Unix-style approach to web application testing

Andras Veres-Szentkiralyi

Unix-style approach to web application testing

GIAC (GWAPT) Gold Certification

Author: András Veres-Szentkirályi, vsza@silentsignal.hu
Advisor: Rajat Ravinder Varuni
Accepted: February 25th 2020

Abstract

Web application testers of our time have lots of tools at their disposal. Some of these offer the option to be extended in ways the original developers did not think of, thus making their tool more useful. However, developing extensions or plugins have entry barriers in the form of fixed costs, boilerplate, et cetera. At the same time, many problems already have a solution designed as a smaller standalone program, which could be combined in the Unix fashion to produce a useful complex tool quickly and easily. In this paper, a (meta)solution is introduced for this integration problem by lowering the entry barriers and offer several examples that demonstrate how it saved time in web application assessments.

1. Introduction to Unix principles in the context of Burp Extensions

1.1. The Burp Extender

Burp Suite has been the choice of web penetration testers for quite a time for its excellent mix of tools that work together well as part of a complex tool.

It had at least a limited support for extending its functionality since 2009, and at the end of 2012, it received great improvements with version 1.5.01 (PortSwigger Ltd, 202). Although developments have followed in the years since then, most things Burp users take for granted nowadays in this regard became a part of Burp Extender then, like the

- ability to use multiple extensions simultaneously and
- dynamic loading and unloading of extensions.

Although one of the points was "easier extension development for non-programmers", this model still required quite a bit of boilerplate, leading to many sophisticated extensions: these filled the virtual shelves of the BApp Store, introduced in early 2014.

1.2. Unix principle

As summarized by Peter H. Salus in A Quarter-Century of Unix (1994), Unix philosophy means following these guidelines: (Eric S. Raymond, 2003)

"Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."

In the world of penetration testing (not limited to web applications), tools following Unix principles often help by making it easy to improvise. As penetration testers encounter different target systems, having the option to put together ad-hoc solutions on-demand within the limited time available for the assessment is a powerful advantage.

However, the built-in extensibility model of Burp does not make including an already existing tool easy if it was not explicitly designed for use with Burp.

1.3. Piper basics

The Burp extension called Piper was born out of the idea that more and more ways to augment Burp functionality just meant invoking an external command and doing the transformations necessary to carry data between Burp and the external program. Duplicating this non-task-specific code would have just led to boilerplate while making it easier to generalize such functionality could enable experimentation with tighter feedback loops.

The plugin is available under GNU GPLv3 license on GitHub: https://github.com/silentsignal/burp-piper

The following sections will demonstrate how Piper can be used as a platform to integrate simple but powerful Unix pipelines into Burp to aid penetration testing.

2. Unix-style approach to web application testing in practice

2.1. Reusing the comment field in Burp for unexpected improvements

2.1.1. Commentator basics

The Proxy module of Burp contains all the requests that passed through the suite in a table-like structure with most columns containing read-only cells. However, one of the exceptions is the “Comment” column, where the user can enter anything, and it gets persisted along with the request. The proxy filters can also narrow the subset of shown requests to those that have a comment only. Since extensions can read and modify the comment programmatically via the API, comments can be used for much more than just taking notes manually.

In January 2017, a Burp extension called Commentator was released with its single piece of functionality being auto-filling this “Comment” field. Such an extension could be useful for more straightforward navigation and analysis where the penetration tester tries to make sense of what requests the browser sent, how the server responded, and deduce useful bits of information about the internals of the target app. Commentator

used regular expressions that could be applied either to the request or response, and the extension copies the first group to the comment field.

For example, in Figure 1 below, the URL-encoded value for the key “IDButton” is extracted from the request, and the comment field automatically gets filled with it.

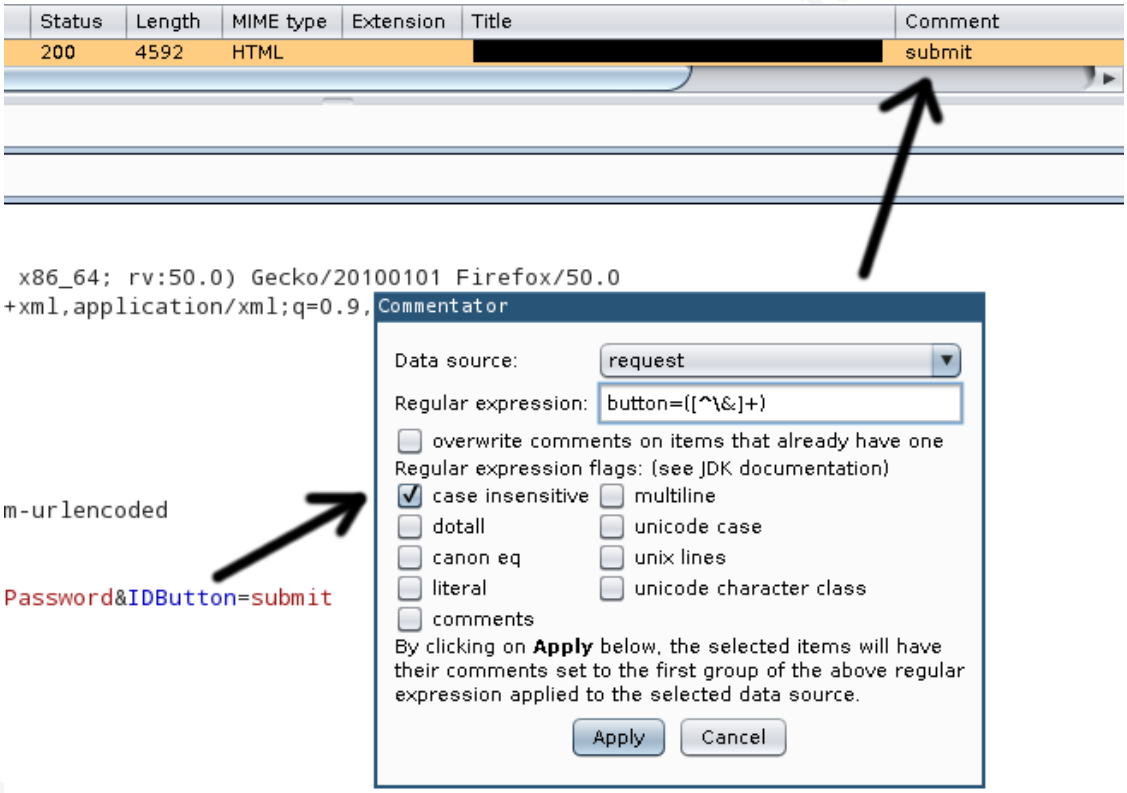

Figure 1 Demonstration of the basic functionality of the Burp Extension "Commentator"

This worked for many scenarios. One example is an API where the URL was always the same, while a POST parameter in the HTTP request body selected the function to be invoked on the server. Without the comment field, every line looked the same, and navigating through would have required stepping through each of them and looking at the request body.

While this approach was simple and covered many use-cases, regular expressions are far from being a universal solution. Thinking about the abstract way Commentator worked reveals another Unix pipe scenario: anything could be used that takes the request/response as the input and produces the comment as the output.

András Veres-Szentkirályi vsza@silentsignal.hu

2.1.2. Using hashes to verify content equality

In web application assessments, one common approach is sending requests with different parameter values and observing the responses. For a small number of requests, comparing the responses is no big deal. In case of more significant amounts of requests, this pairwise comparison is harder, but if the lengths differ, unique messages can still be easily distinguished by sorting by the "Length" column, grouping equally long responses together.

However, if the differences do not change the length of the response, Commentators in Piper can help.

Apart from several application-specific use-cases, one trivial thing appeared right after the inclusion of Commentator functionality into Piper. Most Unix-like operating systems contain binaries named in the form of `Xsum` that calculates the hash digest using function *X* for files or the standard input and print it in a hexadecimal format to the standard output.

Thus, for example, adding a Commentator that simply invokes `sha256sum` without any parameters will fill the "Comment" column with a hexadecimal SHA-256 digest, making it easy to spot identical responses by the identical hash digest. For example, in Figure 2 below, while the first three responses all have the same length, we can only be sure that the contents of those bytes are also the same by populating the comment column with their SHA-256 digest.

Status	Length	MIME type	Extension	Title	Comment
200	250028	flash	swf		9411646e4ee7291bd2fd88f7c9fce...
200	250028	flash	swf		9411646e4ee7291bd2fd88f7c9fce...
200	250028	flash	swf		9411646e4ee7291bd2fd88f7c9fce...
200	575	HTML			3381b0bcd0a6b5cd43f40c4fd2707...
200	575	HTML			0914dacf911440ad96c42093e921...

Figure 2 Demonstration of using sha256sum as a Piper Commentator

Other useful commands for such statistics might be `wc(1)`, which counts lines, words and characters and `true(1)` for clearing comments since its purpose is to "do nothing, successfully" by design.

2.2. Implementing Message Viewers on a budget

Message Editors are one of the most reused components within Burp. Depending on the context, they can be used to view and optionally edit HTTP requests and responses on the tabs Dashboard, Target, Proxy, Repeater, and within extension-provided custom user interfaces. They contain one or more tabs themselves and can edit and display the raw message as text, headers as a table, hexadecimal octets, and some format-specific editors such as XML and HTML.

Extensions can also implement such tabs themselves by providing an algorithm to determine whether to show the tab for a particular message and a user interface (also known as *component* or *widget*) to implement the user-facing part of the editor. While the name is *editor*, it can be downgraded to a viewer both by the implementer (e.g., if editing does not make sense or is simply just not implemented) or Burp itself for contexts such as Proxy where content should be read-only.

The BApp Store itself contains many Message Editors for various formats as web apps and APIs have long gone past the era of using URL encoded parameters as request and HTML as response body formats exclusively. One of the original main ideas behind Piper was that some of these could be implemented quickly by feeding the request or response into an external program and displaying the output as a message viewer. This approach does not include editing, but that already covers lots of use-cases.

2.2.1. Protocol Buffers

Protocol Buffers or Protobuf is a serialization format developed by Google and thus used on some Google APIs, including ones built into Android. Without the schema used by the developers, only partial decoding is possible since serialization includes only a bare minimum of type information so that older clients can skip unknown fields. Nevertheless, it is still useful, and skilled penetration testers can either reverse engineer the fields by experience or find a way to extract the schema itself from the app.

In the case of schemaless decoding, the only setting required is the command itself:

```
protoc --decode_raw
```

The hard thing about Protobuf is that there is no header, so deciding whether to show the tab can only be decided by out-of-band information (such as an honest Content-Type HTTP header) or doing the actual decoding. For this purpose, Piper allows specifying filters on the output and exit status of the command it executes. Thus, if the command can decode the Protobuf serialized data, the tab is shown, otherwise it gets hidden without any additional (explicit) filtering.

2.2.2. ANSI colors

Some commands use colors and Piper can display them too; this demonstrates another essential feature: handling ANSI escape sequences. Burp offers its built-in plain-text editor component to extensions, which works well for most message editors, where the output is plain text only. However, specific escape sequences can be used by console applications to alter the appearance of textual output, including color, underline, bold text, et cetera. By default, Piper uses the aforementioned plain text editor built into Burp, but message viewers can set a Boolean flag that indicates their usage of color. In this case, a better replacement called JTerminal gets used that can understand such escape sequences and display formatted text correctly. Piper reuses this component for standalone commands that use the console for displaying output instead of a custom GUI.

2.3. Comparing requests and responses with external diff tools

As mentioned before, in web application assessments, one common approach is sending requests with different parameter values and observing the responses. For the comparison of responses, Burp offers two built-in solutions in its “Comparator” tab, one for textual, one for binary diffing. Both have their shortcomings, which was the primary motivation behind the development of Piper.

The textual diff, for example, shows both inputs whole, and while it highlights the differences with background colors, lack of word wrapping makes it challenging to find those highlights by merely moving around with the scroll bar, see Figure 3 below. In contrast, most `diff` implementations show only the differing part, with varying amounts of context around those differences.

Git diff can even use colors, ignore differences in whitespace, and compare words instead of lines (like traditional `diff` does). Although designed for the purpose of managing Git repositories, it can compare any two files.

Of course, any external diff tool can be hooked into Burp with Piper this way. An important thing to consider is to see how the tool produces its output. Git diff, for example, is a console application, which just writes its output as a stream of bytes to its standard output, expecting a terminal as its environment. Piper needs configuration for such programs, so that it should open its terminal emulator window that knows how to decode ANSI escape sequences used for formatting terminal output such as colors. The rest of the programs that have their own GUI do not need this, thus ticking the checkbox "Has its own GUI" in Piper disables this unneeded window.

2.3.1. Message Viewer as a filter for piping

Unix philosophy can be involved for yet another purpose here: applying the same transformation for both inputs before running the comparison. In some cases, this helps making the diff more readable, such as with formats like XML and JSON, where omitting whitespace is typical behavior of servers yet makes diffs unreadable. In other cases, it is even more critical, as while binary diffing exists, more meaningful comparison can be made by first converting it to a textual format.

Figure 3 Burp's built-in Comparator compares two large JSON documents

```
diff --git a/tmp/piper-8350675374160785262.bin b/tmp/piper-613289896204064887.bin
index a2def11..667609c 100644
--- a/tmp/piper-8350675374160785262.bin
+++ b/tmp/piper-613289896204064887.bin
@@ -2047,7 +2047,7 @@
         "disabled": false,
         "downloads_url": "https://api.github.com/repos/dnet/chrl/downloads",
         "events_url": "https://api.github.com/repos/dnet/chrl/events",
-        "fork": false,
+        "fork": true,
         "forks": 0,
         "forks_count": 0,
         "forks_url": "https://api.github.com/repos/dnet/chrl/forks",
@@ -2268,7 +2268,7 @@
         "node_id": "MDEwOlJlcG9zaXRvcnk0OTE2MDA0",
         "notifications_url": "https://api.github.com/repos/dnet/cmp/notifications{?since,all,participating}",
         "open_issues": 0,
-        "open_issues_count": 0,
+        "open_issues_count": 1,
         "owner": {
             "avatar_url": "https://avatars1.githubusercontent.com/u/163115?v=4",
             "events_url": "https://api.github.com/users/dnet/events{/privacy}",
```

Figure 4 Piper compares the same two large JSON documents using Python's json.tool piped into git diff

Even in case of a textual input, another obstacle in the way of meaningful and concise comparison is the lack of canonicalization, often abbreviated as c14n. For instance, in both JSON and URL encoded form data (GET or POST parameters), the order of key-value pairs is arbitrary, and reordering them should not change their meaning. Since many programming languages (such as in CPython before 3.6) reorder data structures typically used for representing such key-value lists (dictionary, map, hash), it is not uncommon to encounter HTTP requests with the same set of key-value pairs in a different order.

András Veres-Szentkirályi vsza@silentsignal.hu

Canonicalizing such content leads to minimal, readable diffs: the easiest way is just sorting the keys alphabetically. For JSON, the latter is the default behavior of the "json.tool" program built into the Python standard library, and an example can be seen above in Figure 4. For URL encoded form data, below is an example:

```
head -n1 | cut -d ' ' -f 2 | cut -d '?' -f 2 | tr '&' '\n' | sort
```

The above example handles (so-called "GET") parameters passed in the URL, thus first, we narrow the HTTP request to the first line with `head -n1`. Then the request path is selected by cutting the second field (`-f`) delimited (`-d`) by spaces. Selecting the second field delimited by the question mark cuts the parameters again, leaving the URL parameters. The only things left now are splitting each parameter to a new line by replacing ampersands with newlines, then `sort` deals with the alphabetic order by operating on lines.

HTTP headers behave similarly, their order should not be meaningful, so in case the server mixes up the order, a simple `sort` can canonicalize the order before comparing the headers of two requests. An example of this can be seen below in Figure 5, Piper's window is on the top, the built-in Comparator is on the bottom, comparing the same duo of requests.

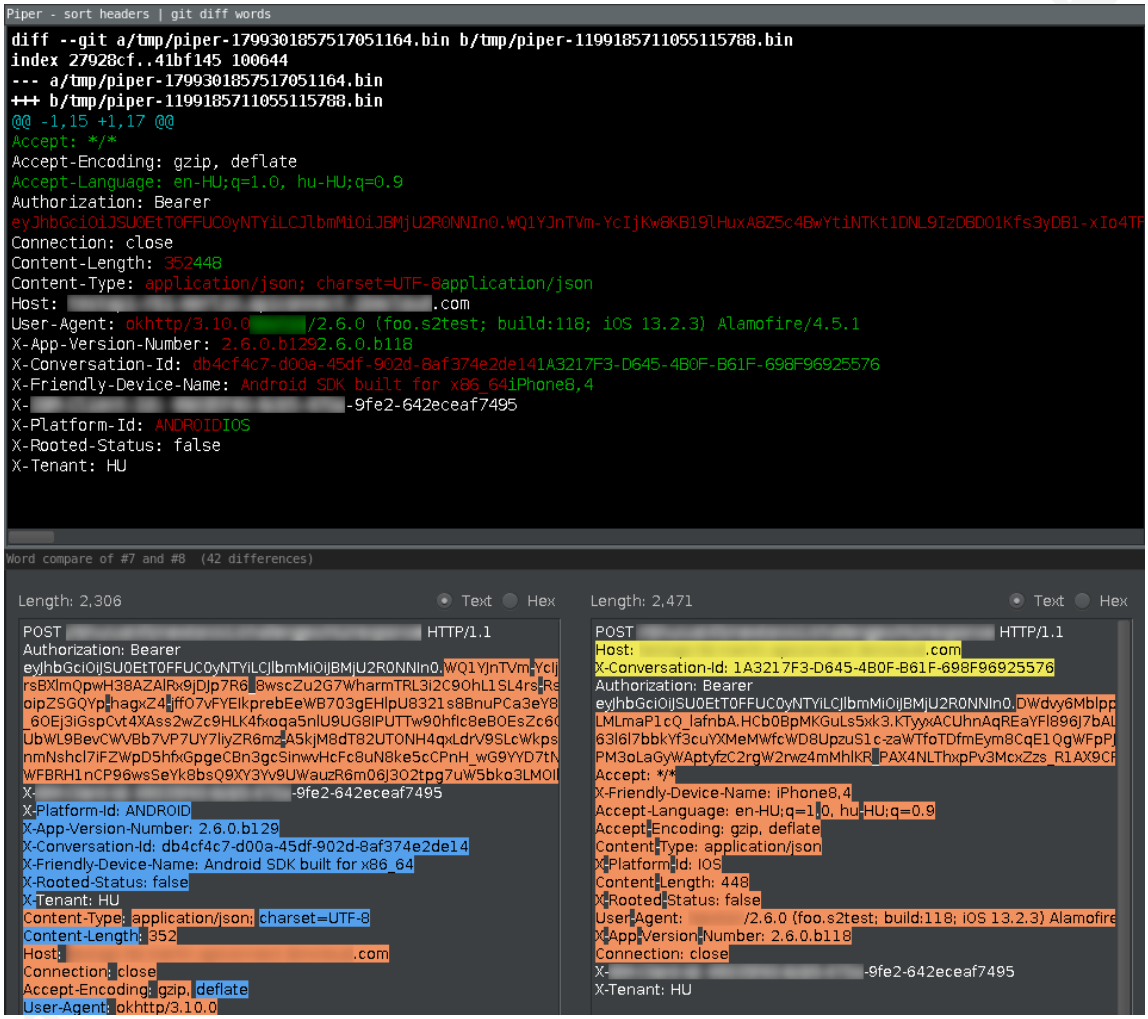

Figure 5 Difference between Piper comparing headers after sorting vs. Burp's built-in Comparator doing a naive diff

Piper offers all combinations for chaining a message viewer and a command – given that the message viewer can handle all inputs (based on its filters, if there are any), and the command does not pose any restrictions on the input format. Such an algorithm inevitably produces false positives (chains that make no sense) and false negatives (chains that would be useful yet are not covered). At least, the former can be eliminated with a checkbox telling Piper to ignore the possibility of piping message viewers into a specific action. These could be improved further, and this is a great topic for future development.

Burp also offers a binary diffing tool presenting each byte as a two-digit hexadecimal number and coloring based on the result of the comparison. For this

purpose, there are also external tools that can perform better. Radare2 follows Unix principles by having separate tools that can work on their own, thus its binary diff program called `radiff2` can be invoked this way. By default, it only shows the offset and values of differing octets, while using the command line parameter `-x` presents a two-column layout with colored hex digits showing identical (green/white) and different (red) bytes.

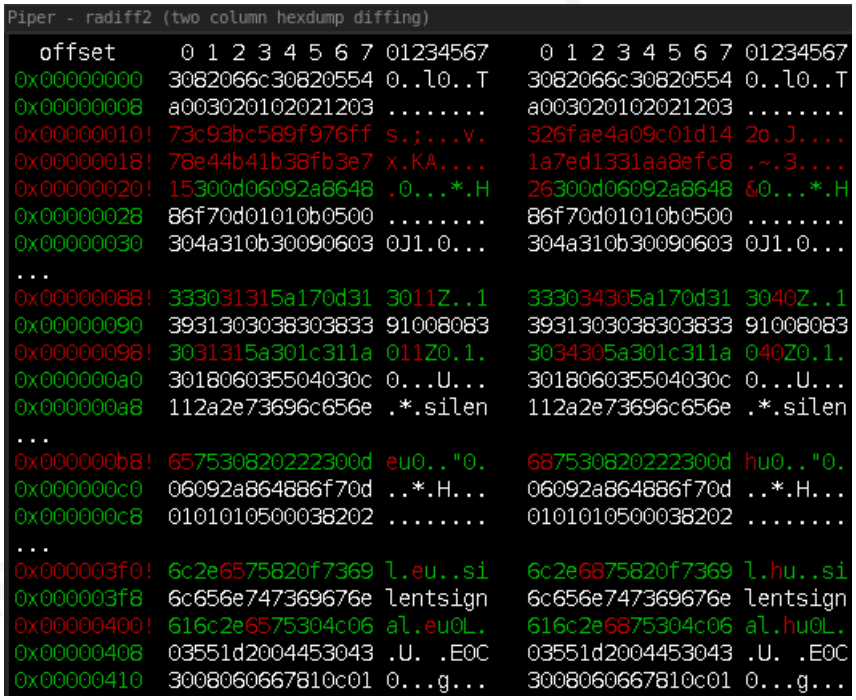

Figure 6 radiff2 showing only the differing parts, and coloring hex octets within lines with differences

2.4. E2E encryption

Although most modern user-facing HTTP stacks (browsers, Android and iOS) enforce the use of TLS, the point-to-point confidentiality it provides not necessarily covers the full channel towards the other endpoint. Many organizations terminate TLS way before, usually for reasons of performance (CDN), compliance (DLP), security (WAF) and any other circumstance that results in such a separation of concern, including legacy or incompatible components.

In such scenarios even though the transport channels might be protected with TLS between the client and the server, it is not a single continuous secure channel, but rather

the concatenation of several ones. The vital difference is that any node that terminates a TLS layer and forwards plaintext data to another one can see and potentially modify everything right below said layer of protection. Thus, in cases where the purpose of the application demands it, additional layers of encryption and signing might be applied. The result is the so-called end-to-end (E2E) protection where only the client and the server can get the (inner) plaintext.

For penetration testers, this offers two unique challenges. First, to read the plaintext and thus understand the inner workings of the protocol, a way must be found to make the E2E encrypted requests and responses readable in Burp. Even though Burp "removes" the TLS layer by being a man-in-the-middle proxy, payloads encrypted using the (inner) E2E layer just appear as unreadable garbage. Second, to mount attacks using Repeater or Scanner, another solution must be used to encrypt the modified payload before sending it to the target.

As E2E encryption is becoming a typical scenario, especially in the context of smartphone apps communicating over HTTPS with their backend APIs, many have come up with solutions for this problem. One purpose-built tool is Brida that bridges Burp with Frida, a dynamic instrumentation framework, hence its name. Piper takes a more generic approach of letting the user provide an implementation in any programming language to perform the transformations described above.

The first part is usually writing a message viewer that can decrypt the payload, as it can be tested in a tight feedback loop easily. The penetration tester develops a script or program that reads an HTTP request or response, decrypts the E2E encrypted part, and displays the plaintext. The result can be even tested outside Burp using saved messages. Once complete, it is now easy to interpret HTTP requests and responses in the Proxy view. With a bit of change regarding the output formatting, the message viewer can be further improved by producing a short one-liner output to be used as a commentator. This way, several messages can be seen at the same time, making complex workflows consisting of several correlated messages easier to understand. Piper helps this by having a button called "Convert to commentator" on the toolbar above the list of message viewers.

The next part is writing a piece of code that encrypts payloads within HTTP requests so that Repeater and the built-in checks of Scanner can test interfaces that expect E2E encrypted and signed data. Burp offers so-called HTTP listeners for this purpose that can intercept requests before leaving Burp and responses right after arriving in Burp and potentially modify them. A small but important detail to be mindful of is the order of extensions registering HTTP listeners. For example, the popular extension Logger++ collects its data using HTTP listeners, thus putting Piper before Logger++ results in it logging the already encrypted request. Reversing the order makes the request appear in Logger++ in its unencrypted form. Both are useful in certain situations, so the penetration tester must be aware of the need to make a choice.

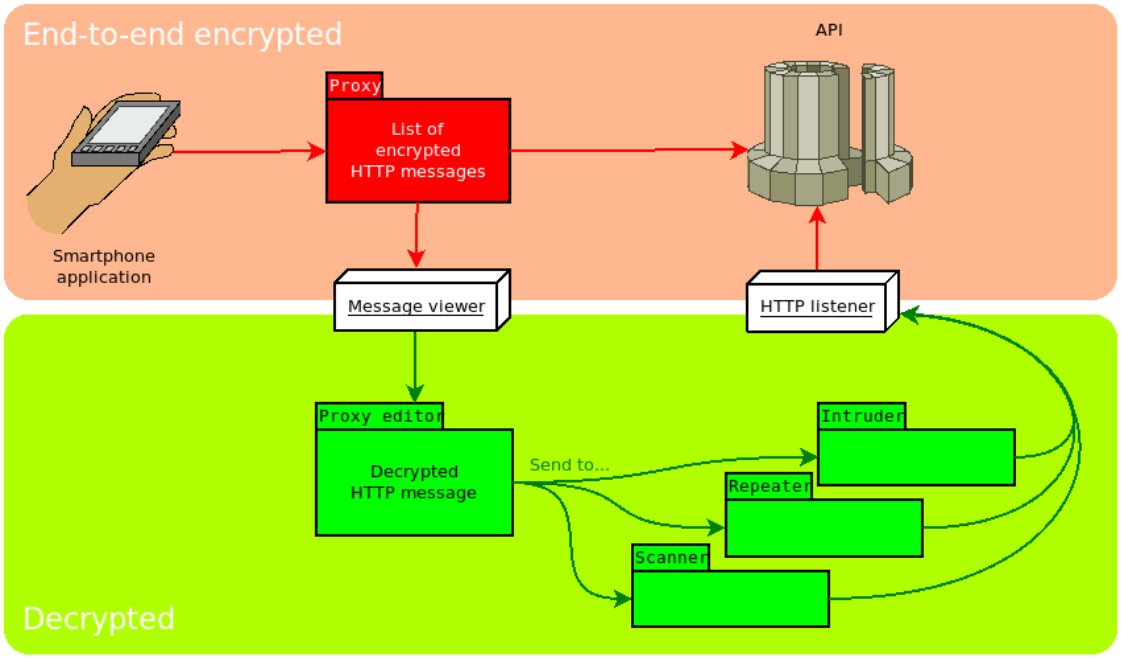

Figure 7 Flow of encrypted and decrypted content during an assessment of an end-to-end encrypted app – red means encrypted, green means plain text

HTTP listeners also receive metadata regarding what tool within Burp initiated the request. Knowing the source can narrow the origins of requests that need to be encrypted. For example, excluding Proxy is a good start most of the time, as it conveys messages sent by the original application – thus, they are probably encrypted already. However, if there is a need for an external attacker tool and for the sake of simplicity, it gets proxied through Burp to make use of the E2E encryption functionality, it can be a

good idea to enable it. Piper makes it possible to fine-tune this behavior using checkboxes for each tool in the HTTP listener configuration dialog.

2.5. Comment collection

While comparing requests was the initial idea that sparked Piper, the implementation allows a broader spectrum of possibilities. Piper just executes an external program, passes the requests or responses to the standard input or command line parameters, and displays the standard output. Even though Burp has a command to export selected requests and responses to XML files, both developing scripts that consume its format and running it repeatedly is hardly convenient for many purposes. However, generalizing the feature of Piper originally designed for comparisons can be used for many unique purposes.

One such use-case is combing through requests, hunting for interesting patterns in the hope of uncovering information leaks. Many web applications emit error messages within the HTTP response, which stay hidden in browsers by utilizing HTML comments. While extensions like Error Message Checks (PortSwigger Ltd., 2018) do a decent job at finding common error messages using an impressive pattern list, in custom applications, manual work is hard to avoid completely. Piper can help with this by piping all selected HTTP responses to the following command:

```
grep -Pzo '(?s)<!--.*?-->'
```

The `-o` switch switches `grep` into a mode where it prints only the matched part of each line to the output. Thus, we need a regular expression that matches HTML comments. `<!--` is the start of the HTML comment, and since none of its characters have a special meaning in regular expressions, these 4 symbols are matched literally. Right after the start of the HTML comment is a dot that matches any character. The star modifier means that not only a single, but any number of any character will match. The question mark limits the greediness, thus matching a minimal amount of characters before encountering the end of the comment (`-->`). Without this, two consecutive comments would have been mashed together, including the non-comment content between them.

In the above wording, “any character” is literally any character. However, the default is that dots in regular expressions match anything but newlines. Since HTML comments can and often do so, an option called “dotall” needs to be set that enables the expected behavior where “any” means “any”. This change is achieved by the `(?s)` prefix, completing the expression.

The `-P` switch enables Perl-compatible regular expression mode, which is needed for the “ungreedy” matching and dotall switching to work, as they are not part of basic regular expressions. Finally, the `-z` switch switches line terminator from newline to a zero byte (ASCII NUL character), thus multiline comments are matched by the regular expression.

2.6. Small everyday functionality

In a way, most developments in IT automate a task that could have been performed by humans, but a machine can be faster, more precise, and have a higher tolerance for monotony. There is a class of functionality where the potential gain is much more significant than the programming effort, such as numerical computations. Then there are the problems where the time to be shaved off is in the same ballpark as automating the task. (Randall Munroe, 2013) Lowering barriers of entry could improve situations like these, and Piper is a good candidate for this.

2.6.1. Copying HTTP requests and responses

One example in the context of web penetration testing is copying HTTP requests and responses, with particular regard for headers that carry metadata important for analysis and debugging. The relevant RFCs specify line endings as carriage returns and line feeds (CR + LF, `"\r\n"` in C-like languages). However, Unix-like systems only use line feeds (LF, `'\n'`) to signal a new line, while at the same time, terminal emulators treat both characters as newlines when coming from the user (such as pasting from a clipboard).

This duality results in situations where pasting HTTP messages into terminal-based text editors such as Vim causes empty lines being pasted between each HTTP header. Piper offers a simple solution by piping it through the following command.

```
dos2unix | xclip
```

The `dos2unix` tool converts text files from "DOS" line endings (CR + LF like in HTTP) to "Unix" ones (LF only). This cleaned output is then piped further into `xclip`, which, by default, reads its standard input and stores it in the clipboard shared by GUI applications. The final result is that by issuing this command after selecting an HTTP message, the request or response gets copied to the clipboard like with Ctrl-C. However, in this case, pasting it into a Unix-like terminal emulator will not have those pesky empty lines between HTTP headers.

2.6.2. Copying binary payloads

Another small but useful thing is getting a binary payload out for tinkering. The following command encodes the body of a request or response using Base64 and copies it to the clipboard. The `-w0` switch inhibits `base64` from wrapping the output with newlines; thus, we get a single long line.

```
base64 -w0 | xclip
```

Now this line could either be used directly on the command line:

```
$ xclip -o | base64 -d | ...
```

Alternatively, it can be inserted into a Python REPL:

```
>>> from base64 import b64decode
>>> request = b64decode("AAAA...")
```

Of course, this is just the start of a journey into analyzing an unknown payload, but lowering the entry barrier makes the penetration tester happier as it is easier to focus on the task at hand when the overhead is lower.

2.7. Intruder payloads

One of Burp's signature tools is the Intruder module, which highlights the importance of tool-assisted manual testing. It lets web application penetration testers to set up every aspect of a request template manually and then let the machine do the tedious, automated part. It has lots of built-in options for generating and processing payloads, but Extender plugins can also extend it in both regards.

2.7.1. Payload processing

Some sites, for example, use encrypted URL parameters to hide internals or protect from attacks. (Silent Signal LLC, 2018) However, if the method of encryption is flawed, a good payload processor can aid in both reversing the encryption and fuzzing the remote service with encrypted payloads.

In this case, the command executed by Piper transforms the payloads generated by Intruder before being sent to the server. Thanks to the design of Intruder, in true Unix fashion, it can still integrate with the built-in Intruder processors, since Piper payload processors share the same list with options for reordering these stages transformation as the built-in ones within the Intruder UI. So, while HTTP listeners are applicable for similar purposes, Piper payload processors can still be useful on their own, especially since, in this case, one can focus on the payload only, while HTTP listeners handle whole HTTP requests or responses.

2.7.2. Turbo intruder

This section would not be complete without mentioning Turbo Intruder, a brilliant extension for a similar purpose, made by James Kettle, a member of the PortSwigger team itself. (PortSwigger Ltd., 2019) Since it uses the Python language to allow customizing requests, there is an overlap of functionality. However, since it uses Python *only* and even that gets executed by Jython (an implementation of Python running on top of JVM instead of CPython), there can be cases when even Python code is incompatible with Turbo Intruder. Such incompatibility can happen when the code calls Python modules implemented in native code (typically written in C), thus are dependent on the execution environment being CPython.

Since Turbo Intruder uses its purpose-built HTTP stack developed with performance in mind, in many cases, it might be better to prefer it to (classic) Intruder and Piper when the constraints allow this. Thus, it is again the penetration tester's job to decide which option makes the most sense for the task at hand.

3. Conclusion

Burp Suite offers extensibility to penetration testers so that it is well-integrated GUI can accommodate an even more significant part of the web application assessment process. Piper blends into this tradition by offering GUI-based configuration interfaces that lower the entry barrier for creating ad-hoc (meta)extensions. Following the UNIX principle of *doing one thing and doing it well*, many tools already exist that can be integrated into Burp this way, well within the timeframe of a typical assessment even for ad-hoc tasks.

By creatively reusing the comment field, the proxy module can make it easier for a penetration tester to have a bird's eye view of how the web application or API works. Piping bodies of HTTP requests and responses through existing programs, simple shell one-liners, or scripts to a message viewer can make otherwise less or unreadable interactions accessible. The built-in comparator can also be extended by well-tested diff programs, optionally with the message viewer filters above, resulting in better signal-to-noise ratio. There are lots of situations where either an ad-hoc solution is necessary to complete the assessment professionally, or a tool outside Burp is already available yet creating an extension from scratch would be unfeasible.

Of course, this does not mean that Burp's Extender interface is of low quality, or penetration testers should not use it. It is still a viable option for extensions that are to be used in several projects, potentially by a large number of people. It enables the creation of rich user interfaces and uses fewer resources since all processing happens in the same process. In contrast, Piper launches new processes every time it invokes an external tool, which involves an overhead.

Thankfully, the two alternatives can coexist, and now penetration testers have the option to choose between the two: putting together a quick-and-dirty proof-of-concept with Piper, loosely integrating a tool from the outside world vs. developing a proper extension, optionally tightly integrating some library. Either way, it helps automate the tedious and menial part of the project, leaving more time and resources for those parts that still need human brainpower.

András Veres-Szentkirályi vsza@silentsignal.hu

References

PortSwigger Ltd. (2012) *Release notes for Burp Suite Professional 1.5.01* Retrieved February 7, 2020, from http://releases.portswigger.net/2012/12/v1501.html

Eric S. Raymond (2003) *The art of UNIX programming* Addison-Wesley

PortSwigger Ltd. (2018) *Error Message Checks – BApp Store* Retrieved February 7, 2020 from https://portswigger.net/bappstore/4f01db4b668c4126a68e4673df796f0f

Randall Munroe (2013) *Is It Worth the Time?* Retrieved February 7, 2020 from https://www.xkcd.com/1205/

Silent Signal LLC (2018) *The curious case of encrypted URL parameters – Silent Signal techblog* Retrieved February 7, 2020 from https://blog.silentsignal.eu/2018/05/22/the-curious-case-of-encrypted-url-parameters/

PortSwigger Ltd. (2019) *Turbo Intruder: embracing the billion request attack – PortSwigger Research* Retrieved February 7, 2020 from https://portswigger.net/research/turbo-intruder-embracing-the-billion-request-attack

András Veres-Szentkirályi vsza@silentsignal.hu

Last Updated: March 28th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Chicago Spring 2020	**Chicago, ILUS**	**Jun 01, 2020 - Jun 06, 2020**	**Live Event**
SANS ICS Europe Summit & Training 2020	**Munich, DE**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANS Budapest June 2020	**Budapest, HU**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANS Paris June 2020	**Paris, FR**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANS FOR508 Milan June 2020 (in Italian)	**Milan, IT**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANS New Orleans 2020	**New Orleans, LAUS**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANS Las Vegas Summer 2020	**Las Vegas, NVUS**	**Jun 08, 2020 - Jun 13, 2020**	**Live Event**
SANSFIRE 2020	**Washington, DCUS**	**Jun 13, 2020 - Jun 20, 2020**	**Live Event**
SANS Zurich June 2020	**Zurich, CH**	**Jun 15, 2020 - Jun 20, 2020**	**Live Event**
SANS Chennai 2020	**Chennai, IN**	**Jun 22, 2020 - Jun 27, 2020**	**Live Event**
SANS Pittsburgh 2020	**Pittsburgh, PAUS**	**Jun 22, 2020 - Jun 27, 2020**	**Live Event**
SANS Silicon Valley - Cupertino 2020	**Cupertino, CAUS**	**Jun 22, 2020 - Jun 27, 2020**	**Live Event**
SANS London June 2020	**OnlineGB**	**Jun 01, 2020 - Jun 06, 2020**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**