from travelling over the network.

Signatures are used to detect more than just intrusions:

- scanning
- service enumeration and profiling
- nonstandard use of protocols
- beaconing from installed malware

Observing the Malware in Its Natural Habitat

The best way to start network-focused malware analysis is to mine the logs, alerts, and packet captures that were already generated by the malware (coming from real networks, rather than from a lab environment). Because:

- Live-captured info provide a view of a malware's true behaviour, as malware can be programmed to detect lab environments.
- Real traffic provides info at both ends (client and server) whereas in a lab environment, we typically have access to only one end.
- When passively reviewing information, there is no risk that your analysis activities will be leaked to the attacker.

OPSEC = Operations Security

OPSEC, a term used by the government and military, describes a process of preventing adversaries from obtaining sensitive info.

- Certain actions of malware investigators can inform the malware author that they have identified the malware, or may even reveal personal details of the investigator to the attacker.
- If attackers are aware that they are being investigated, they may change tactics and effectively disappear.

OPSEC

How malware can find out that it is investigated:

- Send a targeted phishing (known as spear-phishing) email with a link to a specific individual and watch for access attempts to that link from IP addresses outside the expected geographical area.
- Create an encoded link in a blog comment, effectively creating a private but publicly accessible infection audit trail.
- Embed an unused domain in malware and watch for attempts to resolve the domain.

Safely Investigate an Attacker Online

Network Countermeasures

Safely Investigate an Attacker Online

Content-Based Network Countermeasures

Combining Dynamic and Static Analysis Techniques

Understanding the Attacker’s Perspective

Indirection Tactics

- Use Tor, an open proxy, or a web-based anonymizer for anonymity, although it may give a clue that you are trying to hide
- You can hide the precise location of a dedicated machine in several ways, such as the following:
  - using a cellular connection
  - tunnelling via SSH or a VPN through a remote infrastructure
  - using an ephemeral VM in a cloud service, such as Amazon EC2

Getting IP Address and Domain Information

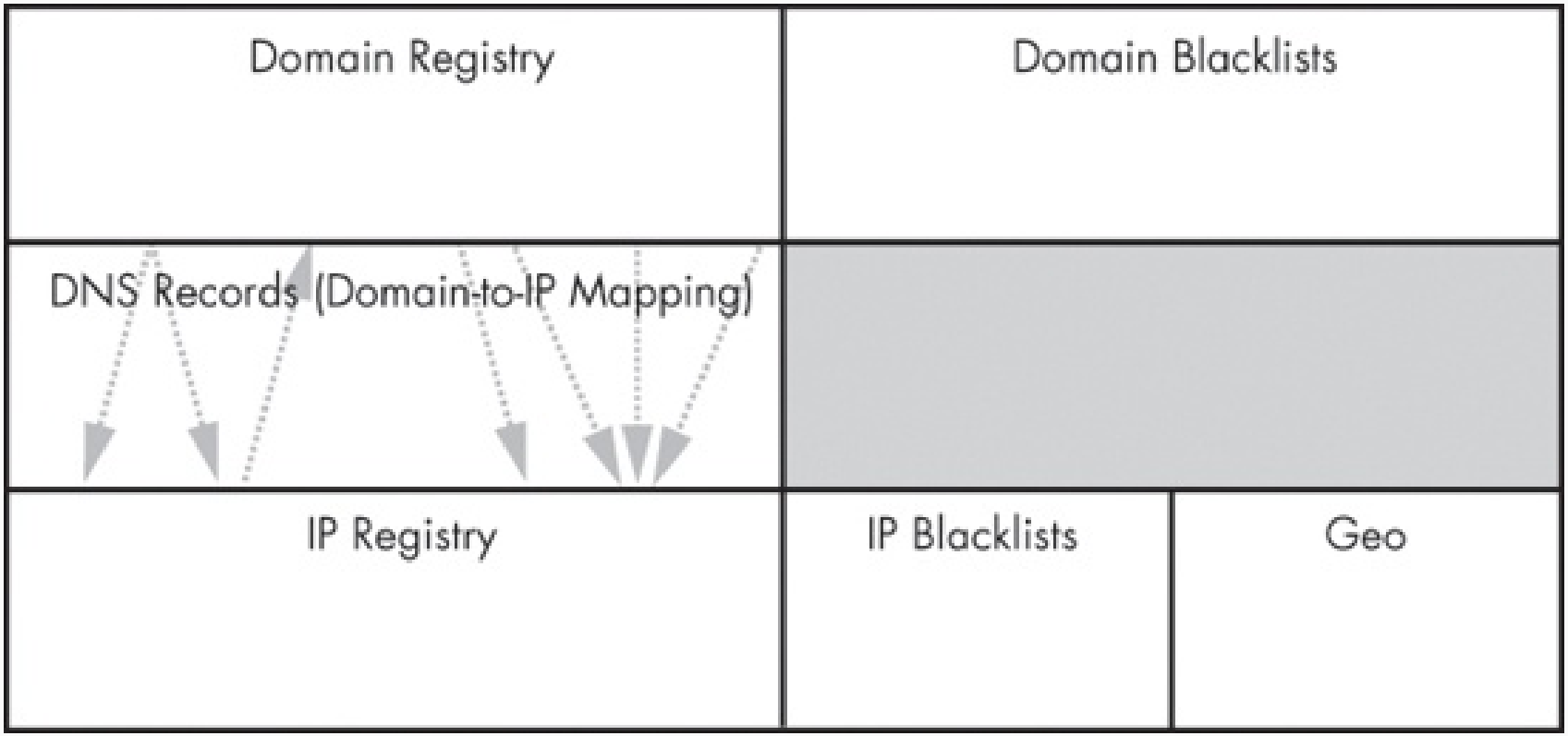
DNS translates domain names like `www.yahoo.com` into IP addresses (and back).

Malware also uses DNS to maintain flexibility and robustness when hosting its malicious activities.

When a domain name is registered, the domain, its name servers, relevant dates, and contact information for the entity who registered the name is stored in a domain registrar.

DNS information represents the mapping between a domain name and an IP address. Additionally, metadata is available, including blacklists (which can apply to IP addresses or domain names) and geographical information (which applies only to IP addresses).

Getting IP Address and Domain Information



Types of information available about DNS domains and IP addresses

Getting IP Address and Domain Information

While both of the domain and IP registries can be queried manually using command-line tools, using websites has several advantages:

- Many will do follow-on lookups automatically.
- They provide a level of anonymity.
- They provide additional metadata based on historical info or other sources, including blacklists and geographical information.

Three lookup sites deserve special mention:

- **DomainTools (`www.domaintools.com/`)**
  Provides historical **`whois`** records, reverse IP lookups showing all the domains that resolve to a particular IP address, and reverse whois, allowing lookups based on contact information metadata.
- **RobTex (`www.robtex.com/`)**
- **BFK DNS logger (`www.bfk.de/bfk_dnslogger_en.html`)**
  Uses passive DNS monitoring data. However “Due to the EU GDPR policy, this service has been shut down until further notice.”

Content-Based Network Countermeasures

Network Countermeasures

Safely Investigate an Attacker Online

Content-Based Network Countermeasures

Combining Dynamic and Static Analysis Techniques

Understanding the Attacker’s Perspective

Content-Based Network Countermeasures

Basic indicators such as IP and domain names can be valuable for defending against a specific version of malware, but attackers are adept at quickly moving to different addresses or domains.

Indicators based on content tend to be more valuable and longer lasting, since they use more fundamental characteristics of malware.

Signature-based IDSs are the oldest and most commonly deployed systems for detecting malicious activity via network traffic. IDS detection depends on knowledge about what malicious activity looks like, i.e., its signature, to detect it when it happens again.

An ideal signature can send an alert every time something malicious happens (true positive), but will not create an alert for anything that looks like malware but is actually legitimate (false positive).

Intrusion Detection with Snort

One of the most popular IDSs is called **Snort**.

Snort is used to create a signature or rule that links together a series of elements (called *rule options*) that must be true before it fires.

- *payload rule options*: identify content elements
- *nonpayload rule options*: identify elements that are not content related, e.g., certain flags, specific values of TCP or IP headers, size of the packet payload. Examples:
  - `flow:established,to_client` packets that are a part of a TCP session that originate at a server and are destined for a client.
  - `dsize:200`, which selects packets that have 200 bytes of payload.

Intrusion Detection with Snort

Let’s create a basic Snort rule to detect the initial malware example.

When browsers and other HTTP applications make requests, they populate a **User-Agent** header field. A typical value may look like **`Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)`**. This provides information about the version of the browser and OS.

The User-Agent used by our malware is **`Wefa7e`**, which is distinctive and can be used to create a signature:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
    (msg:"TROJAN Malicious User-Agent";
content:"|0d 0a|User-Agent\: Wefa7e";
    classtype:trojan-activity; sid:2000001; rev:1;)
```

Snort Rule's keywords and descriptions

Keyword	Description
`msg`	The message to print with an alert or log entry
`content`	Searches for specific content in the packet payload (see the discussion following the table)
`classtype`	General category to which rule belongs
`sid`	Unique identifier for rules
`rev`	With **`sid`**, uniquely identifies rule revisions

Intrusion Detection with Snort

Snort rules are composed of two parts:

- a rule **header**, and
- rule **options**.

The rule **header** contains the rule action (typically `alert`), protocol, source and destination IP, and source and destination ports.

Snort rules use variables to allow customization of its environment:

- `$HOME_NET` and `$EXTERNAL_NET` variables are used to specify internal and external network IP address ranges.
- `$HTTP_PORTS` ports that should be interpreted as HTTP traffic.

`$HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS` header matches outbound traffic destined for HTTP ports.

Within the **content** term, the pipe symbol `|` indicates the start and end of hexadecimal notation. Thus, `|0d 0a|` represents the break between HTTP headers. So `|0d 0a|User-Agent Wefa7e` matches the HTTP header field `User-Agent: Wefa7e`.

We have IP addresses to block at firewalls, a domain name to block at the proxy, and a network signature to load into the IDS. Stopping here, however, would be a mistake, since the current measures provide only a false sense of security.

A malware analyst must strike a balance between expediency and accuracy. For network-based malware analysis, the expedient route is to run malware in a sandbox and assume the results are sufficient. The accurate route is to fully analyze malware function by function.

Intrusion Detection with Snort: Taking a Deeper Look

Suppose that in fact we had seen two values for the User-Agent strings in real traffic: **`Wefa7e`** and **`Wee6a3`**.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"ET TROJAN WindowsEntSuite FakeAV Dynamic User-Agent";
flow:established,to_server;
content:"|0d 0a|User-Agent\: We"; isdataat:6,relative;
content:"|0d 0a|";
distance:0; pcre:"/User-Agent\: We[a-z0-9]{4}\x0d\x0a/";
classtype:trojan-activity;
reference:url,www.threatexpert.com/report.aspx?md5=
    d9bcb4e4d650a6ed4402fab8f9ef1387;
sid:2010262; rev:1;)
```

Intrusion Detection with Snort: Taking a Deeper Look

Additional Snort rule's keywords and descriptions.

Keyword	Description
`flow`	Specifies characteristics of the TCP flow being inspected, such as whether a flow has been established and whether packets are from the client or the server.
`isdataat`	Verifies that data exists at a given location (optionally relative to the last match)
`distance`	Modifies the **`content`** keyword; indicates the number of bytes that should be ignored past the most recent pattern match
`pcre`	A Perl Compatible Regular Expression that indicates the pattern of bytes to match
`reference`	A reference to an external system

Intrusion Detection with Snort: Taking a Deeper Look

The core of the rule is simply the User-Agent string where `We` is followed by exactly four alphanumeric characters (`We[a-z0-9]{4}`). In the Perl Compatible Regular Expressions (PCRE) notation used by Snort, the following characters are used:

- Square brackets (`[` and `]`) indicate a set of possible characters.
- Curly brackets (`{` and `}`) indicate the number of characters.
- Hexadecimal notation for bytes is of the form `\xHH`.

`flow:established,to_server;` ensures that the rule fires only for client-generated traffic within an established TCP session.

Intrusion Detection with Snort: Taking a Deeper Look

However, this rule creates false positives associated with the use of the popular **`Webmin`** software. So the rule is modified to the following:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"ET TROJAN WindowsEntSuite FakeAV Dynamic User-Agent";
flow:established,to_server;
content:"|0d 0a|User-Agent|3a| We"; isdataat:6,relative;
content:"|0d 0a|"; distance:0;
content:!"User-Agent|3a| Webmin|0d 0a|";
pcre:"/User-Agent\: We[a-z0-9]{4}\x0d\x0a/";
classtype:trojan-activity;
reference:url,www.threatexpert.com/report.aspx?md5=
    d9bcb4e4d650a6ed4402fab8f9ef1387;
reference:url,doc.emergingthreats.net/2010262;
reference:url,www.emergingthreats.net/cgi-bin/
    cvsweb.cgi/sigs/VIRUS/TROJAN_WindowsEnterpriseFakeAV;
sid:2010262; rev:4;)
```

Intrusion Detection with Snort: Taking a Deeper Look

`content:!"User-Agent|3a| Webmin|0d 0a|"`: bang symbol `!` means the rule triggers only if the content described is not present.

This example illustrates several attributes typical of the signature-development process:

- Most signatures are created based on analysis of the network traffic, rather than the malware that generates the traffic.
- The uniqueness of the pattern is tested by running the signature across real traffic to ensure that it is free of false positives.

To ensure that we have a more robust sample, we can repeat the dynamic analysis of the malware many times.

Intrusion Detection with Snort: Taking a Deeper Look

Let's imagine the following list of User-Agent strings are generated:

```
We4b58  We7d7f  Wea4ee  We70d3  Wea508  We6853  We3d97
We8d3a  Web1a7  Wed0d1  We93d0  Wec697  We5186  We90d8
We9753  We3e18  We4e8f  We8f1a  Wead29  Wea76b  Wee716
```

The results appear to use a smaller character set than specified by **`/User-Agent We[a-z0-9]40d0a/`**: the results suggest that the characters are limited to **`a-f`** rather than **`a-z`**. This suggests that somewhere binary values are converted to hex representations.

As an additional thought experiment, imagine that the results from multiple runs resulted in the following User-Agent strings instead:

```
WWfbcc5 Wf4abd  Wea4ee  Wfa78f  Wedb29  W101280 W101e0f
Wfa72f  Wefd95  Wf617a  Wf8a9f  Wf286f  We9fc4  Wf4520
Wea6b8  W1024e7 Wea27f  Wfd1c1  W104a9b Wff757  Wf2ab8
```

Intrusion Detection with Snort: Taking a Deeper Look

So, the signature is not ideal given the User-Agent can start with `Wf` and `W1` too, in addition to `We`, and could be seven characters long.

So, dynamically generating additional samples allows an analyst to make more informed assumptions about the underlying code.

It is also helpful to have at least two systems generating sample traffic, as malware may use some host attribute as an input.

E.g., multiple runs on hosts 1 and 2 may produce the following:

```
Wefd95 Wefd95 Wefd95 Wefd95 Wefd95 Wefd95 Wefd95 Wefd95 Wefd95
```

```
We9753 We9753 We9753 We9753 We9753 We9753 We9753 We9753 We9753
```

Combining Dynamic and Static Analysis Techniques

Network Countermeasures

Safely Investigate an Attacker Online

Content-Based Network Countermeasures

Combining Dynamic and Static Analysis Techniques

Understanding the Attacker’s Perspective

Combining Dynamic and Static Analysis Techniques

So far, we have been using either existing data or output from dynamic analysis to inform the generation of our signatures.

While such measures are expedient and generate information quickly, they sometimes fail to identify the deeper characteristics of the malware that can lead to more accurate and longer-lasting signatures.

In general, there are two objectives of deeper analysis:

- **full coverage of functionality:** involves providing new inputs (in dynamic analysis) so that the code continues down unused paths, to determine what the malware is expecting to receive.
- **understanding functionality, including inputs and outputs:** static analysis can be used to see where and how content is generated, and to predict the behaviour of malware. Dynamic analysis can then be used to confirm the expected behaviour.

The Danger of Overanalysis

If the goal of malware analysis is to develop effective network indicators, then you don't need to understand every block of code.

Hierarchy of Malware Analysis Levels

Analysis level	Description
Surface analysis	An analysis of initial indicators, equivalent to sandbox output
Communication method coverage	An understanding of the code for each type of communication technique
Operational replication	The ability to create a tool that allows for full operation of the malware (a server-based controller, for example)
Code coverage	An understanding of every block of code

To develop robust network indicators, we must reach a level between "communication method coverage" and "operational replication".

Hiding in Plain Sight

Signatures should differentiate between regular traffic and malware's. This is a challenge: Malware has evolved to evade detection by trying to blend in with the background, using the following techniques.

- Mimicking Existing Protocols:
- Using Existing Infrastructure
- Leveraging Client-Initiated Beaconing

Hiding in Plain Sight: Attackers Mimic Existing Protocols

Mimicking Existing Protocols:

Attackers use the most popular communication protocols, so that their malicious activity is more likely to get lost in the crowd. E.g., it is difficult to monitor the large amount of traffic using HTTP, HTTPS, and DNS. Also, they are less likely to be blocked, due to the potential consequences of accidentally blocking a lot of normal traffic.

Attackers often use HTTP for beaconing, as the beacon is basically a request for further instructions, like the HTTP GET request, and they use HTTPS to hide the nature and intent of the communications.

Hiding in Plain Sight: Attackers Mimic Existing Protocols

Attackers also abuse standard protocols, e.g., Malware attempting to pass a user's password could perform a DNS request for the domain `www.thepasswordisflapjack.maliciousdomain.com`.

Attackers can also abuse the HTTP standard. Since the GET method is intended for requests, it provides a limited amount of space for data (typically around 2KB). Spyware regularly includes instructions on what it wants to collect in the URI path or query of an HTTP GET, rather than in the body of the message.

In an observed malware, all information from the infected host was embedded in the User-Agent fields of multiple HTTP GET requests.

Hiding in Plain Sight: Attackers Mimic Existing Protocols

```
GET /world.html HTTP/1.1
User-Agent: %^&NQvtmw3eVhTfEBnzVw/aniIqQB6qQgTvmxJzVhjqJMjcHtEhI97n9+yy+duq+h3
b0RFzThrfE9AkK9OYIt6bIM7JUQJdViJaTx+q+h3dm8jJ8qfG+ezm/C3tnQgvVx/eECBZT87NTR/fU
QkxmgcGLq
Cache-Control: no-cache

GET /world.html HTTP/1.1
User-Agent: %^&EBTaVDPYTM7zVs7umwvhTM79ECrrmd7ZVd7XSQFvV8jJ8s7QVhcgVQOqOhPdUQB
XEAkgVQFvms7zmd6bJtSfHNSdJNEJ8qfGEA/zmwPtnC3d0M7aTs79KvcAVhJgVQPZnDIqSQkuEBJvn
D/zVwneRAyJ8qfGIN6aIt6aIt6cI86qI9mlIe+q+OfqE86qLA/FOtjqE86qE86qE86qHqfGIN6aIt6
aIt6cI86qI9mlIe+q+OfqE86qLA/FOtjqE86qE86qE86qHsjJ8tAbHeEbHeEbIN6qE96jKt6kEABJE
86qE9cAMPE4E86qE86qE86qEA/vmhYfVi6J8t6dHe6cHeEbI9uqE96jKtEkEABJE86qE9cAMPE4E86
qE86qE86qEATrnw3dUR/vmbfGIN6aINAaIt6cI86qI9ulJNmq+OfqE86qLA/FOtjqE86qE86qE86qN
Ruq/C3tnQgvVx/e9+ybIM2eIM2dI96kE86cINygK87+NM6qE862/AvMLs6qE86qE86qE87NnCBdn87
JTQkg9+yqE86qE86qE86qE86qE86bEATzVCOymduqE86qE86qE86qE86qE96qSxvfTRIJ8s6qE86qE
86qE86qE86qE9Sq/CvdGDIzE86qK8bgIeEXItObH9SdJ87s0R/vmd7wmwPv9+yJ8uIlRA/aSiPYTQk
fmd7rVw+qOhPfnCvZTiJmMtj
Cache-Control: no-cache
```

Attackers tunnel communications by misusing fields in a protocol to avoid detection. If defenders search the contents of the body of the HTTP session in our sample, for example, they won't see any traffic.

Hiding in Plain Sight: Attackers Mimic Existing Protocols

E.g.: Evolution of User-Agent field in malware mimicking a browser:

- First generation used completely manufactured strings.
- Next generation used a common value in real network traffic, e.g.:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
```

  While that made them blend in better, network defenders could still use a static User-Agent field to create effective signatures.
- Next, malware would switch between multiple values, all commonly used by normal traffic.
- The latest technique uses a native library call that constructs requests with the same code that the browser uses, so that its User-Agent string is indistinguishable from the browser's.

Hiding in Plain Sight: Attackers Use Existing Infrastructure

Using Existing Infrastructure:

The attacker may use a server that has many purposes.

The legitimate uses will obscure the malicious uses, since investigation of the IP address will also reveal the legitimate uses.

A more sophisticated approach is to embed commands for the malware in a legitimate web page. For example:

```
<!DOCTYPE html PUBLIC "[...]"> [...]
<head> [...]
<title>  Roaring Capital | [...]</title>
[...]
<!-- adsrv?bG9uZ3NsZWVw -->
<!--<script type="text/javascript" src="/js/dotastic.custom.js"></script>-->
[...]
```

There is indeed an encoded command for malware (in a comment): **`bG9uZ3NsZWVw`** is Base64 encoding of **`longsleep`**.

Hiding in Plain Sight: Leveraging Client-Initiated Beaconing

Leveraging Client-Initiated Beaconing:

One trend in network design is the increased use of Network Address Translation (NAT) and proxy solutions.

Attackers waiting for requests from malware have difficulty identifying which (infected) host is communicating, as all requests look like they are coming from the proxy IP address.

One common malware technique is to construct a profile of the victim machine and pass that unique identifier in its beacon.

A defender having found how the malware identifies distinct hosts can use that information to identify and track infected machines.

Understanding Surrounding Code

Suppose we have suspected the following request in our traffic logs:

```
GET /10119619177581151161015848102102102565653356 HTTP/1.1
Accept: * / *
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.badsite.com
Connection: Keep-Alive
Cache-Control: no-cache
```

Running the malware in our lab environment (or sandbox), we notice the following similar request:

```
GET /145862058658109971085848484853555255551 HTTP/1.1
Accept: * / *
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.badsite.com
Connection: Keep-Alive
Cache-Control: no-cache
```

Understanding Surrounding Code

Using Internet Explorer, we find that the User-Agent on this system is:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 2.0.50727; .NET CLR 3.0.04506.648)
```

Given the different User-Agent strings, it appears that this malware's User-Agent is hard-coded, but a common one (risk of false positives).

The next step is to perform **dynamic analysis** by running the malware a couple of times. The GET requests were the same except the URIs:

```
/10119619177581151161015848102102102565653 56 (actual traffic)
/14586205865810997108584848485355525551
/79115541725810997108584848485356541001 02
/23325115618458109971085848485357985255
```

There appears to be some common characters in the middle (**5848**).

Finding the Networking Code

Next, **static analysis** can be used to figure out exactly how the request is being created. The first step is to find the **system calls** that performs the communication.

An overview of the Windows API calls to implement networking functionality.

WinSock API	WinINet API	COM interface
`WSAStartup`	`InternetOpen`	`URLDownloadToFile`
`getaddrinfo`	`InternetConnect`	`CoInitialize`
`socket`	`InternetOpenURL`	`CoCreateInstance`
`connect`	`InternetReadFile`	`Navigate`
`send`	`InternetWriteFile`	
`recv`	`HTTPOpenRequest`	
`WSAGetLastError`	`HTTPQueryInfo`	
	`HTTPSendRequest`	

Finding the Networking Code

In our sample malware, suppose our static analysis shows that its imported functions include `InternetOpen` and `HTTPOpenRequest`, suggesting that the malware uses the **`WinINet API`**.

When we investigate the parameters to `InternetOpen`, we see that the User-Agent string is hard-coded in the malware.

Additionally, `HTTPOpenRequest` takes a parameter that specifies the accepted file types, which we see also hard-coded.

Another `HTTPOpenRequest` parameter is the URI path, and we see that the contents of the URI are generated from calls to `GetTickCount`, `Random`, and `gethostbyname`.

Knowing the Sources of Network Content

Creating an effective signature requires knowledge of the origin of network content. The following are the fundamental sources:

- *Random data*, e.g., from a call to a pseudo-random function.
- *Data from standard networking libraries*, e.g., the `GET` created from a call to `HTTPSendRequest`.
- *Hard-coded data*, e.g. a hard-coded User-Agent string.
- *Data about the host and its configuration*, e.g., the hostname, the current time according to the system clock, the CPU speed.
- *Data received from other sources, such as a remote server or the file system*, e.g. a nonce sent from server for use in encryption, a local file, keystrokes captured by a keystroke logger.

Note that there can be various levels of encoding imposed on this data prior to its use in networking, but its fundamental origin determines its usefulness for signature generation.

Hard-Coded Data vs. Ephemeral Data

Malware that uses lower-level networking APIs such as Winsock requires more manually generated content to mimic common traffic than malware that uses a higher-level networking API like the COM interface.

More manual content means more hard-coded data, which increases the likelihood that the malware author will have made some mistake that you can use to generate a signature.

The mistakes can be obvious, such as the misspelling of Mozilla (Mozila), or more subtle, such as missing spaces or a different use of case than is seen in typical traffic (MoZilla).

In our sample malware, a mistake exists in the hard-coded **`Accept`** string: it is statically defined as **`* / *`** instead of the usual **`*/*`**.

Hard-Coded Data vs. Ephemeral Data

Recall that the URI from our example has the following form:

```
/145862058658109971085848485355525551
```

The URI generation function calls **`GetTickCount`**, **`Random`**, and **`gethostbyname`**, and when concatenating strings, the malware uses the colon **`:`** character. The hard-coded **`Accept`** string and the hard-coded colons are good candidates for inclusion in the signature.

Suppose that our debugging shows that the function creates a string with the following format (before being sent to an encoding function):

```
<4 random bytes>:<first three bytes of hostname>:
    <time from GetTickCount as a hexadecimal number>
```

The encoding seems to take each byte and convert it to its ASCII decimal form (for example, the character a becomes 97).

Identifying and Leveraging the Encoding Steps

The `GetTickCount` results are hidden by two layers of encoding, first turning the binary `DWORD` value into an 8-byte hex representation, and then translating each of those bytes into its decimal ASCII value.

The final regular expression is as follows:

```
/\/([12]{0,1}[0-9]{1,2}){4}58[0-9]{6,9}58(4[89]|5[0-7]|9[789]|11[012]){8}/
```

<4 random bytes>	:	<first 3 bytes of host-name>	:	<time from GetTick-Count>
0x91, 0x56, 0xCD, 0x56	:	”m”, ”a”, ”l”	:	00057473
0x91, 0x56, 0xCD, 0x56	0x3A	0x6D, 0x61, 0x6C	0x3A	0x30, 0x30, 0x30, 0x35, 0x37, 0x34, 0x37, 0x33
1458620586	58	10997108	58	4848485355525551
(([1-9]|1[0-9]|2[0-5])0,1[0-9])4	58	[0-9]6,9	58	(4[89]|5[0-7]|9[789]|10[012])8

Creating a Signature

The following is the proposed Snort signature for our sample malware: a static User-Agent string, an unusual Accept string, an encoded colon (58) in the URI, a missing referrer, and a GET request matching the regular expression described previously.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"TROJAN Malicious Beacon ";
content:"User-Agent: Mozilla/4.0 (compatible\; MSIE 7.0\; Windows NT 5.1)";
content:"Accept: * / *"; uricontent:"58";
content:!"|0d0a|referer:"; nocase; pcre:"/GET
\/([12]{0,1}[0-9]{1,2}){4}58[0-9]{6,9}58(4[89]|5[0-7]|9[789]|10[012]){8}
HTTP/"; classtype:trojan-activity; sid:2000002; rev:1;)
```

Note: Our focus was on creating a signature that works. Optimizing a signatures to ensure good performance is also important.

Analyze the Parsing Routines

So far, we have discussed how to analyze the traffic that the malware generates, but information in the malware about the traffic that it receives can also be used to generate a signature.

As an example, recall this scenario: a malware makes a request for a web page at a site the attacker has compromised and search for the hidden message embedded (as an HTML comment) in the web page.

Analyze the Parsing Routines

When comparing the strings in the malware and the web page, we see that there is a common term in both: **`adsrv?`**. The web page that is returned has a single line that looks like this:

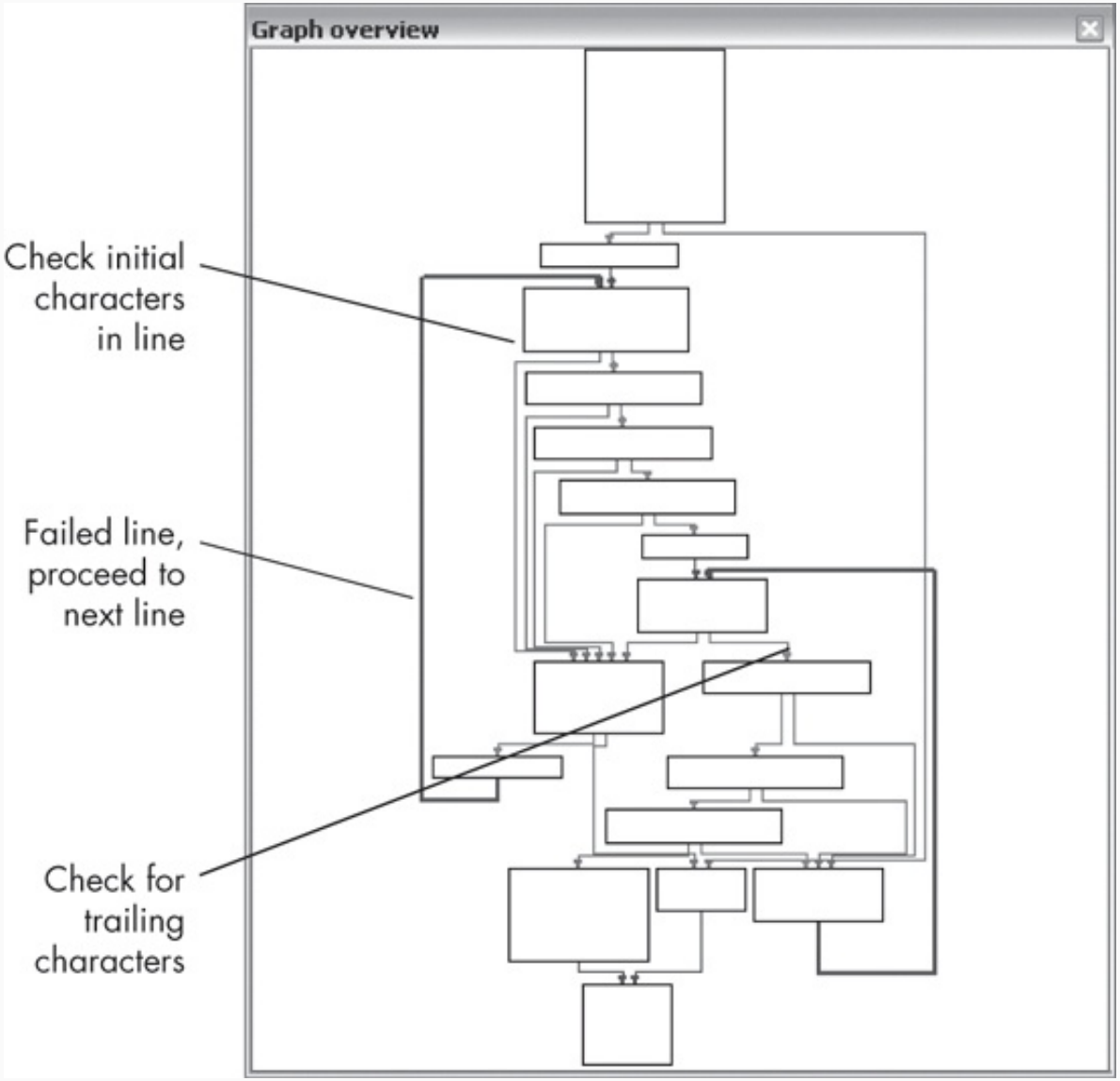
```
<!-- adsrv?bG9uZ3NsZWVw -->
```

It might be tempting to create a network signature based on the observed traffic, but doing so would result in an incomplete solution. First, two questions must be answered:

- What other commands might the malware understand?
- How does it identify that the web page contains a command?

We first look for the networking routine where the page is received and passed to the **parsing function**.

Analyze the Parsing Routines



The IDA Pro graph of a sample parsing function

Analyze the Parsing Routines

The design is typical of a custom parsing function, which is often used in malware instead of something like a regular expression library. Custom parsing routines are generally organized as a cascading pattern of tests for the initial characters.

This sample function has a double cascade and loop structure:

- the first loop checks for the `<!--` characters
- and the second looks for `-->` characters

In the block between the cascades, there is a function call that tests the contents that come after the `<!--`. Thus, the command will be processed only if the contents in the middle match the internal function and both sides of the comment enclosure are intact.

Analyze the Parsing Routines

Digging deeper into the parsing function, we find that it first checks that the **`adsrv?`** string is present.

The attacker places a command for the malware between the question mark and the comment closure, and performs a Base64 conversion of the command to provide rudimentary obfuscation.

The parsing function does the Base64 conversion, but it does not interpret the resulting command. The command analysis is performed later on in the code once parsing is complete.

Sample Malware Commands

Command example	Base64 translation
`longsleep`	`bG9uZ3NsZWVw`
`superlongsleep`	`c3VwZXJsb25nc2xlZXA=`
`shortsleep`	`c2hvcnRzbGVlcA==`
`run:www.example.com/fast.exe`	`cnVuOnd3dy5leGFtcGxlLmNvbS9mYXN0LmV4ZQ==`
`connect:www.example.com:80`	`Y29ubmVjdDp3d3cuZXhhbXBsZS5jb206ODA=`

Analyze the Parsing Routines

One approach to creating signatures for this backdoor is to target the full set of commands known to be used by the malware (including the surrounding context):

```
<!-- adsrv?bG9uZ3NsZWVw -->
<!-- adsrv?c3VwZXJsb25nc2xlZXA= -->
<!-- adsrv?c2hvcnRzbGVlcA== -->
<!-- adsrv?cnVu
<!-- adsrv?Y29ubmVj
```

The last two expressions target only the static part of the commands (**`run`** and **`connect`**), and since the length of the argument is not known, they do not target the trailing comment characters (**`-->`**).

Signatures that use all of these elements have a risk of being too specific: If the attacker changes any part of the malware, it will cease to be effective.

Targeting Multiple Elements

We saw that different parts of the command interpretation were in different parts of the code. Given that knowledge, we can create different signatures to target the various elements separately.

The three elements that appear to be in distinct functions are comment bracketing, the fixed **`adsrv?`** with a Base64 expression following, and the actual command parsing. Based on these three elements, a set of signature elements could include the following:

```
pcre:"/<!-- adsrv\?([a-zA-Z0-9+\/=]{4})+ -->/"
content:"<!-- "; content:"bG9uZ3NsZWVw -->"; within:100;
content:"<!-- "; content:"c3VwZXJsb25nc2xlZXA= -->"; within:100;
content:"<!-- "; content:"c2hvcnRzbGVlcA== -->"; within:100;
content:"<!-- "; content:"cnVu";within:100;content: "-->"; within:100;
content:"<!-- "; content:"Y29ubmVj"; within:100; content:"-->"; within:100;
```

Targeting Multiple Elements

The first signature targets the command prefix **`adsrv?`** followed by a generic Base64-encoded command. The rest of the signatures target a known Base64-encoded command without any dependency on a command prefix.

The parsing occurs in a separate section of the code, so it makes sense to target it independently. If the attacker changes one part of the code, our signatures will still detect the unchanged part.

Note that we are still making assumptions: that the attacker will most likely continue to use comment bracketing, since comment bracketing is a part of regular web communications and is unlikely to be considered suspicious. Nevertheless, this strategy provides more robust coverage than our initial attempt and is more likely to detect future variants of the malware.

Targeting Multiple Elements

Let's revisit the signature we created earlier for beacon traffic. Recall that we combined every possible element into the same signature:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"TROJAN Malicious Beacon ";
content:"User-Agent: Mozilla/4.0 (compatible\; MSIE 7.0\; Windows NT 5.1)";
content:"Accept: * / *"; uricontent:"58"; content:!"|0d0a|referer:"; nocase;
pcre:"/GET
\/([12]{0,1}[0-9]{1,2}){4}58[0-9]{6,9}58(4[89]|5[0-7]|9[789]|10 [012]){8} HTTP/
classtype:trojan-activity; sid:2000002; rev:1;)
```

This signature has a limited scope and would become useless if the attacker made any changes to the malware. A way to address different elements individually and avoid rapid obsolescence is with these two targets:

- Target 1: User-Agent string, Accept string, no referrer
- Target 2: Specific URI, no referrer

Targeting Multiple Elements

This strategy would yield two signatures:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"TROJAN Malicious Beacon UA with Accept Anomaly";
content:"User-Agent: Mozilla/4.0 (compatible\; MSIE 7.0\; Windows NT 5.1)";
content:"Accept: * / *"; content:!"|0d0a|referer:"; nocase;
classtype:trojan-activity; sid:2000004; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg:"TROJAN Malicious Beacon URI";
uricontent:"58"; content:!"|0d0a|referer:"; nocase; pcre:"/GET
\/([12]{0,1}[0-9]{1,2}){4}58[0-9]{6,9}58(4[89]|5[0-7]|9[789]|10[012]){8} HTTP/"
classtype:trojan-activity; sid:2000005; rev:1;)
```

Understanding the Attacker's Perspective

Network Countermeasures

Safely Investigate an Attacker Online

Content-Based Network Countermeasures

Combining Dynamic and Static Analysis Techniques

Understanding the Attacker's Perspective

Understanding the Attacker's Perspective

Attackers too struggle to update software, to remain current and compatible with changing systems! Any necessary changes should be minimal, as large changes threaten the integrity of their systems.

So, using multiple signatures targeting different parts of the malicious code makes detection more resilient to attacker modifications.

Here are additional rules of thumb to exploit attacker weaknesses:

- *Focus on the protocol elements that are part of both endpoints.* Changing either the client or the server code alone is easier than changing both.
- *Focus on any elements of the protocol known to be part of a key.*
- *Identify elements of the protocol that are not immediately apparent in traffic.* To avoid getting obsolete by the attacker's response to another defender, try to identify aspects that other defenders might not have focused on.

Next: Anti-Disassembly