Rockstar Games disclosed on HackerOne: Stored XSS in Snapmatic +...

Rockstar Games disclosed on HackerOne: Stored XSS in Snapmatic +... - europa, HackerOne.

- Published: date not stated
- Original: <https://hackerone.com/reports/309531>
- Preserved from: https://hackerone.com/reports/309531 (browser) on 2026-09-14
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

118

#309531

Stored XSS in Snapmatic + R Editor comments

Report

Summary by Rockstar Games

[[image: image]](https://hackerone.com/rockstargames)

Summary provided by the Researcher, @europa .

I requested the disclosure of what I hope is the final report regarding stored cross-site-scripting vulnerabilities on the Rockstar Games SocialClub, to also allow me to summarize the research that went into the other 5 reports. Have fun!

Report #1

The 6-months adventure into researching and bypassing the SocialClub WAF begun with a simple discovery at first: while the WAF was removing anything enclosed in `<.*`, some **control characters** (`\b \f \n \r \t`) weren't being taken into account when injecting a `<`, allowing an adversary to create a malicious payload in the simple form of `<\t`.

A fix was deployed to **remove anything following a** `<`.

Report #2

Two weeks after the fix, I ended up discovering what would soon become a “head-scratching” mystery: injecting a **single** `%` in the payload would bypass the filter entirely and force the back-end to somehow produce an unescaped `<` along with the escaped one.

The original payload was complex and confusing, and it led me to the wrong conclusion that over-consumption flaws were to blame, but as analysis proceeded, it was finally discovered that the culprit was the **simple, single** `%`.

The final payload

```
<%&lt;script/src=//...?
```

produced an output of

```
&lt;%<script/src="//..." <="" p="">
```

from the back-end.

A fix was deployed and the WAF rules were made more strict, defeating all attempts with a 302 redirect to an error page.

Report #3

Two months after the last fix, I discovered how the WAF wouldn't account for Full-Width and Small-Forms variants which, chained with the `%` confusion from the second report would again trick the back-end into producing a valid output: indeed, giving **`U+FF1C`** or **`U+FE64`** as the input would pass the WAF and the back-end would transform both into `<`. This is called a best-fit match flaw and it usually happens on Windows-powered technology stacks, where one of the processing layers fails to properly account for missing characters in destination codepages.

The payload `\uFE64%\uFF1Cscript/src=//...?`, evaded the WAF and produced

```
&lt;%<script/src="//...?" class="badLink"
```

in the HTML page.

A first fix was deployed preventing both script injections and DOM events manipulation, both of which I was able to bypass after a few days using a combination of **control chars, percentages, breaks, and exotic function invokation**. The payload `\uFF1C%\uFE64input/autofocus onfocus\b='[1].find(alert)'` successfully bypassed the new filters and popped an alert before the report was closed as resolved, allowing the team to look for a better solution in time. A second, stronger fix was deployed and the WAF rules were made even stricter prohibiting any combination of direct or indirect forms of `<` and `%` in suspicious contextes, plus any shape or form of `onXXX` DOM events.

Report #4

The new WAF rules prevented any kind of injection: no useful HTML elements, no DOM events. Anything went straight to /dev/null. After spending a few weeks in trial & error tests, I remembered how the payload from **report #3** would have a `badLink class` added to it, as the back-end detected a suspicious URI in the comment and would strike it out and prevent it from becoming clickable.

After weeks of tests, in a few hours I was able to chain ***eight* different techniques** to go through the WAF, the back-end filter, and the client-side Javascript filter:

- using `<>` to separate "trigger words" in order to turn them "invisible" to the WAF (ie: `&<>lt;`). The *back-end* would remove it for me.
- using `\u0025` instead of `%` which would now trigger the WAF
- using the unaccounted for `MATH` MathML element
- using control characters (`\n \t \b \r \f` from **report #1**) to break element names to trick the *back-end* (not the WAF) into reassembling them in output (ie: `<m\bath` instead of `<math`)
- using the `xml:base` attribute instead of the usual `href` to specify a Javascript URI
- injecting quotes to mess up the output from the back-end
- using an innocuos `href=#` to make everything following the payload clickable
- using a **fake URL** enclosed in `[]` to exploit a flaw in the rendering engine in the back-end that would cause it to move the payload outside of the "badUrl" element and place it where we could use it

The final payload was `&<>lt;%&<>lt;m\bath xml:base=\"j<>avascript:alert(document.domain)//\" href=#\" [bad.url.pls]` which produced

```
&lt%<math xml:base="javascript:alert(document.domain)//" href="#" x="" class="badLink">[bad.url.pls]
```

As a bonus note, this led to the discovery of a particular payload that would render a newsfeed comment **un-repliable and un-deletable**. Both flaws were fixed with better rules, and by preventing the back-end from stripping "*conveniently-placed*" tags and control characters.

Report #5

Somewhat less-related to the SocialClub per sé, this was a variation on **report #3** where it was discovered that Snapmatic and R Editor comments would go a different validation flow than any other entry, and the best-fit matchings would once again act up but on a different codepage this time, when using **Left-Angle brackets** `U+3008 " "` from the Cjk Symbols and Punctuation block, and **Left-pointing Angle brackets** `U+2329 " "` from the Miscellaneus Technical block.

While the Snapmatic/R Editor back-end would block `U+FF1C` and `U+FE64`, the other two would go through and get "matched" to `<` somewhere in the web technology stack. My last payload was `script/src=//...?` and it was promptly fixed in both its variations.

Conclusions

The Rockstar Games team is amazing. My first duplicate report was with them back in September and if it wasn't for @jmarshall reacting so politely to my unjustified noobish irk to a duplicate I

would've probably dropped bug bounties altogether.

It's been great to be involved all these months into researching new things and approaches—failing for weeks at a time allowed me to learn new techniques and extremely peculiar quirks I now feel ready to share with the community. I still go back and try new ideas as of today, so far without success. Which is great.

Ad maiora!

Summary by europa

[[image: image]](https://hackerone.com/europa)

I requested the disclosure of what I hope is the final report regarding stored cross-site-scripting vulnerabilities on the Rockstar Games SocialClub, to also allow me to summarize the research that went into the other 5 reports. Have fun!

Report #1

The 6-months adventure into researching and bypassing the SocialClub WAF begun with a simple discovery at first: while the WAF was removing anything enclosed in `<.*`, some **control characters** (`\b \f \n \r \t`) weren't being taken into account when injecting a `<`, allowing an adversary to create a malicious payload in the simple form of `<\t`.

A fix was deployed to **remove anything following a `<`**.

Report #2

Two weeks after the fix, I ended up discovering what would soon become a “head-scratching” mystery: injecting a **single `%`** in the payload would bypass the filter entirely and force the back-end to somehow produce an unescaped `<` along with the escaped one. The original payload was complex and confusing, and it led me to the wrong conclusion that over-consumption flaws were to blame, but as analysis proceeded, it was finally discovered that the culprit was the **simple, single `%`**.

The final payload

```
<%&lt;script/src=//...?
```

produced an output of

```
&lt;%<script/src="//..." <="" p="">
```

from the back-end.

A fix was deployed and the WAF rules were made more strict, defeating all attempts with a 302 redirect to an error page.

Report #3

Two months after the last fix, I discovered how the WAF wouldn't account for **Full-Width** and **Small-Forms** variants which, chained with the `%` confusion from the second report would again trick the back-end into producing a valid output: indeed, giving **`U+FF1C`** **or** **`U+FE64`** as the input would pass the WAF and the back-end would transform both into `<`. This is called a best-fit match flaw and it usually happens on Windows-powered technology stacks, where one of the processing layers fails to properly account for missing characters in destination codepages.

The payload `\uFE64%\uFF1Cscript/src=//...?`, evaded the WAF and produced

```
&lt;%<script/src="//...?" class="badLink"
```

in the HTML page.

A first fix was deployed preventing both script injections and DOM events manipulation, both of which I was able to bypass after a few days using a combination of **control chars, percentages, breaks, and exotic function invokation**. The payload `\uFF1C%\uFE64input/autofocus onfocus\b='[1].find(alert)'` successfully bypassed the new filters and popped an alert before the report was closed as resolved, allowing the team to look for a better solution in time. A second, stronger fix was deployed and the WAF rules were made even stricter prohibiting any combination of direct or indirect forms of `<` and `%` in suspicious contextes, plus any shape or form of `onXXX` DOM events.

Report #4

The new WAF rules prevented *any* kind of injection: no useful HTML elements, no DOM events. Anything went straight to /dev/null. After spending a few weeks in trial & error tests, I remembered how the payload from **report #3** would have a `badLink` class added to it, as the back-end detected a suspicious URI in the comment and would strike it out and prevent it from becoming clickable.

After weeks of tests, in a few hours I was able to chain ***eight* different techniques** to go through the WAF, the back-end filter, and the client-side Javascript filter:

- using `<>` to separate "trigger words" in order to turn them "invisible" to the WAF (ie: `&<>lt;`). The *back-end* would remove it for me.
- using `\u0025` instead of `%` which would now trigger the WAF
- using the unaccounted for `MATH` MathML element
- using control characters (`\n \t \b \r \f` from **report #1**) to break element names to trick the *back-end* (not the WAF) into reassembling them in output (ie: `<m\bath` instead of `<math`)
- using the `xml:base` attribute instead of the usual `href` to specify a Javascript URI
- injecting quotes to mess up the output from the back-end
- using an innocuos `href=#` to make everything following the payload clickable
- using a **fake URL** enclosed in `[]` to exploit a flaw in the rendering engine in the back-end that would cause it to move the payload *outside* of the "badUrl" element and place it where we could use it

The final payload was `&<>lt;%&<>lt;m\bath xml:base=\"j<>avascript:alert(document.domain)//\" href=#\" [bad.url.pls]` which produced

```
&lt%<math xml:base="javascript:alert(document.domain)//" href="#" x="" class="badLink">[bad.url.pls]
```

As a bonus note, this led to the discovery of a particular payload that would render a newsfeed comment **un-repliable and un-deletable**. Both flaws were fixed with better rules, and by preventing the back-end from stripping *"conveniently-placed"* tags and control characters.

Report #5

Somewhat *less-related* to the SocialClub per sé, this was a variation on **report #3** where it was discovered that Snapmatic and R Editor comments would go a different validation flow than any other entry, and the best-fit matchings would once again act up but on a different codepage this time, when using **Left-Angle brackets** `U+3008 " "` from the Cjk Symbols and Punctuation block, and **Left-pointing Angle brackets** `U+2329 " "` from the Miscellaneus Technical block.

While the Snapmatic/R Editor back-end would block `U+FF1C` and `U+FE64`, the other two would go through and get "matched" to `<` somewhere in the web technology stack. My last payload was `script/src=//...?` and it was promptly fixed in both its variations.

Conclusions

The Rockstar Games team is amazing. My first duplicate report was with them back in September and if it wasn't for @jmarshall reacting so politely to my unjustified noobish irk to a duplicate I would've probably dropped bug bounties altogether. It's been great to be involved all these months into researching new things and approaches—failing for weeks at a time allowed me to learn new techniques and extremely peculiar quirks I now feel ready to share with the community. I still go back and try new ideas as of today, so far without success. Which is great.

Ad maiora!

Timeline

[[image: europa]](https://hackerone.com/europa)

europa

submitted a report to **Rockstar Games**.

January 26, 2018, 11:36am UTC

jmarshall

Rockstar Games staff

.

January 26, 2018, 8:48pm UTC

jmarshall

Rockstar Games staff

changed the status to **Triaged**.

January 26, 2018, 8:48pm UTC

europa

.

Updated January 27, 2018, 9:52am UTC

jmarshall

Rockstar Games staff

.

February 16, 2018, 2:24am UTC

Rockstar Games

rewarded europa with a bounty.

February 16, 2018, 2:24am UTC

europa

.

February 16, 2018, 9:30am UTC

europa

.

Updated March 3, 2018, 6:26pm UTC

europa

.

March 19, 2018, 11:17am UTC

jmarshall

Rockstar Games staff

closed the report and changed the status to **Resolved**.

March 19, 2018, 3:29pm UTC

europa

requested to disclose this report.

April 8, 2018, 1:42pm UTC

jmarshall

Rockstar Games staff

.

April 9, 2018, 2:05pm UTC

europa

.

April 9, 2018, 2:09pm UTC

europa

.

April 9, 2018, 5:55pm UTC

jmarshall

Rockstar Games staff

.

April 19, 2018, 3:25pm UTC

europa

.

April 19, 2018, 3:38pm UTC

jmarshall

Rockstar Games staff

agreed to disclose this report.

April 19, 2018, 10:14pm UTC

This report has been disclosed.

April 19, 2018, 10:14pm UTC

Recovery notes

Source evidence recovered on 2026-09-14. The earlier source capture (SHA-256 `98d0deda23369bd40f13201e8115bc802d5b2ec5948b766fd782a7143339af9c`) is no longer available. This publication uses a separately preserved capture of the same document recorded on 2026-09-14 (SHA-256 `f14fbe267fe7ed3feb73470db772f1927fcd13d9ac4b8f367062e12905c84550`). The exposed summaries were checked against this fresh capture. Altered HTML entities and Unicode characters have been restored exactly, and payload examples are preserved as inert code. The report and comment bodies remain unavailable in the public page, so this is still a partial capture. The missing earlier capture remains documented in the archive history.