Autograph Quick Start Guide

Autograph Development Team
Email: autograph@autograph.cs.cmu.edu
URL: http://autograph.cs.cmu.edu

Table of Contents

1 Autograph Introduction

1.1 What is Autograph?

Autograph is a system that *automatically* generates signatures for novel Internet worms propagate using TCP transport. The signatures generated by Autograph are the byte patterns unique and specific to Internet worm traffic payloads. Those signatures can be used by conventional Intrusion Detection Systems (IDSes) to identify and/or filter malicious worm flows.

Fast and accurate generation of worm signatures is crucial for IDSes to effectively intervene to halt and reverse the spreading of novel Internet worms. However, the signature generation has been known to entail non-trivial human labor and thus significant delay. Autograph's main goal is to reduce the time to generate good signatures when worms outbreak.

Autograph is designed with the following desirable properties in mind:

Timeliness to enable fast reaction against aggresively spreading Internet worms.

Automation to take human out of the signature generation loop and thus achieve faster signature generation.

High quality signature generation to allow reliable and fully automated signature deployment.

Application neutrality to defend a broader set of Internet applications.

Robustness against worm payload variability to generate useful signatures even when the worms change some portion of payloads while propagating.

Autograph achieves those properties by monitoring suspicious traffic flowing into the edge networks' DMZs and analyzing the *prevalence of portions of flow payloads*. Moreover, Autograph leverages the multiple monitoring points dispersed in the Internet to achieve much faster signature detection. For more detail on Autograph's signature generation technique and the preliminary analysis see one of our Usenix Security papers titled *Autograph: Toward Automated, Distributed Worm Signature Detection* (`http://autograph.cs.cmu.edu`).

1.2 Features and Limitations

Autograph is recently proposed and still evolving. The current distribution of Autograph source code includes only a subset of the full Autograph features presented in the Usenix Security paper.

Simple suspicious flow selection technique: The current Autograph distribution uses a simple port-scanner-detection technique to determine suspicious incoming flows. Autograph counts failed connection attempts from each external source IP by monitoring the completion of TCP SYN/ACK handshaking. ICMP monitoring, countermeasure of spoofed scanning, more sophisticated scanner detection techniques are not yet incorperated into this version of Autograph.

Generating signatures of TCP scanning worms: Currently, Autograph can generate signatures for TCP worms that spread by scanning the Internet. Autograph is

application-neutral and thus it does not require any application-specific knowledge above TCP layer.

Flow reassembly: Autograph performs flow reassembly for packets belonging to suspicious flows. Note that the perfect flow reassembly is a non-trivial task and often it is impossible to achieve because of the difference between Autograph's view and the connection endpoints' view.

COntent-based Payload Partitioning(COPP): After accumulating enough number of suspicious flows, Autograph performs COPP to partition the payloads into variable-length chunks (*content-block*). Then, it constructs a prevalence histogram to determine frequently occuring patterns across captured suspicious flows. COPP is a technique to partition payloads based on their content and thus, Autograph can tolerate a certain degree of payload variability. Note that, however, strong polymorphism or encryption is still a big challenge.

Output in Bro's signature format: This version of Autograph outputs the detected signatures only in Bro's signature format. However, you can find it easy to convert the output into a different IDS's signature format.

Linux-based system: We developed and tested the current version only in Red Hat Linux system with a Fast Ethernet network interface. We hope to port Autograph for other type of Unix-based operating systems and different network interfaces.

Single monitor support: The current distribution relies on the local observation to select suspicious traffic and generate signatures. It does not include the *tattler* module that enables information sharing across multiple Autograph monitors yet.

We keep adding more features into Autograph implementation. We will appreciate so much your feedback and suggestion, that is valuable for the future release of improved Autograph.

1.3 More information

The official Autograph website, where you can find the latest Autograph distribution and the related document, is located at:

http://autograph.cs.cmu.edu.

Send questions on any Autograph subject to *autograph-users@mailman.srv.cs.cmu.edu*. You can subscribe by going to the official Autograph website,

2 Requirements

2.1 Network Tap

Autograph needs to be installed at the boundary of your network, where communication between internal and external hosts can be monitored. DMZ is the typical place for you to put Autograph. However, note that Autograph currently relies on port-scanner information to identify suspicious flows. Thus, Autograph needs to be placed before any proxy that filters out scanning activities.

You may choose to feed Autograph only inbound traffic because of restriction in your network topoloty or large traffic volume. Then, Autograph will try to guess the completion of connection setup based on time-out. Since the current version performs content-prevalence analysis on inbound suspicious flows, Autograph is still able to generate signatures. However, this one-way communication monitoring will affect the performance and accuracy of Autograph's suspicious flow selection heuristic.

2.2 Hardware and Software Requirement

Hardware: Current Autograph requires no special hardware, and can be run on general type of PCs. For example, we are currently running Autograph on a general PC with Intel Pentium4 3.06GHz CPU, 1GB RAM, and a 100GB HDD to monitor a T3 network link.

Operating System: Current Autograph is developed and tested on Linux and FreeBSD. We plan to support other type of Unix operating systems in the near future.

Hard disk: Current Autograph stores intermediate states and outputs the final results in HDD. Fast and large harddisk drive is recommended.

User privileges: *superuser* to install Autograph and tap network interfaces in promiscuous mode. However, if you just want to test Autograph offline with tcpdump packet traces, you do not have to be a superuser.

Network Interfaces: Packet capture on 100Mbps Ethernet card with libpcap support is tested.

Software:

libpcap version 0.8 or higher (`http://www.tcpdump.org`) for network tap

rabinpoly version 1.0 or higher (`http://www.cs.cmu.edu/~hakim/software`) for COPP

Perl version 5.6 or higher (`http://www.perl.org`) to use utilities

Note: Autograph needs the full source code of rabinpoly for compile. You don't have to compile or install the rabinpoly library. For more detail, refer to the Installation section.

3 Installation

3.1 Download

1. Download **Autograph** from: http://autograph.cs.cmu.edu.
2. Download **rabinpoly** from: http://www.cs.cmu.edu/~hakim/software.

3.2 Install

You have to have the following information before beginning Autograph installation.

*** localnets:** a list of local subnets for your network. Autograph needs to know which networks are *internal* and generates signatures only from *inbound flows* currently.

*** network interface:** the network interface of the monitoring box where the monitored network traffic will be fed. By default, the easy-to-start script tries to access `eth0`. If you use different interface, you have to specify the interface name accordingly. For more detail, please refer to the section "Running Autograph".

Autograph installation is very easy. First, un-tar both Autograph and rabinpoly distributions under the same directory. Let's say you start Autograph building from '`tmp-autograph`' directory.

```
> cd tmp-autograph
> tar xvzf autograph-0.1.tar.gz
> tar xvzf rabinpoly-1.0.tar.gz
> cd autograph-0.1
> ./configure
```

This will generate Makefiles that automatically locate rabinpoly source distribution and will eventually install Autograph in '`/usr/local/autograph`'. If you want to install Autograph in a different directory, say '`/path/to/autograph`', run

```
> ./configure --prefix=/path/to/autograph
```

If you have the rabinpoly source distribution in a different directory, run '`./configure`' with `--with-rabinpoly` option specified.

```
> ./configure --with-rabinpoly=/path/to/rabinpoly-1.0
```

Then, you build the source code by typing,

```
> make
> make install
```

This will install compiled Autograph components ('`sffilter`', '`copp`', '`interpreter.pl`', '`autograph`') under '`/path/to/autograph/bin`' directory and sample configuration files ('`sffilter.cfg`', '`copp.cfg`', '`blacklist.txt`') under '`/path/to/autograph/etc`' directory. The last command may need you to be a superuser, depending on the permission of the installation directory.

If you want to install related documents including this manual, type

```
> make doc
> make docinstall
```

This will copy a set of compiled Autograph documents into '`/path/to/autograph/doc`' directory.

4 Getting Started

Current Autograph consists of two components, each of which runs as a separate program and needs a separate configuration file.

`sffilter`: takes packet streams from a network interface or tcpdump-style packet dumps, selects suspicious flows, and triggers `copp`'s content-prevalence analysis if necessary.

`copp`: responds to `sffilter`'s request by applying COPP to suspicious flows accumulated in the suspicious flow pool, and generating bro-style signature after performing content-prevalence analysis.

4.1 Quick Configuration

Autograph needs sufficient harddisk space to construct the suspicious flow pool and store the intermediate information. That means you need prepare a directory Autograph can access. Let's say you use '`/autograph-dir`' for this purpose.

First, copy two example configuration files, '`/path/to/autograph/etc/sffilter.cfg`' and '`/path/to/autograph/etc/copp.cfg`', into '`/autograph-dir`'. You can find the example configuration files in `etc` directory under the original Autograph source tree, too.

```
> cd /autograph-dir
> cp /path/to/autograph/etc/sffilter.cfg .
> cp /path/to/autograph/etc/copp.cfg .
```

Second, you need to provide your site-specific information to Autograph by editing the copied `sffilter.cfg`. The value of *internal_network* should be replaced with your local-net information. Assume that your network uses the IP address space 10.0.0.0/16 and 172.16.1.0/24. Then, changed `sffilter.cfg` looks like

```
internal_network 10.0.0.0/16, 172.16.1.0/24
stat_report_interval 600 # 5 min
 . . .
```

If you want to run Autograph with the default parameter values, this is the end of the configuration. For the detail of other parameter configuration, see Chapter 6 [Configuration], page 9.

4.2 Start/Stop Autograph

For your convenience, Autograph distribution includes a wrapper shell script `autograph` that invokes `sffilter` and `copp` at once. The wrapper shell script is in '`/path/to/autograph/bin`' or in '`aux`' directory under the original Autograph source tree.

If the network interface you are monitoring is `eth0`, you can start Autogran by simply typing,

```
> cd /autograph-dir
> /path/to/autograph/bin/autograph start
```

If the network interface is other than `eth0`, specify the name with `--with-interface=<netinterface>` option. Use `--help` option to find more information on other available options of the wrapper shell script.

```
> /path/to/autograph/bin/autograph --help

Usage: ./autograph {start|stop} [VAR=VALUE] ...

For start command, you have to specify the following variables. Otherwise,
the program will try to determine the values after examine a few possible
directories in your system.

Variables:
  --with-autograph=ARG  Path to installed autograph directory.
  --with-copp-config=ARG        Configuration file name for COPP.
  --with-sffilter-config=ARG    Configuration file name for Filter.
  --with-bpffilter=ARG  Optional BPF filter. (default: none)
  --with-interface=ARG  Network interface. (default: eth0)
```

Check with `ps` if both programs `sffilter` and `copp` are successfully launched.

In order to stop Autograph, use `autograph stop` command.

4.3 Autograph Output

If you didn't make any configuration change other than *internal_network*, Autograph will creat '`suspect_pool`' directory under your current directory (here, '`/autograph-dir`'). Autograph uses this directory as its suspicious flow pool (SFP).

You will see other files in your current directory that hold the various output Autograph generates. **NOTE:** you may not see anything interesting in those files except `sig.out` unless you have configured `sffilter` and `copp` to report states. See Chapter 6 [Configuration], page 9 for the detail.

`sig.out`: generated signatures in bro-style signature format.

`sffilter.err, sffilter.std`: output from `sffilter`.

`copp.err, copp.std`: output from `copp`.

`scanners_stat.txt`: external IPs accused of scanning.

`sfp_state.txt`: suspicious flows in SFP.

See *bro* manual to understand bro's signature format. http://www.bro-ids.org. The script `script/interpreter.pl` in Autograph source tree may be helpful to check the generated signatures.

5 Autograph Overview

This chapter presents a brief overview of the current Autograph implementaion included in this distribution. This will help you understand the configurable parameters, too. As noted, this initial distribution omits some interesting features of the complete Autograph system. For example, this version of Autograph does not include *tattler* component that allow distributed collaboration among multiple Autograph monitors. See our Usenix Security paper to get the full picture of the entire Autograph system.

Currently, Autograph consists of two interacting programs (`sffilter`,`copp`). `sffilter` performs suspicious flow selection heuristics (scanner-detection-based heuristic) and triggers `copp`'s content-prevalence analysis via FIFO when enough number of suspicious flows are accumulated in SFP. The FIFO is non-blocking, and thus `sffilter` can send request messages as long as the FIFO queue can hold. `copp` processes the request one by one.

5.1 Suspicious flow selection (sffilter)

`sffilter` processes packets either from a live network interface or from tcpdump-style packet dump traces using libpcap library. It checks if the packet is originated from an external scanner. If so, it performs flow reassembly. Completed suspicious flows' payloads are stored in SFP directory as files. The files (completed suspicious flows) are removed from SFP after *t_thresh* seconds. If the current number of flows for a port equals to or is greater than *theta_thresh* , `sffilter` signals `copp` by sending a request message via FIFO.

All inbound/outbout TCP packets are used to determine scanners. If an external host has made more than or equal to *s_thresh* failed connection attemps during the last *scanner_lifetime* seconds, Autograph considers the external host to be scanning. Once the IP address is accused of scanning, the IP is considered a scanner for *scanner_lifetime* seconds.

Here, a failed connection attempt means that the internal peer host is non-existent or does not run a service on the destination port so that there is no subsequent response from the internal host within *flow_inactivity_timeout* seconds after the initial inbound SYN packet. Or, the internal connection peer has rejected the connection setup request. In order to tolerate a temporal failure of servers or the noise from failed-connection-prone p2p applications, we keep track of the pair of (internal host IP, port) used for successful connection during the last *server_lifetime* seconds, and exclude any failed connection attempts from counting for scanner detection. You can turn on/off this heuristic by configuring *live_server_heuristic* parameter.

5.2 Content-prevalence analysis (copp)

`copp` waits for incoming requests from `sffilter`. The request message contains information on protocol number, port, number of suspicious flows `sffilter` observed, and time. When a request message is received `copp` checks SFP directory and reads all the files corresponding to the flows with the protocol and port number specified in the request message. Then, it chops each flow with COPP and constructs content-prevalence histogram. To be selected as a signature, a content-block should 1) be highly-ranked in the histogram, 2) be generated from at least *min_prevalence* flows, 3) be sent by at least *source_count* different external, and 4) be not listed in signature blacklist. See Section 6.3 [signature blacklist],

page 10 to know how to use signature blakclist option. The best source for more information on COPP and Autograph's content-prevalence analysis is the recent Usenix Security Paper.

6 Configuration

6.1 sffilter configuration

You can specify the packet source and the configuration file in command line.

```
sffilter [-hv] [-i interface] [-r inputfile1 [-r inputfile2 ...] ]
         [-w output] [-f bpffilter] [-c config]

options:
You have to select either network interface (-i) for online monitoring
or tcpdump packet traces (-r) for offline analysis.
-i interface  network interface to be monitored
-r inputfile tcpdump-style packet dump trace. sffilter can process
multiple packet dump traces.
-w output dump packets into output in tcpdump-style format.
-f bpffilter tcpdump-style bpf filter.
-c config  sffilter configuration file.
-h print help
-v verbose
```

`sffilter` configuration file is used to specify the following parameters.

`internal_network`: list of your local net address block. CIDR format, comma separated. *__NOTE__: you have to configure this before starting Autograph

`stat_report_interval`: interval (second) to report SFP, scanner statistics if non-zero value specified. `sffilter` will not report statistics if this value is zero. (default: 0)

`live_server_heuristic`: use a heuristic that does not count failed connection attempts toward internal servers that were active for last *server_lifetime*. This heuristic helps tolerate temporary service failure or p2p application effect. (default: true)

`server_lifetime`: value used for `live_server_heuristic`. (default: 86400 = 24 hours)

`flow_inactivity_timeout`: flow inactivity timeout in second. If a connection is not established before timeout, the connection is considered a failed connection attempts. If a suspicious flow is idle for this timeout value, the flow is considered terminated and sent to SFP. (default: 120)

`scanner_lifetime`: the lifetime of scanners (default: 86400 = 24hours)

`s_thresh`: *s* threshold. Minimum number of failed connections to detect a scanner. (default: 2)

`content_analysis`: signature generation module. currently, only `copp` is supported. (default: copp)

`sfp_path`: directory name used for SFP. It should be same as `sfp_path` specified in `copp` configuration. (default: `"suspect_pool"`)

`t_thresh`: lifetime of suspicious flows. Suspicious flows will be removed from SFP *t_threshold* seconds after captured. (default: 1800 = 30min)

`theta_thresh`: *theta* parameter. Minimum number of suspicious flows in SFP to trigger signature generation for the port. (default: 5)

`analysis_interval`: minimum interval to request signature generation. If `sffilter` requested the signature generation for the port less than *analysis_interval* seconds ago, suppress the request this time and wait for another chance. (default: 60 = 1min)

6.2 copp configuration

```
copp [-hv] [-c config]

options:
-c config  copp configuration file.
-h print help
-v verbose
```

`copp` configuration file is used to specify the following parameters.

`stat_report`: will report content-prevalence histogram snapshot(default: false)

`max_block_size`: COPP parameter to specify maximum content-block size in bytes (default: 1024)

`avg_block_size`: COPP parameter to specify average content-block size in bytes (default: 64)

`min_block_size`: COPP parameter to specify minimum content-block size in bytes (default: 32)

`window_size`: COPP parameter to specify the sliding window size in bytes (default: 16)

`breakmark`: COPP parameter to specify the breakmark value (default: 0x7)

`target_fraction`: w parameter to indicate target coverage of generated signatures. shoule be between 0-1. (default: 0.95)

`source_count`: minimum number of external sources for a content-block to be selected as a signature. (default: 2)

`min_prevalence`: p parameter to specify the minimum occurence of a content-block to be selected as a signature. (default: 5)

`max_sig_num`: maximum number of signatures `copp` can generate for each iteration. (default: 30)

`signature_out`: `copp` will output generated signatures in this file. (default: `"sig.out"` in current directory)

`blacklist`: file of signature blacklist. If not specified, `copp` does not use signature blacklisting. See Section 6.3 [signature blacklist], page 10 for more detail. (default: signature blacklist is not used)

`sfp_path`: directory name used for SFP. It should be same as `sfp_path` specified in `sffilter` configuration. (default: `"suspect_pool"`)

6.3 signature blacklist

Autograph generates signatures based on content-prevalence analysis, and thus it possibly reports signatures that are not specific or sensitive. Signature blacklisting is one way to prevent Autograph from generating bad signatures. Make your own signature blacklist file

that lists signatures Autograph can generate but not good signatures. The format of the blacklist is exactly the same as Autograph's signature output (bro-style signature format). That means, you can generate a signature blacklist by editing the output of Autograph from training period. Then, specify the blacklist file name in `copp` configuration file (*blacklist* parameter). Autograph will ignore all the content blocks that are *substrings* of one of the blacklist patterns.

Here is an example signature blacklist:

```
signature blacklist0 {
        header ip[9:1] == 6
        payload /.*\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70/
        event "Signature : An Example Signature Black List Entry"
        }
```

Index