Journal of Electronical & Cyber Defence
Vol. 9, No. 2, 2021, Serial No. 34

# Automatic XSS Exploit Generation Using Grammatical Evolution

A. Moghaddasi, M. Bagheri*

*Imam Hossein Comprehensive University

(Received: 07/06/2020, Accepted: 26/10/2020)

**ABSTRACT**

*Fuzzers can reveal vulnerabilities in the software by generating test input data and feeding inputs to the software under test. The approach of grammar-based fuzzers is to search in the domain of test data which can be generated by grammar in order to find an attack vector with the ability to exploit the vulnerability. The challenge ahead of the fuzzers is the very large or infinite search domain as finding the answer in such a domain is hard. Grammatical Evolution (GE) is one of the evolutionary algorithms that can utilize grammar to solve the search problem. In this research, a new approach for generation of fuzz test input data by using grammatical evolution is introduced to exploit the cross-site scripting vulnerabilities. For this purpose, a grammar for generating XSS attack vectors is presented and a fitness calculation function is proposed to guide the GE in the search for exploitation. This method has realized the automatic exploitation of vulnerability with the black-box approach. The results of this research show 19% improvement in the number of vulnerabilities discovered compared to the white-box method of NAVEX and black-box ZAP tool, without any false positives*.

**Keywords:** Vulnerability, Exploitation, Cross Site Scripting (XSS), Grammatical Evolution, Fuzzer, Fuzz Testing

* Corresponding Author Email: m.bagheri@ihu.ac.ir

نشریه علمی "پدافند الکترونیکی و سایبری"
سال نهم، شماره ۲، تابستان ۱۴۰۰، ص ۱۱۹-۱۰۱

علمی – پژوهشی

# بهره‌برداری خودکار آسیب‌پذیری تزریق اسکریپت با استفاده از تکامل گرامری

علی مقدسی[۱]، مسعود باقری[۲*]

۱- دانشجوی دکتری، ۲- استادیار، دانشگاه جامع امام حسین(ع)

(دریافت: ۱۳۹۹/۰۶/۱۷، پذیرش: ۱۳۹۹/۰۸/۰۵)

## چکیده

فازرها می‌توانند از طریق تولید ورودی‌های آزمون و اجرای نرم‌افزار با این ورودی‌ها، آسیب‌پذیری‌ها را در نرم‌افزار آشکار کنند. رویکرد فازرهای مبتنی بر گرامر، جستجو در دامنه داده‌های قابل‌تولید توسط گرامر به‌منظور یافتن یک بردار حمله با قابلیت بهره‌برداری از آسیب‌پذیری است. چالش این فازرها، دامنه جستجوی بسیار بزرگ یا نامحدود هست و یافتن پاسخ در این دامنه یک مسئله سخت است. تکامل گرامری یکی از الگوریتم‌های تکاملی است که می‌تواند برای حل مسئله جستجو از گرامر استفاده نماید. در این تحقیق با استفاده از تکامل گرامری یک رویکرد جدید جهت تولید داده ورودی آزمون فاز به‌منظور بهره‌برداری از آسیب‌پذیری تزریق اسکریپت معرفی شده است. به این منظور یک روش استنتاج گرامر تزریق اسکریپت از روی بردارهای حمله ارائه شده است و یک تابع محاسبه برازندگی جهت هدایت تکامل گرامری برای جستجوی بهره‌برداری نیز ارائه گردیده است. این روش، بهره‌برداری خودکار از آسیب‌پذیری تزریق اسکریپت را با رویکرد جعبه سیاه محقق ساخته است. با روش این تحقیق ۱۹٪ بهبود در تعداد اکسپلویت‌های کشف‌شده نسبت به روش جعبه سفید ناوکس و ابزار جعبه سیاه زپ و بدون هیچ اتهام غلط، به‌دست آمده است.

**کلیدواژه‌ها:** آسیب‌پذیری، بهره‌برداری از آسیب‌پذیری، تزریق اسکریپت یا XSS، تکامل گرامری، فازر، آزمون فاز

## ۱- مقدمه

آسیب‌پذیری تزریق اسکریپت[۱] یا XSS همچنان در بین چند آسیب‌پذیری اصلی برنامه‌های وب قرار دارد [۱]. در موضوع تشخیص حمله تزریق اسکریپت تحقیقات زیادی با رویکرد صرفاً کشف آسیب‌پذیری انجام شده است [۶-۲]. اما چنانچه یک آسیب‌پذیری کشف شود ولی امکان بهره‌برداری[۲] از آن وجود نداشته باشد یک اتهام غلط[۳] محسوب می‌شود. منظور از بهره‌برداری این است که حداقل یک بردار حمله ارائه گردد که بتواند آسیب‌پذیری کشف‌شده را به‌طور موفقیت‌آمیزی مورداستفاده قرار دهد. چنین بردار حمله‌ای یک اکسپلویت نامیده می‌شود. لازمه بهره‌برداری این است که اول کشف آسیب‌پذیری انجام شود و سپس امکان بهره‌برداری از آن بررسی شود. رویکرد تحقیقات دیگر با دیدگاه کاهش اتهام غلط بوده است که این کار را با بررسی امکان بهره‌برداری از آسیب‌پذیری کشف‌شده با روش‌های مختلف انجام داده‌اند [۱۰-۷]. روش برخی از این تحقیقات در این رویکرد، فقط آزمون یک بردار حمله ساده یا فهرستی از بردارهای حمله است که از قبل آماده شده‌اند [۱۱ و ۱۲]. روش‌های دیگر تلاش برای تولید یک اکسپلویت است [۱۶-۱۳].

آزمون فاز[۴] نقشی قابل‌توجه و مؤثر در کشف آسیب‌پذیری‌های نرم‌افزاری داشته است [۱۷]. چالش فازرها و روش‌های کشف آسیب‌پذیری، تولید داده‌های آزمون با توجه به الگوی قابل‌پذیرش ورودی‌های برنامه است. تولید داده آزمون مبتنی بر گرامر، میزان عدم پذیرش داده‌های آزمون را در ابتدای ورود به برنامه کاهش می‌دهد و احتمال موفقیت در بهره‌برداری از آسیب‌پذیری را بالاتر می‌برد.

رویکرد این تحقیق خودکارسازی تولید اکسپلویت از طریق آزمون فاز جعبه سیاه و با استفاده از تکامل گرامری[۵] است. در این تحقیق برای اولین بار ایده «استفاده از تکامل گرامری برای پیاده‌سازی آزمون فاز به‌منظور بهره‌برداری از آسیب‌پذیری تزریق اسکریپت» مطرح می‌گردد. برای تشخیص موفقیت یا عدم موفقیت تزریق اسکریپت، چهار معیار ارزیابی ارائه می‌شود و نحوه

*رایانامه نویسنده مسئول: m.bagheri@ihu.ac.ir

[1] Cross-site Scripting
[2] Exploit
[3] False Positive
[4] Fuzz Testing or Fuzzing
[5] Grammatical Evolution(GE)



ناشر: دانشگاه جامع امام حسین (ع)

محاسبه آن‌ها توضیح داده می‌شود. با کمک این معیارها و در جهت ارزیابی داده‌های ورودی آزمون، یک تابع محاسبه برازندگی ارائه شده و توضیح داده می‌شود. تزریق اسکریپت موردنظر در این تحقیق از نوع انعکاسی است، هرچند ازلحاظ نظری این امکان وجود دارد که روش این تحقیق را در مورد سایر انواع تزریق اسکریپت نیز انجام داد.

موارد اصلی که این تحقیق ارائه می‌دهد به‌صورت زیر هستند:

- یک رویکرد جدید برای کشف و بهره‌برداری از آسیب‌پذیری تزریق اسکریپت در برنامه‌های وب با استفاده از تکامل گرامری ارائه می‌گردد.

یک روش استنتاج گرامر ارائه شده و بر اساس آن یک گرامر به فرم BNF با قابلیت استفاده در فازرهای تولیدی ارائه می‌گردد.

- چهار معیار ارزیابی موفقیت تزریق اسکریپت معرفی می‌شود.
- بر اساس معیارهای مذکور، تابع محاسبه برازندگی معرفی می‌گردد.
- نتایج پیاده‌سازی این رویکرد، ارائه شده و موردبررسی و مقایسه قرار می‌گیرد.

ادامه این مقاله به این شرح است: ابتدا مروری بر کارهای انجام‌شده قبلی جهت کشف آسیب‌پذیری تزریق اسکریپت انجام می‌شود. سپس رویکرد پیشنهادی این تحقیق و پیاده‌سازی و بررسی نتایج آزمایش‌ها ارائه می‌گردد. درنهایت به نتیجه‌گیری و بیان کارهای آینده پرداخته خواهد شد.

## ۲- کارهای مرتبط

تحقیقات درباره کشف یا جلوگیری از آسیب‌پذیری اسکریپت در برنامه‌های وب سابقه طولانی دارد و مقالات زیادی دراین‌باره ارائه شده است. با توجه به رویکرد این تحقیق، برخی از کارهای مرتبط که از آزمون فاز استفاده کرده یا از گرامر برای کشف تزریق اسکریپت بهره برده‌اند مرور می‌گردد.

کیزان و همکارانش [۱۲] در مقاله خود ابتدا با کمک تحلیل آلودگی، مسیرهایی از یک مبدأ ورودی به یک حفره را شناسایی می‌کنند و آزمون‌های بعدی را فقط روی این مسیرها که احتمال آسیب‌پذیری XSS در آن هست انجام می‌دهند. روش این مقاله نیاز به متن برنامه دارد و همچنین تعداد بردار حمله ورودی آزمون در آن محدود است.

اوانسینی و سکاتو [۲] در مقاله خود موضوع کشف آسیب‌پذیری تزریق اسکریپت از دو روش الگوریتم ژنتیک و اجرای نمادین-عینی را موردبررسی قرار داده و پیشنهاد می‌کنند که برای افزایش کارایی و میزان پوشش فرآیند کشف این آسیب‌پذیری، بهتر است این دو روش باهم ترکیب شوند. این مقاله تنها می‌تواند آسیب‌پذیری‌هایی را کشف و بهره‌برداری کند که ساختار صفحه را تغییر می‌دهند.

گیو و همکارانش [۳] در مقاله خود افزایش کارایی آزمون فاز را برای پیدا کردن آسیب‌پذیری تزریق اسکریپت موردتوجه قرار داده‌اند. هدف این مقاله مستقیماً استفاده از گرامر برای تولید داده‌های ورودی آزمون نیست بلکه هدف ترمیم و بهینه‌سازی مجموعه بردارهای حمله موجود است. این مقاله ابتدا تعداد زیادی بردار حمله به‌عنوان ورودی اولیه فازر جمع‌آوری کرده است و سپس با بررسی ویژگی‌ها و گرامر و الگوی موجود در این بردارهای حمله اولیه اقدام به ترمیم و بهینه‌سازی این مجموعه کرده است تا حجم محاسبات لازم پایین آمده و کارایی فازر بالا برود.

هوانگ و همکارانش [۱۸] در مقاله خود پیشنهاد داده‌اند که به‌منظور بهبود عمل تولید داده آزمون در فرآیند آزمون فاز بجای استفاده از یک عملگر جهش در هر بار، به‌طور هم‌زمان از ترکیب چندین عملگر جهش برای تولید داده آزمون استفاده شود. این مقاله تلاشی برای هدایت فازر در حین جستجو برای بردار بهره‌برداری موفق ندارد.

محمدی و همکاران [۱۹ و ۲۰] به این مسئله پرداخته‌اند که حتی اگر برنامه‌ها به‌منظور جلوگیری از تزریق اسکریپت از توابع فرار از اسکریپت مثل escapehtml یا escapejs استفاده کرده باشند ولی اگر استفاده در جای خود نبوده و یا به ترتیب درست نبوده باشد کار بی‌فایده‌ای است و بازهم امکان XSS وجود خواهد داشت. این مقاله بیان می‌کند که مکان حمله تزریق اسکریپت در نقاط خاص در بخش URI یا بخش CSS و یا در Event ها قابل‌تفکیک است و این‌که یک حمله تزریق اسکریپت باید درنهایت شامل یک اجرای تابع جاوا اسکریپت یا یک عملگر تخصیص باشد. برای تولید بردارهای حمله تزریق اسکریپت یک گرامر مستقل از متن ارائه کرده‌اند. برای کاهش اتهام غلط، علاوه‌بر تولید داده‌های ورودی آزمون از روی گرامر، با استفاده از مرورگر بی‌سر JWebUnit [1] خروجی را واقعاً تست می‌کنند تا نتیجه اجرای اسکریپت انتخابی را ببینند. رویکرد این مقاله جعبه سفید است.

ژوشو و همکارانش [۲۱] مسئله کشف آسیب‌پذیری تزریق اسکریپت از طریق آزمون فاز را به‌طور اختصاصی در مورد برنامه‌های «وب ایمیل» بررسی نموده و برای این کار استفاده از گرامر را موردتوجه قرار داده‌اند تا نقاطی که عمل جهش می‌تواند

[1] Headless Browser

در آنجا اقدام نماید را مشخص کنند.

وانگ و همکارانش [۲۲] در مقاله خود ابتدا فهرستی از بردارهای اولیه حمله تزریق اسکریپت از منابع اینترنتی جمع‌آوری کرده و سپس روی آن‌ها یک روش یادگیری ساختارمند به‌کار می‌برند و درنتیجه یک گرامر منظم[1] که درواقع یک اتوماتای احتمالاتی است ایجاد کرده و از روی آن نهایتاً یک مدل مخفی مارکوف استخراج می‌کنند. این مدل برای تولید ورودی‌های آزمون استفاده می‌شود. این مقاله تست نهایی موفقیت یا عدم موفقیت بهره‌برداری را به‌صورت خودکار انجام نمی‌دهد.

دوچن و همکارانش [۴] در مقاله خود کشف آسیب‌پذیری تزریق اسکریپت را از طریق آزمون فاز به کمک الگوریتم ژنتیک انجام داده‌اند و برای این کار از گرامر بهره گرفته‌اند. برنامه وب را به‌صورت یک ماشین حالت مدل کرده‌اند و آزمون را از حالتی شروع می‌کنند که نقطه ورود تزریق اسکریپت از قبل کشف شده است. گرامر حمله مذکور به‌صورت یک گرامر مستقل از متن در مقاله تعریف شده است. از این گرامر در عملگرهای جهش و تقاطع استفاده می‌شود.

وانگ و همکارانش [۱۰] در مقاله خود کشف تزریق اسکریپت از نوع DOM را موردتوجه قرار داده‌اند و برای تولید داده ورودی آزمون از یک گرامر بهره برده‌اند. ابتدا با استفاده از روش ردیابی آلودگی[2] مکان تزریق اسکریپت را پیدا می‌کنند و سپس با توجه به متن، پارامترهای گرامر را مقداردهی کرده و به‌این‌ترتیب داده واقعی آزمون را تولید می‌کنند. این مقاله برای تست برنامه به متن آن احتیاج دارد.

اتلیداکیس و همکارانش [۲۳] اقدام به تست REST API ها از طریق آزمون فاز کرده‌اند. برای این کار یادگیری یک مدل آماری شامل ساختار درخواست‌های API و موارد وابسته به آن‌ها را انجام می‌دهند و سپس از این مدل آماری و از طریق راهبردهای جهش، ورودی‌های معتبر آزمون تولید می‌نمایند. بازخوردهای مناسب برای افزایش پوشش، به‌منظور انتخاب ورودی‌های مناسب‌تر برای پوشش بیشتر استفاده می‌شوند.

ابرلین و همکارانش [۲۴] در مقاله خود از روی ورودی‌های اولیه یک گرامر احتمالاتی استخراج می‌کنند. سپس از این گرامر اما با معکوس کردن احتمالات برای تولید داده ورودی آزمون استفاده می‌کنند به این شکل که قوانینی که در هنگام یادگیری دارای احتمال (تکرار) کمتری بوده‌اند را بیشتر استفاده می‌کنند تا میزان پوشش کد بیشتر شود. همچنین از الگوریتم تکاملی استفاده می‌کنند و در هر نسل، بهترین‌ها را انتخاب می‌کنند و با کمک آن‌ها ضرایب احتمال را در گرامر احتمالاتی بهبود می‌بخشند. به‌این‌ترتیب الگوریتم تکاملی به سمت تولید ورودی‌های آزمون با ویژگی‌های خاص هدایت می‌شود.

ذاکری و پارسا [۲۵] آزمون فاز برنامه‌هایی را که ورودی فایل می‌پذیرند مد نظر قرار داده و از روی فایل‌های نمونه اولیه یک گرامر در قالب مدل زبان استنتاج می‌کنند. این کار را به کمک شبکه عصبی و یادگیری عمیق انجام می‌دهند. سپس از این مدل برای تولید داده جدید ورودی آزمون استفاده می‌کنند.

هرچند کارهای انجام‌شده قبلی تااندازه‌ای به حل مسئله کمک کرده‌اند ولی هرکدام از آن‌ها چالش‌هایی دارند. روش‌هایی که جعبه سفید عمل کرده‌اند اولاً به زبان خاص وابسته هستند و ثانیاً در مواردی که متن برنامه در دسترس نیست کارایی ندارند. همچنین بسیاری از این روش‌ها چه در وضعیت جعبه سفید و چه جعبه سیاه صرفاً امکان وجود آسیب‌پذیری یا صرفاً وجود انعکاس را بررسی کرده‌اند و دارای اتهام غلط زیاد هستند. نکته مهم دیگر اینکه تخصص ویژه نفوذگران در اغلب روش‌ها نادیده گرفته شده است و این روش‌ها قابلیت رقابت با روش‌های دستی و حرفه‌ای را ندارند [۲۶]. در رویکرد پیشنهادی که در بخش بعد توضیح داده می‌شود تلاش شده است برای رفع این چالش‌ها یک راه‌حل ارائه گردد.

## ۳- رویکرد پیشنهادی: آزمون فاز با تکامل گرامری

همان‌طور که ذکر شد اغلب روش‌های کشف آسیب‌پذیری دارای اتهام غلط هستند یا قادر نیستند برای آسیب‌پذیری کشف‌شده، بهره‌برداری ارائه نمایند. برای کاهش اتهام غلط در فرآیند کشف آسیب‌پذیری، لازم است نحوه بهره‌برداری از آن معلوم شود. آزمون فاز می‌تواند به‌عنوان یک روش مؤثر برای بررسی چگونگی بهره‌برداری از آسیب‌پذیری استفاده شود. چالش فازرها، دامنه جستجو بسیار بزرگ یا نامحدود است و یافتن پاسخ در این دامنه یک مسئله ان‌پی‌سخت است[3].

در این تحقیق یک رویکرد جدید آزمون فاز برای کاهش اتهام غلط در کشف آسیب‌پذیری تزریق اسکریپت از طریق بررسی امکان بهره‌برداری از آسیب‌پذیری ارائه می‌گردد. قبل از معرفی رویکرد پیشنهادی ذکر موارد زیر لازم است.

هنگام آزمون فاز برنامه وب باهدف کشف آسیب‌پذیری تزریق اسکریپت، مانند هر آزمون فاز دیگری ابتدا باید ورودی‌های برنامه وب شناسایی شوند. مرحله بعدی در آزمون فاز پس از احصاء و

[1] Regular Grammar
[2] Taint Tracking

[3] در نظریه پیچیدگی، مسائل NP-Hard مسائلی هستند که سختی آن‌ها از مسائل چندجمله‌ای نامعین بیشتر است.

پیدا کردن ورودی‌ها، تزریق داده ورودی به‌قصد ایجاد سقوط [1] در برنامه است. در مورد آسیب‌پذیری تزریق اسکریپت انعکاسی، سقوط مترادف آن است که یک اسکریپت از ورودی به خروجی منعکس شود و در مرورگر اجرا گردد.

ازنظر این تحقیق، در یک حمله تزریق اسکریپت، صرف مشاهده انعکاس دلیل بر موفقیت حمله نیست و حتی شرط لازم هم نیست؛ بلکه ملاک موفقیت، اجرای صحیح اسکریپت است.

پیدا کردن محل انعکاس بردار حمله به‌عنوان نقطه آسیب‌پذیر، موضوع تحقیق بسیاری از محققین بوده است ولی در خصوص امکان بهره‌برداری از آن تلاش کمتری صورت گرفته است. بنابراین، در این تحقیق فرض بر این است که محل انعکاس مشخص است و مسئله اصلی، انعکاس یک «بردار حمله هدف» است زیرا اگر به پیدا کردن محل انعکاس نیز پرداخته شود فضای کافی برای بحث اصلی باقی نمی‌ماند. روش انتخابی این تحقیق، استفاده از آزمون فاز تولیدی [2] است و البته از مفاهیم و ابزارهای فازر جهشی [3] نیز استفاده شده است. علت انتخاب این روش این است که چنانچه برای تولید ورودی‌های یک برنامه، مدل یا الگو یا گرامری وجود داشته باشد انتخاب فازر تولیدی ارجح است زیرا ورودی‌های تولیدشده قواعد ورودی برنامه تحت آزمون را رعایت می‌کنند و بنابراین، احتمال موفقیت آن‌ها در عبور از فیلترهای اعتبارسنجی درون برنامه بیشتر خواهد بود.

رویکرد این تحقیق برای حل مسئله، استفاده از «تکامل گرامری» و تولید داده ورودی آزمون به‌وسیله یک گرامر است. تکامل گرامری می‌تواند علاوه بر رفتار ذاتی خود به‌عنوان یک روش تکاملی، از گرامر موجود نیز حداکثر بهره‌برداری را در تولید داده ورودی آزمون میسر سازد و همچنین می‌تواند ویژگی‌های آزمون فاز جهشی را هم‌زمان با آزمون فاز تولیدی به‌کار بگیرد.

البته اولین شرط برای استفاده از تکامل گرامری، در اختیار داشتن یک گرامر است. بنابراین، یکی از چالش‌های اصلی در این رویکرد، تأمین و ارائه یک گرامر مناسب است. به این منظور در این تحقیق، یک روش استنتاج گرامر ارائه می‌شود و در تکامل گرامری مورداستفاده قرار می‌گیرد.

با توجه به استفاده از تکامل گرامری در طرح پیشنهادی این تحقیق، ابتدا یک معرفی مختصر از آن ارائه می‌گردد و سپس طرح پیشنهادی تشریح خواهد گردید.

[1] Crash
[2] Generation Fuzz Testing
[3] Mutation Fuzzer

## ۳-۱- تکامل گرامری

روش انتخابی این تحقیق برای حل مسئله، استفاده از روش تکامل گرامری (GE) است. تکامل گرامری یک روش برنامه‌نویسی ژنتیک [4] است که توسط رایان و اونیل و همکارانشان در سال ۱۹۹۸ معرفی گردیده است [۲۷ و ۲۸]. در این روش، الگوریتم تکاملی برای جستجو و تولید اعضاء از یک گرامر استفاده می‌کند.

در تکامل گرامری، افراد جامعه ژنوتایپ [5] هستند که از روی هر ژنوتایپ توسط قواعد گرامر یک جمله یا فنوتایپ [6] تولید می‌گردد. هدف از تکامل گرامری این است که جمله‌ای از روی گرامر موجود ساخته شود که بیشترین (یا کمترین) میزان برازندگی را داشته باشد. محاسبه میزان برازندگی با انتخاب یک تابع برازندگی [7] مناسب محقق می‌گردد. تکامل گرامری در بین جمله‌های تولیدشده توسط گرامر به جستجو می‌پردازد تا جمله بهینه را پیدا نماید.

ایده تفکیک ژنوتایپ از فنوتایپ کمک می‌کند تا بتوان هر الگوریتم تکاملی ازجمله بهینه‌سازی ازدحام ذرات [8]، تکامل تفاضلی [9] و راهبردهای تکاملی [10] را علاوه بر الگوریتم ژنتیک که الگوریتم پیش‌فرض در تکامل گرامری است روی ژنوم‌ها انجام داد. شکل (۱) ساختار یک سامانه تکامل گرامری را نشان می‌دهد [۲۹]. ورودی‌های این سامانه عبارت‌اند از مسئله، گرامر و الگوریتم جستجو. مسئله مشخص می‌کند که از سامانه چه انتظاری وجود دارد و این کار با معرفی یک تابع برازندگی مناسب انجام می‌شود. گرامر مشخص می‌کند که خروجی‌های سامانه که افراد جامعه هستند به چه شکلی می‌باشند. الگوریتم جستجو طریقه ایجاد نسل‌های جامعه را معین می‌کند که این کار به‌طور مثال با معرفی عملگرهای لازم مانند عملگر جهش و عملگر تقاطع در الگوریتم ژنتیک مشخص می‌گردد. خروجی تکامل گرامری همانند برنامه‌نویسی ژنتیک یک (قطعه) برنامه است و به‌عبارت‌دیگر، جمله‌ای است که از روی گرامر ساخته شده و برازندگی مورد انتظار را برآورده می‌سازد.

الگوریتم جستجوی پیش‌فرض در تکامل گرامری همان الگوریتم ژنتیک است. عملگرهای تکاملی متعددی نیز برای تشکیل نسل، انتخاب اعضاء، تقاطع و جهش قابل‌استفاده است و

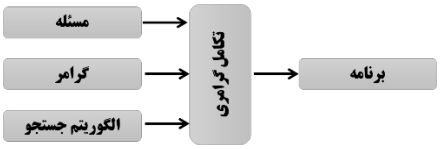
[4] برنامه‌نویسی ژنتیک، اقتباس‌شده از الگوریتم ژنتیک است و نوعی الگوریتم تکاملی است که در آن به‌جای ژن‌ها، افرادی (عناصری) که مورد تکامل قرار می‌گیرند برنامه‌های کامپیوتری هستند.

[5] Genotype
[6] Phenotype
[7] Fitness Function
[8] Particle Swarm Optimization
[9] Differential Evolution
[10] Evolution Strategies

توسط فنتون و اونیل و همکارانشان تشریح گردیده است [۱] [۳۰].

## ۳-۲- طرح پیشنهادی مبتنی بر تکامل گرامری

طرح پیشنهادی در این تحقیق استفاده از سامانه تکامل گرامری شکل (۱) است که اجزاء این سامانه به شرح زیر تأمین می‌گردند:




**شکل (۱):** ساختار یک سامانه تکامل گرامری [۲۹].

***مسئله:*** مسئله عبارت از بهره‌برداری از آسیب‌پذیری تزریق اسکریپت است. فنوتایپ‌ها بردارهای حمله هستند و جمله‌هایی هستند که توسط گرامر تولید می‌شوند و به‌عنوان داده ورودی آزمون به برنامه وب تزریق می‌شوند و به هریک از آن‌ها یک مقدار برازندگی اختصاص می‌یابد.

***گرامر:*** در اختیار داشتن یک گرامر مناسب یکی از ارکان استفاده از روش تکامل گرامری است. همان‌طور که ذکر شد در این تحقیق یک گرامر برای تولید بردارهای حمله تزریق اسکریپت استنتاج می‌گردد. این گرامر قادر است بردارهای حمله پیچیده و غنی تولید نماید که قابل‌مقایسه با بردارهای حمله دست‌ساز توسط نفوذگران خبره است و همین قابلیت را به فازر معرفی‌شده در این تحقیق، منتقل می‌نماید. روش پیشنهادی این تحقیق برای استنتاج گرامر، در ادامه این بخش توضیح داده می‌شود.

***الگوریتم جستجو:*** الگوریتم جستجو می‌تواند غیر از پیش‌فرض آن که الگوریتم ژنتیک است انتخاب شود. در طرح پیشنهادی این تحقیق از الگوریتم «راهبردهای تکاملی» برای این منظور استفاده شده است. انتخاب این الگوریتم به این دلیل است که در جریان این تحقیق آزمایش‌های زیادی با الگوریتم‌های متعدد انجام شده و این الگوریتم کارایی بالاتر نسبت به دیگران نشان داده است.

***برنامه:*** برنامه(خروجی)، یک بردار حمله دارای بیشترین برازندگی حاصل از انجام فرآیند تکامل است. چنانچه مقدار این برازندگی از یک آستانه تعریف‌شده در سامانه بالاتر باشد به‌عنوان بهره‌برداری از آسیب‌پذیری معرفی می‌گردد. در غیر این صورت نتیجه‌گیری می‌شود که آسیب‌پذیری مورد آزمون، قابل بهره‌برداری نیست.

***محاسبه برازندگی:*** ارائه تابع برازندگی یکی از مهم‌ترین ارکان الگوریتم تکاملی است. به همین دلیل نحوه محاسبه برازندگی به‌طور مشروح توضیح داده خواهد شد.

[۱] جهت پیاده سازی تکامل گرامری و تست روش این تحقیق از پروژه متن از PonyGE2 استفاده شده است. در این راستا نیاز است تغییراتی در متن آن داده شود که در انجام این تحقیق تغییرات لازم اعمال گردیده است

## ۳-۳- استنتاج گرامر

همان‌طور که در بخش‌های قبلی ذکر شد لازم است برای فازرهای تزریق اسکریپت یک گرامر مناسب تهیه شود. در این بخش به معرفی روشی برای تولید و ارائه یک گرامر به‌منظور تولید بردارهای حمله تزریق اسکریپت پرداخته می‌شود.

منابع متعددی در اینترنت وجود دارند که مجموعه ای از بردارهای حمله ارائه می‌کنند، به‌عنوان نمونه سایت owasp.org و پروژه fuzzdb قابل‌ذکر است. این منابع معتبر هستند زیرا به‌عنوان مرجع در مقالات مورداستفاده قرار می‌گیرند [۳۳-۳۱] و علاوه بر آن مورداستفاده متخصصین امنیتی و همچنین هکرها هستند. بنابراین، استنتاج یک گرامر با استفاده از بردارهای حمله موجود، در صورت امکان، کاملاً منطقی است.

روش پیشنهادی این مقاله، این بردارهای حمله را به‌عنوان ورودی می‌پذیرد و یک گرامر حمله از روی آن‌ها استنتاج می‌کند که قادر باشد علاوه بر بازتولید تمامی بردارهای حمله اولیه، بردارهای حمله جدید بسیاری نیز طبق گرامر تولید نماید.

شکل (۲) فرآیند استنتاج گرامر از روی بردارهای حمله ورودی را که در این مقاله پیشنهاد می‌شود نشان می‌دهد. مراحل مذکور به ترتیب زیر هستند:

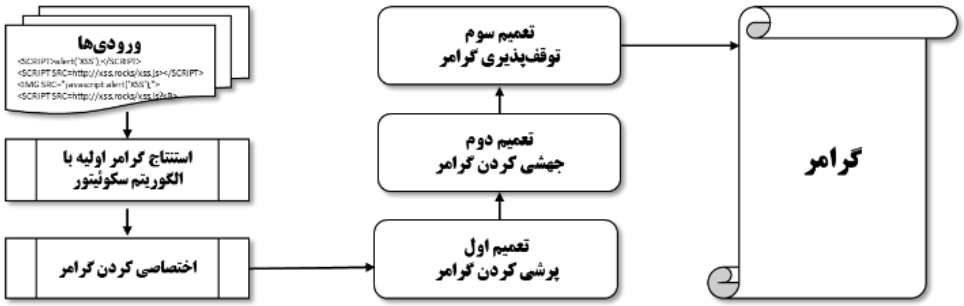
۱- ابتدا فهرست بردارهای اولیه انتخاب می‌شود و به‌عنوان ورودی به فرآیند داده می‌شود.
۲- با استفاده از الگوریتم سکوئیتور[۲] یک مرحله فشرده‌سازی روی ورودی‌های اولیه انجام می‌شود. فهرست فشرده‌شده در ظاهر شبیه یک گرامر مستقل از متن خواهد بود.
۳- اقدامات اولیه روی نتیجه مرحله قبل انجام می‌شود تا به یک گرامر قابل‌استفاده نزدیک‌تر شود. این اقدامات به نام اختصاصی کردن گرامر مورداشاره قرار خواهد گرفت.
۴- از این مرحله به بعد عملگرهای تعمیم اعمال خواهند گردید. تعمیم اول پرشی کردن گرامر است که امکان بازگشتی بودن و قابلیت تولید جمله‌های جدید غیر از ورودی‌های اولیه را برای گرامر فراهم می‌کند.
۵- تعمیم بعدی جهشی کردن گرامر است که امکان ایجاد تنوع در جمله‌ها را فراهم می‌کند و قابلیت فازرهای جهشی را نیز به فازر تولیدی می‌افزاید.

[2] Sequitur algorithm

۶- تعمیم بعدی توقف‌پذیری است که امکان ایجاد جمله‌های کوتاه را نیز علاوه بر جمله‌های بلند به گرامر اضافه می‌کند.

۷- در پایان، خروجی به‌صورت یک گرامر مستقل از متن قوی و مناسب برای فازر تحویل می‌شود.

در ادامه تمامی مراحل فوق توضیح داده خواهند شد.




**شکل (۲):** فرآیند استنتاج گرامر از روی ورودی‌ها.

### ۳-۳-۱- استنتاج گرامر اولیه

الگوریتم سکوئیتور (یا الگوریتم نویل-منینگ[1]) یک متن حاوی تعدادی جمله را که هرکدام در خطوط جداگانه قرار دارند به‌عنوان ورودی می‌گیرد و آن را فشرده می‌کند. این الگوریتم، متن فشرده‌شده را به‌صورت یک گرامر مستقل از متن به فرم BNF ارائه می‌کند که از روی این گرامر می‌توان تمام جملات موجود در متن را استخراج نمود [۳۴]. به همین دلیل این الگوریتم که نوعی الگوریتم حریص محسوب می‌شود یکی از روش‌های مورداستفاده در استنتاج گرامر به شمار می‌آید [۳۵]. اهمیت این الگوریتم به این دلیل است که در زمان و مکان خطی است و با سرعت بسیار بالا کار می‌کند.

نحوه کار الگوریتم سکوئیتور این است که ساختارهای تکراری را در متن پیدا می‌کند و آن‌ها را به‌صورت قاعده تولید[2] در گرامر معرفی می‌کند. هر قاعده گرامر(غیر از اولین قاعده) حداقل باید دو بار استفاده شده باشد. خروجی این الگوریتم به شکلی است که قاعده تولید اول، وظیفه بازسازی جمله اصلی (شامل همه جمله‌های اولیه در یک سطر) را به عهده دارد و سایر قواعد تک گزینه‌ای هستند. قواعد به‌صورت R0، R1،R2، ... نام‌گذاری می‌شوند و R0 قاعده اول و قاعده شروع کار الگوریتم است.

این الگوریتم فرض می‌کند ورودی تنها یک جمله است. اگر ورودی یک فایل متنی شامل فهرستی از جمله‌ها باشد این الگوریتم همه جمله‌ها را در یک خط قرار داده و یک جمله بزرگ متشکل از همه جمله‌های مستقل ایجاد می‌کند و سپس اقدام به فشرده‌سازی می‌نماید. درنتیجه، تفکیک بردارهای حمله مخدوش می‌شود و این گرامر در هنگام تولید جمله، تنها می‌تواند یک جمله تولید کند که این جمله شامل همه بردارهای حمله اولیه در یک سطر طولانی خواهد بود.

خروجی الگوریتم سکوئیتور به شکل پیش‌فرض آن مناسب نیست و لازم است تغییراتی در آن داده شود که اختصاصی کردن الگوریتم نامیده می‌شود. به این منظور لازم است تغییراتی در خروجی الگوریتم سکوئیتور اعمال گردد، ازجمله اینکه بعضی عبارت‌ها در بردارهای حمله وجود دارند که نباید به آن‌ها اجازه شکستن داد، زیرا هرچند این کار برای فشرده‌سازی خوب است ولی برای گرامر تزریق اسکریپت بی‌معنی است و مطلوب نیست. برای کاراکتر فضای خالی (space) و خط جدید (new line) نیز باید تغییرات مناسب صورت گیرد.

با اعمال تغییرات لازمه، هر قاعده تولید درنهایت در یک سطر جداگانه قرار می‌گیرد و قاعده تولید اول تعداد زیادی(به‌اندازه تعداد جمله‌ها) گزینه دارد که با علامت ( | ) از هم تفکیک‌شده‌اند. این اقدام باعث خواهد شد که گرامر تولیدی، دیگر تنها یک جمله تولید نمی‌کند بلکه قادر خواهد بود هریک از جمله‌های اولیه را به‌طور جداگانه در خطوط جدا از هم بازتولید نماید.

تغییرات اعمال‌شده در این مرحله پیش‌نیاز ادامه کار است اما این تغییرات کافی نیستند زیرا هنوز یک مشکل بزرگ در این گرامر وجود دارد و آن‌هم این است که این گرامر، بازگشتی نیست و به همین دلیل فقط قادر است همان بردارهای حمله اولیه خودش را تولید کند و بیش از آن خروجی جدیدی نخواهد داشت. ولی نیاز است که بتوان از روی این گرامر، بردارهای حمله بی‌شماری تولید کرد و به‌عنوان داده ورودی آزمون از آن استفاده کرد. برای حل این معضل یک عمل تعمیم به نام «پرشی کردن» روی گرامر انجام می‌شود. تغییرات اعمال‌شده در الگوریتم استنتاج

[1] Nevill-Manning algorithm
[2] Production Rule

به‌منظور پرشی کردن گرامر در ادامه توضیح داده شده است.

### ۳-۳-۲- تعمیم اول: پرشی کردن گرامر

همان‌طور که گفته شد تاکنون گرامری ایجاد شده است که فقط همان تعداد جملات قبلی را تولید می‌کند و دقیقاً در قاعده اول به تعداد جملات ورودی، قاعده تولید وجود دارد (که با علامت | از هم تفکیک‌شده‌اند) و درواقع فقط یک مرحله فشرده‌سازی انجام شده است، درحالی‌که هدف این تحقیق این است که قادر باشد تعداد بسیار زیادی جمله تولید کند فلذا لازم است تغییراتی در قواعد داده شود. این تغییرات اولاً باید کمک کند که جملات جدید قابل‌تولید باشند و ثانیاً نباید با مجموعه بردارهای حمله ورودی در تناقض باشد. برای انجام این کار راه‌حل‌های متعددی می‌توان پیشنهاد داد. روشی که در این تحقیق انتخاب شده است پرشی کردن گرامر است و به‌صورت زیر پیشنهاد می‌گردد:

**Algorithm 1:** Jumpify the Grammar

**Input :** $G_i$ , a grammar in BNF form
**Output :** $G_o$ , the Jumpified version of $G_i$ , in BNF form

```
1:  Go ← Empty string
2:  RuleIndex ← 0
3:  while true do
4:    RuleName ← '<R'+ RuleIndex+'>'      // <R0> or <R1> or <R2> or……
5:    RuleBranches ← Find all occurences of RuleName in Gi that end to '|' or 'new line'
6:    if RuleBranches is empty then          //Rules are finished
7:      break
8:    end if
9:    defaultrule ← FINDDEFAULTRULE(RuleBranches)  // <R1>::= or <R2>::= or <R3>::= ……
10:                                          // Each RuleName has only one default rule
11:   defaultproduction = ← RIGHTHANDSIDE(defaultrule)  // Right part of "::=" operator in rule
12:   production ← Empty string
13:   for each branch in RuleBranches do
14:     if branch ≠ defaultrule then
15:       //there may be multiple <Rx> in this branch, like <R1>...<R1>..<R1>..\n
16:       while true do
17:         TOKENIZE(branch) with char '|' and 'new line'
18:         subbranch ← First Token from branch
19:         // Subbranches with empty right hand side are not accepted
20:         if RIGHTHANDSIDE(subbranch) ≠ Ø then
21:           production ← append(' | ' + defaultproduction+ RIGHTHANDSIDE(subbranch))
22:         end if
23:         branch ← Find next RuleName in branch
24:         if branch is empty then          //subbranches are finished
25:           break
26:         end if
27:       end while
28:     end if
29:   end for
30:   production ← defaultrule+production
31:   Go ← append (production+'new line')
32:   INCREMENT RuleIndex
      end while
```

**الگوریتم (۱):** الگوریتم پرشی کردن گرامر.

در هر غیر ترمینال در ادامه آن به‌جای این‌که همین قاعده فعلی ادامه پیدا کند، پرش به‌جای دیگری در بین قواعد که همین غیر ترمینال را دارد انجام شود و ازآنجا به بعد ادامه پیدا کند. این کار کمک می‌کند جمله‌های جدید قابل ایجاد باشند بدون این‌که گرامر اولیه به‌صورت فاحش نقض شود. مثلاً اگر قاعده این است:

happy <R1> day to you

و تا happy رفته است و قرار است <R1> جاگذاری شود و بعدازآن day to you بیاید، از همین لحظه به قاعده زیر پرش می‌شود:

I am <R1> of you

و بعد از R1 عبارت of you گذاشته می‌شود یعنی قاعده به شکل زیر تغییر پیدا می‌کند:

happy <R1> of you

البته پرش در جریان اجرا انجام نمی‌شود بلکه پرش‌ها به‌صورت قواعد جدید معرفی شوند یعنی حالت‌های مختلف در قواعد موجود به‌صورت قواعد جدید با کاراکتر | به آن‌ها اضافه شود. این

کار اجازه تولید جمله جدید بدون نقض فاحش گرامر اولیه را می‌دهد.

منطق تعمیم پرشی کردن این است که تحمیل قاعده خاصی به فهرست جمله‌های موجود میسر نیست و هرچه هست باید از دل خود فهرست استخراج شود. واضح‌ترین واقعیت موجود در این جمله‌ها، ترتیب و توالی عبارات است. اگر این ترتیب و توالی‌ها قبلاً موفق بوده‌اند و توانسته‌اند کار مفیدی انجام دهند بنابراین، باید به‌این‌ترتیب و توالی‌ها اهمیت داده و تعمیم را بر اساس آن‌ها تعریف کرد. پرشی کردن درواقع همان ارزش قائل شدن به ترتیب و توالی‌ای است که در جمله‌های فهرست اولیه وجود دارد و این ایده را ایجاد می‌کند که جمله‌های تولیدشده توسط چنین گرامری بتوانند موفقیت‌های جمله‌های اولیه را تکرار کنند.

این تعمیم، گرامر موجود را به یک گرامر بازگشتی تبدیل می‌کند. نحوه انجام این تعمیم روی گرامر اختصاصی شده مرحله قبل در الگوریتم (۱) آمده است.

### ۳-۳-۳- تعمیم دوم: جهشی کردن گرامر

پرشی کردن گرامر یک تعمیم قوی و مؤثر است که قابلیت بازگشتی بودن به گرامر می‌دهد و به فازر تولیدی این امکان را می‌دهد که بردارهای حمله متنوع به تعداد لازم از روی گرامر تولید نماید. اما این گرامر امکان ایجاد جهش در قواعد را ندارد و بنابراین، فازری که از آن استفاده می‌کند از امکان جهش در داده‌های ورودی محروم است. این در حالی است که ایجاد جهش در ورودی‌ها تأثیر بسیار موفق خود را در فازرهای جهشی نشان داده است.

منظور از جهش در فازرها این است که بخشی از جمله ورودی به‌طور تصادفی تغییر کند. این تغییر ممکن است باعث ایجاد خطا در برنامه تحت آزمون شود و منجر به کشف آسیب‌پذیری گردد. برای این‌که عمل جهش هوشمندانه‌تر و مؤثرتر انجام گیرد به‌طور معمول مرزهایی در جمله ورودی مشخص می‌شود و تغییرات جهش در آن مرزها انجام می‌پذیرد. بر این اساس در این مرحله یک تعمیم دیگر به گرامر اعمال می‌شود تا آن را جهشی کند. مراحل این تعمیم در الگوریتم (۲) نشان داده شده است. این الگوریتم به ابتدا و انتهای همه غیرترمینال‌ها (غیر از غیرترمینال اول) یک مرز اضافه می‌کند و این مرز نیز به‌عنوان یک قاعده جداگانه به گرامر اضافه خواهد شد. با کمک این مرزها و تغییراتی که در اثر آن‌ها ایجاد می‌شود جهش تحقق می‌یابد.

در الگوریتم (۲) ابتدا همه قواعد R، غیر از قاعده R0، به M تبدیل می‌شوند. به‌طور مثال <R1> به <M1> و <R2> به <M2>. سپس مجدداً قواعد جدید M به R برمی‌گردند ولی قبل و بعد از آن‌ها یک قاعده <RBORDER> اضافه می‌شود.

### ۳-۳-۴- تعمیم سوم: توقف‌پذیری گرامر

اگر تنها تعمیم‌های انجام‌گرفته روی گرامر شامل پرشی کردن و جهشی کردن باشند، یک مشکل به وجود خواهد آمد. گرامری که در ابتدا بازگشتی نبود و نمی‌توانست هیچ قاعده جدیدی تولید کند، اکنون ممکن است به‌شدت بازگشتی شده باشد به‌طوری‌که جمله‌های تولیدشده بسیار طویل باشند و این احتمال نیز وجود دارد که فرآیند تولید جمله در بعضی موارد بی‌پایان باشد. بنابراین، مناسب است امکانی به گرامر اضافه شود که شانس تولید جمله‌های کوتاه نیز برای گرامر (و فازری که از آن استفاده می‌کند) وجود داشته باشد. پیشنهادی که این مقاله برای این عملگر تعمیم معرفی می‌کند این است که یک گزینه <RFINISH> به انتهای تمام قواعد تولید غیر از قاعده اول اضافه شود. این کار را می‌توان در همان زمان پرشی کردن انجام داد.

اجرای الگوریتم‌های (۱) و (۲) فرآیند شکل (۲) را تحقق می‌بخشد.

**Algorithm 2:** Add Mutation Capability to the Grammar

**Input :** $G_i$ , a grammar in BNF form
**Output :** $G_o$ , the version of $G_i$ with mutation capability, in BNF form

```
1:   Go ← Empty string
2:   for each production in Gi do
3:     if first production then      // Do this only for <R0>
4:       Replace all occurences of '<R' to '<M'
5:       Replace back '<M0>' to '<R0>'
6:       Go ← append (production)
7:     else
8:       Go ← append (production)
9:     end if
10:  end for
11:  // Now define <M1>, <M2>, … based on <R1>, <R2>, …
12:  for index from 1 to Number_of_Rules do
13:    Go ← append (<Mindex> ::= <RBORDER><Rindex><RBORDER>)
14:  end for
```

**الگوریتم (۲):** الگوریتم جهشی کردن گرامر.

قبلاً اشاره شد که الگوریتم سکوئیتور یک الگوریتم خطی و از مرتبه O(n) است. می‌توان نشان داد که الگوریتم (۱) ارائه‌شده در این تحقیق نسبت به تعداد قواعد تولید ورودی آن از مرتبه $O(n^2)$ و الگوریتم (۲) از مرتبه O(n) است. بنابراین، روش ارائه‌شده در این تحقیق از مرتبه $O(n^2)$ است. یادآوری می‌گردد که اعمال تعمیم‌های سه‌گانه فوق موجب تعمیم بیش‌ازحد در گرامر نمی‌شود. تعمیم بیش‌ازحد به این معنی است که در اثر آن هر جمله‌ای (مثلاً جمله‌های کاملاً تصادفی) را بتوان توسط گرامر تولید کرد. واضح است که گرامر استنتاج شده توسط روش ارائه‌شده چنین مشکلی را ندارد.

## ۳-۴- محاسبه برازندگی

با توجه به این‌که در این تحقیق از تکامل گرامری استفاده می‌شود لازم است مانند هر الگوریتم تکاملی، یک تابع یا روند محاسبه برازندگی ارائه گردد. این بخش به معرفی تابع برازندگی می‌پردازد. طبیعی است که در آسیب‌پذیری تزریق اسکریپت، مقدار برازندگی را باید با میزان موفقیت تزریق تعیین نمود. وقتی یک بردار حمله به ورودی برنامه تزریق می‌شود در قدم اول باید بتوان تشخیص داد که آیا موفقیتی حاصل شده است یا خیر؟

به‌طور مثال آیا بردار تزریق‌شده کاملاً در خروجی برنامه وب منعکس شده یا بخشی از آن منعکس شده است یا خیر؛ و اینکه آیا این انعکاس به معنی موفقیت حمله هست یا خیر و چه میزان موفقیت می‌توان به آن نسبت داد. ملاک تشخیص موفقیت حمله، اوراکل[1] نامیده می‌شود. انتخاب اوراکل از اهمیت ویژه‌ای برخوردار است و تصمیم‌گیری درباره محاسبه برازندگی نیز به انتخاب اوراکل بستگی دارد. فلذا در اولین قدم به بیان اوراکل انتخابی این تحقیق پرداخته می‌شود.

### ۳-۴-۱- انتخاب اوراکل

ازنظر این تحقیق در یک حمله تزریق اسکریپت، صرف مشاهده انعکاس دلیل بر موفقیت حمله نیست بلکه اجرای صحیح اسکریپت ملاک موفقیت است. در این تحقیق دو معیار اصلی برای ارزیابی موفقیت تزریق اسکریپت معرفی و استفاده می‌شود و برای محاسبه آن‌ها دو معیار کمکی دیگر نیز معرفی و مورداستفاده واقع می‌گردند که هرکدام به‌طور جداگانه توضیح داده می‌شوند:

۱- تغییر در ساختار صفحه
۲- تغییر در ساختار محاسباتی
۳- نتیجه پارسر جاوا اسکریپت
۴- انعکاس ارکان بردار حمله هدف

تغییر در ساختار صفحه: اوانسینی و سکاتو [۳۶] در مقاله باارزشی که ارائه کرده‌اند بیان می‌کنند که یک تزریق اسکریپت موفق باید ساختار صفحه HTML را که با DOM تعریف می‌شود تغییر دهد. به‌عنوان‌مثال، شکل (۳) مثالی از خروجی یک صفحه وب را نشان می‌دهد که عبارت PAGE از ورودی گرفته شده و در صفحه قرار داده شده است.

ساختار استخراجی از این صفحه مطابق شکل (۴) خواهد بود. اکنون اگر بجای عبارت PAGE بردار حمله (آ) تزریق شود:

```
TEST' "><script>alert(1)</script><tr
border=" '                                    (آ)
```

خروجی صفحه وب مطابق شکل (۵) و ساختار استخراجی آن مطابق شکل (۶) خواهد شد. ملاحظه می‌شود که در این شکل در خط ۱۱ یک تگ جدید <script> و پس‌ازآن یک تگ جدید <tr> اضافه شده است و به‌این‌ترتیب ساختار صفحه پس از تزریق عوض شده است.

ازنظر این تحقیق، اوراکل انتخابی اوانسینی و سکاتو [۳۶] مهم و باارزش است ولی کامل نیست. به‌عبارت‌دیگر، در اغلب مواردی که آسیب‌پذیری تزریق اسکریپت در یک برنامه وجود دارد، می‌توان به نحوی تزریق اسکریپت را انجام داد که در ساختار صفحه خروجی تغییر ایجاد کند، یعنی می‌توان یک بردار حمله ورودی پیدا کرد که این کار را انجام دهد.

```
<html dir="ltr" lang="en">
<head>
  <title>Administration Tool</title>
  <script type="text/javascript" src="script1.js"></script>
  <script type="text/javascript" src="script2.js"></script>
</head>
<body>
  <div id="contentText">
    <table border="0" width="100%">
      <tr id="default" class="data"
onclick="document.location.href='/index.php?page=PAGE&action
=preview' ">
    </table>
  </div>
</body>
</html>
```

**شکل (۳):** مثالی از خروجی یک صفحه وب.

```
1    <html>
2    <head>
3      <title></title>
4      <script></script>
5      <script></script>
6    </head>
7    <body>
8      <div>
9        <table>
10         <tr>
11       </table>
12     </div>
13   </body>
14   </html>
```

**شکل (۴):** ساختار صفحه وب.

[1] Oracle

این امر به روش این تحقیق که نوعی جستجو در بین بردارهای حمله است خیلی کمک می‌کند و چون یک اوراکل مشخص و واضح ارائه می‌کند پیاده‌سازی را راحت‌تر می‌نماید. ولی موارد دیگری نیز وجود دارد که آسیب‌پذیری تزریق اسکریپت در برنامه وجود دارد اما به‌هیچ‌وجه نمی‌توان از طریق تزریق بردار حمله، در ساختار خروجی تغییر ایجاد کرد. چنین مواردی عمدتاً به معنای آن است که یک سازوکار فیلتر کننده یا پاک‌کننده در ورودی برنامه وجود دارد و از تغییر ساختار برنامه جلوگیری می‌کند. به‌طور طبیعی، مطلوب این تحقیق نیست که از کشف و بهره‌برداری از آسیب‌پذیری‌ها در این‌گونه موارد صرف‌نظر کند.

```
<html dir="ltr" lang="en">
<head>
  <title>Administration Tool</title>
  <script type="text/javascript" src="script1.js"></script>
  <script type="text/javascript" src="script2.js"></script>
</head>
<body>
  <div id="contentText">
    <table border="0" width="100%">
      <tr id="default" class="data" onclick="document.location.
href='/index.php?page=TEST' "><script>alert(1)</script><tr
border='' '&action=preview' ">
    </table>
  </div>
</body>
</html>
```

**شکل (۵):** خروجی صفحه وب پس از تزریق بردار حمله (آ).

```
1    <html>
2    <head>
3      <title></title>
4      <script></script>
5      <script></script>
6    </head>
7    <body>
8      <div>
9        <table>
10         <tr>
11         <script></script>
12         <tr>
13       </table>
14     </div>
15   </body>
16   </html>
```

**شکل (۶):** ساختار صفحه وب پس از تزریق بردار حمله.

به‌عنوان مثال، شکل (۷) حالتی را نشان می‌دهد که در ورودی برنامه وب شکل (۳) یک فیلتر وجود دارد که حروف گیومه ' و " در ورودی تزریق‌شده (آ) را پیش از درج در متن صفحه به \' و \" تغییر می‌دهد. همان‌طور که مشخص است، تگ <script> در داخل صفت href محصور است و به‌عنوان یک تگ مستقل تفسیر نمی‌شود و ساختار صفحه را تغییر نمی‌دهد. درواقع باوجود چنین فیلتری، به‌هیچ‌وجه نمی‌توان در ساختار این صفحه تغییر ایجاد کرد. در این حالت اوراکل پیشنهادی آوانسینی و سکاتو [۳۶] قادر نیست هیچ بهره‌برداری از این آسیب‌پذیری را تأیید کند.

```
…
<tr id="default" class="data" onclick="document.location.
href='/index. php?page=TEST\'
\&quot;><script>alert(1)</script><tr border=\&quot;
\'&action=preview' ">
…
```

**شکل (۷):** خروجی صفحه وب پس از تزریق بردار حمله (آ) با حضور فیلتر.

به دلایل ذکرشده فوق، این تحقیق در انتخاب اوراکل از ایده اوانسینی و سکاتو [۳۶] استفاده می‌کند ولی به این انتخاب محدود نمی‌شود. البته لازم به ذکر است که تفاوت مهم این تحقیق نسبت به کار اوانسینی و سکاتو این است که آن‌ها درنهایت صرفاً در مورد وجود یا عدم وجود یک حمله تزریق اسکریپت تصمیم‌گیری می‌کنند درحالی‌که در این تحقیق لازم است میزان نزدیک بودن یا دور بودن از یک حمله تزریق اسکریپت اندازه‌گیری شود لذا یک معیار اندازه‌گیری[1] برای محاسبه «اندازه و مقدار موفقیت حمله» می‌خواهد که از روی آن مقدار برازندگی را حساب کند.

***تغییر در ساختار محاسباتی:*** در مثال شکل (۷) بهره‌برداری از آسیب‌پذیری موجود غیرممکن نیست. در این مورد، اگر بردار حمله (ب) به برنامه تزریق شود:

%26%2339%3B-alert(1)-%26%2339%3B (ب)

در این صورت خروجی برنامه مطابق شکل (۸) خواهد بود. حال اگر روی عبارت مربوطه در صفحه کلیک شود و تابع onclick اجرا گردد، عبارت « ' » ابتدا توسط مفسر HTML به گیومه(') تبدیل می‌شود (شکل (۹)) و سپس عبارت A=x-y-z برای اجرا به مفسر جاوا اسکریپت سپرده می‌شود که در آن، متغیرها به‌قرار زیر هستند و بنابراین، دستور alert(1) در هنگام این محاسبه با موفقیت اجرا خواهد شد:

```
A : document.location.href
'=x : '/index. php?page
(y : alert(1
'z : '&action=preview
```

در این حالت در ساختار صفحه هیچ تغییر داده نشده است اما علت موفقیت تزریق اسکریپت، تغییر در ساختار محاسباتی است که به‌صورت درج عبارت -alert(1)– در یک عبارت تخصیص دهی نمود پیدا کرده است. بنابراین، توانایی بردار حمله در ایجاد تغییر در ساختار محاسباتی یکی از ملاک‌های این تحقیق در

[1] Measure

ارزیابی موفقیت تزریق اسکریپت است.

تغییر در ساختار محاسباتی در این تحقیق، به معنای قرار گرفتن بخشی از بردار حمله تزریق‌شده در داخل یک محاسبه و تشکیل مرز محاسباتی در سمت چپ و سمت راست خود است. به‌عنوان‌مثال در شکل (۹)، بردار حمله توانسته است با قرار دادن یک گیومه و یک علامت منفی، در سمت چپ خود یک عبارت صحیح و یک مرز درست ایجاد کند. همچنین این بردار توانسته است با یک علامت منفی و یک گیومه، در سمت راست خود نیز یک مرز صحیح ایجاد کند و عبارت محاسباتی را معنی‌دار نماید. این مثال یک حالت کامل از مرزشکنی در سمت چپ و راست را نمایش می‌دهد. اما همواره موفقیت به این شکل کامل نیست و ممکن است بعضی بردارهای حمله بتوانند در سمت چپ مرز صحیح ایجاد کنند ولی قادر نباشند مرز سمت راست را نیز ایجاد نمایند. حالت‌های دیگری نیز ممکن است رخ دهد مثلاً ممکن است مرز سمت راست هم درست ایجاد شود ولی عبارت نهایی کامل نباشد. همه این حالت‌ها در محاسبه برازندگی تأثیر دارد و در این تحقیق در نظر گرفته شده است.

```
…
<tr id="default" class="data" onclick="document.location.
href='/index. php?page=&#39;-alert(1)-&#39;&action=preview'
">
…
```

**شکل (۸):** خروجی صفحه پس از تزریق بردار حمله (ب) با حضور فیلتر

```
…
… onclick="document.location. href='/index. php?page='-
alert(1)- '&action=preview' "…
…
```

**شکل (۹):** خروجی شکل (۸) ازنظر مفسر جاوا اسکریپت پس از کلیک

***نتیجه پارسر جاوا اسکریپت:*** همان‌طور که ذکر گردید، ازنظر این تحقیق اجرای صحیح اسکریپت ملاک موفقیت است. تست کامل این موضوع مستلزم اجرای برنامه وب در مرورگر واقعی و مشاهده رفتار نهایی است. هرچند این کار قابل انجام است ولی مستلزم صرف زمان و هزینه زیاد است. بجای این کار و به‌منظور افزایش سرعت و صرفه‌جویی، خروجی تزریق بردار حمله در این تحقیق به پارسر جاوا اسکریپت سپرده می‌شود و نتیجه‌ای که پارسر تولید می‌کند ملاک ارزیابی تزریق قرار می‌گیرد. در نتیجه خروجی پارسر جاوا اسکریپت دو حالت متصور است:

الف) هیچ خطایی توسط پارسر گزارش نمی‌شود

ب) خطایی گزارش شده و محل خطا تعیین می‌گردد. اگر پارسر هیچ خطایی گزارش نکند، این موضوع دو معنی می‌تواند داشته باشد:

۱- تزریق کاملاً موفق و بدون خطا بوده است،

۲- یا اینکه اصلاً موفقیتی نداشته و در ساختار محاسباتی اولیه(که آن‌هم درست و بدون خطا فرض می‌شود) هیچ تغییری نتوانسته ایجاد کند.

حالت ۱ مستحق اعطای حداکثر امتیاز بخش پارسر است و حالت ۲ شایسته هیچ پاداشی نیست. اما اگر خطا توسط پارسر گزارش شود نشان‌دهنده یک موفقیت نسبی است زیرا محاسبه اولیه خطایی نداشته و بروز خطا مؤید دست‌کاری در ساختار محاسبات یا به‌اصطلاح، مرزشکنی در محاسبات است هرچند به‌طور ناقص و با خطا انجام شده است. لذا این امر مستحق پاداش است و میزان این پاداش بستگی به حالت‌های مختلفی دارد:

۱- موفقیت در ایجاد مرز در سمت چپ

۲- موفقیت در عبور از مرز چپ

۳- موفقیت در ایجاد مرز سمت راست

۴- موفقیت در عبور از مرز سمت راست

در مورد آخر هرچه محل خطا فاصله بیشتری از مرز سمت راست داشته باشد پاداش باید بیشتر باشد. این تصمیم برای این است که الگوریتم تکاملی را به سمتی هدایت کند که محل خطا دور و دورتر از مرزها برود تا اینکه خطا کاملاً محو شود.

***انعکاس ارکان بردار حمله هدف:*** به‌طور عمومی نمی‌توان اجرای صحیح اسکریپت را قضاوت کرد چون هر بردار حمله، اسکریپت خاص خودش را دارد که به نیت حمله‌کننده برمی‌گردد. در این تحقیق فرض بر این است که یک «بردار حمله هدف» از قبل وجود دارد و فازر در جستجوی روشی برای تزریق این بردار حمله هدف به برنامه وب است. هر اسکریپت از چند رکن تشکیل شده است که اجزاء اصلی آن اسکریپت هستند. روش پیشنهادی این تحقیق این است که ارکان اصلی اسکریپت «بردار حمله هدف» را مشخص کند و سپس در خروجی به دنبال ظاهر کردن آن ارکان باشد به‌طوری‌که خروجی بیشترین شباهت را با آنچه مطلوب است داشته باشد.

به‌طور مثال اگر بردار حمله هدف به‌صورت عبارت (ج) باشد، ارکان آن به شرح زیر خواهند بود:

(ج) <script>alert(1)</ script>

۱- یک تگ script

۲- یک تابع alert

۳- پارامتر تابع alert که عبارت ۱ است.

به هر میزان که ارکان بردار هدف در خروجی بردارهای حمله مشاهده شوند به آن بردار حمله امتیاز متناسبی اعطا می‌شود.

بهترین بردار حمله در این معیار، برداری است که بتواند تمامی ارکان بردار حمله هدف را در خروجی ظاهر کند.

### ۳-۴-۲- تابع یا روال محاسبه برازندگی

مطالبی که در بخش اوراکل بیان شد مبنای محاسبه برازندگی قرار می‌گیرد. هرچند توضیح کامل مراحل محاسبه برازندگی فضای زیادی می‌طلبد که در اینجا گنجایش آن وجود ندارد ولی تلاش می‌شود این محاسبات به‌طور مجمل بیان شود. لازم به ذکر است اعداد و ضرایب استفاده‌شده در این محاسبات مانند ۱۰۰۰، ۱۰۰، ۱۰، ۳۰ بر اساس منطقی هستند که به‌طور ابتکاری[1] و بر اساس تجربه و همچنین سعی و خطا توسط نویسندگان مقاله پیشنهاد شده‌اند.

**محاسبه برازندگی بردار حمله با معیار تغییر در ساختار صفحه:** مراحل و محاسبات پیشنهادی این تحقیق جهت محاسبه برازندگی بردار حمله با معیار تغییر در ساختار صفحه وب، با کمک ارزیابی پارسر جاوا اسکریپت و میزان انعکاس ارکان بردار هدف، با روابط ۱ تا ۲۳ بیان شده است. در این روابط، $v$ «بردار حمله مورد آزمایش»، $\tau$ «بردار حمله هدف»، $\delta$ اختلاف خروجی برنامه وب در قبل و بعد از تزریق، و $l_x$ اندازه طول بردار یا عبارت $x$ هستند.

فرآیند محاسبه برازندگی با تزریق بردار حمله به برنامه وب آغاز می‌گردد. ابتدا ساختارهای صفحه وب قبل از تزریق و بعد از تزریق بردار حمله باهم مقایسه می‌شوند و اختلاف آن‌ها محاسبه شده و در $\delta$ نگهداری می‌شود. مبنای محاسبه اختلاف این دو ساختار، استفاده از الگوریتم هانت و زیمانسکی است[2] [۳۷]. چنانچه تغییری در ساختار صفحه ایجاد شده باشد باید یک یا چند تگ جدید به ساختار اضافه یا کم شده باشد. در محاسبه برازندگی با این معیار، تنها تگ‌هایی که اضافه می‌شوند، باارزش هستند.

ارزش‌گذاری این تگ‌ها در سه سطح عالی، خوب و معمولی انجام می‌پذیرد. نمونه‌هایی از این تگ‌ها در مجموعه‌های $\mathcal{B}$، $\mathcal{G}$ و $\mathcal{N}$ به ترتیب در روابط ۱، ۲ و ۳ ملاحظه می‌شود. توابع $b(\delta)$، $g(\delta)$، و $n(\delta)$ در روابط ۴، ۵ و ۶ امتیاز مربوط به تگ‌ها را محاسبه نموده و حاصل جمع آن‌ها در رابطه ۷ در $S_\delta$ قرار می‌گیرد. به‌طور مثال اگر یک تگ script اضافه شده باشد ۱۰۰۰ امتیاز تعلق می‌گیرد و سایر تگ‌ها امتیاز کمتری دارند. در این روابط برای تگ‌های سطح معمولی امتیاز ۱۰، برای سطح خوب ده برابر آن یعنی ۱۰۰ و برای عالی ده برابر سطح خوب یعنی ۱۰۰۰ در نظر گرفته شده است. این اختلاف برای این بوده که اولاً الگوریتم تکاملی به سمت سطح عالی هدایت شود و ثانیاً اختلاف‌ها کاملاً مشهود و قابل‌محاسبه باشد.

**ارزیابی انعکاس ارکان بردار حمله هدف:** رابطه ۸ نتیجه انعکاس بردار حمله $v$ (که ممکن است کاملاً مساوی خود $v$ باشد یا نباشد) به همراه تگی که در آن محصور است را در $v^t$ قرار می‌دهد و همین نتیجه ولی بدون تگ در رابطه ۹ در $v^{\bar{t}}$ ذخیره می‌گردد. با داشتن این مقادیر، یک شباهت سنجی بین بردار حمله $v$ و بردار هدف $\tau$ در رابطه ۱۰ محاسبه می‌شود که در محاسبه برازندگی استفاده می‌گردد.

در رابطه (۱۰) اندازه دو بردار و اختلاف اندازه آن‌ها تأثیر دارند. تابع $C_m$ دو بردار را باهم مقایسه کرده و تعداد حروفی که باهم یکسان هستند و در مکان یکسانی نیز قرار دارند را می‌شمارد. تابع $C_{dm}$ تعداد حروفی که در دو بردار متفاوت هستند را احصاء می‌کند. این تفاوت در ۳ ضرب می‌شود و به‌عنوان امتیاز منفی محسوب می‌گردد. همان‌طور که در رابطه (۱۱) مشاهده می‌گردد امتیاز تشابه دو بردار، مقداری بین صفر تا دو برابر طول بردار هدف خواهد بود.

$$\mathcal{B} = \{ \text{script} \} \quad (۱)$$

$$\mathcal{G} = \{ \text{div, img, form, iframe, input, button, …} \} \quad (۲)$$

$$\mathcal{N} = \{ \text{a, param, meta, …} \} \quad (۳)$$

$$b(\delta) = \begin{cases} 1000 & if\ \mathcal{B} \cap \delta \neq \emptyset \\ 0 & otherwise \end{cases} \quad (۴)$$

$$g(\delta) = \begin{cases} 100 & if\ b(\delta) = 0\ and\ \mathcal{G} \cap \delta \neq \emptyset \\ 0 & otherwise \end{cases} \quad (۵)$$

$$n(\delta) = \begin{cases} 10 & if\ g(\delta) = 0\ and\ b(\delta) = 0\ and\ \mathcal{N} \cap \delta \neq \emptyset \\ 0 & otherwise \end{cases} \quad (۶)$$

$$S_\delta = b(\delta) + g(\delta) + n(\delta) \quad (۷)$$

$$v^t = t(v) \quad (۸)$$

$$v^{\bar{t}} = \bar{t}(v) \quad (۹)$$

$$SM(v) = \max\left(0, l_\tau - \left|l_\tau - l_{v^{\bar{t}}}\right| + C_m\left(v^{\bar{t}}, \tau\right) - 3\left(C_{dm}\left(v^{\bar{t}}, \tau\right)\right)\right) \quad (۱۰)$$

$$0 \leq SM(v) \leq 2.l_\tau \quad (۱۱)$$

$$s_J = Jaccard\left(v^{\bar{t}}, \tau, 4\right) \qquad 0 \leq s_J \leq 1 \quad (۱۲)$$

$$s_P = Pars\left(v^{\bar{t}}\right) = \bigcup_i P_i^{v^{\bar{t}}} \quad (۱۳)$$

$$s_K = Tok(\tau) = \bigcup_i W_i^{\tau} \quad (۱۴)$$

$$s_{PK} = \frac{|s_P \cap s_K|}{|s_K|} \qquad 0 \leq s_{PK} \leq 1 \quad (۱۵)$$

$$X_t(v) = \begin{cases} 10.s_J + 100.s_{PK} + 300 & if\ s_{PK} > 0 \\ 10.s_J & if\ s_{PK} = 0 \end{cases} \quad (۱۶)$$

$$\hat{S}_{TX} = \begin{cases} SM(v) + X_t(v) - l_{v^t} & if\ it\ is \geq 0\ and\ SM(v) > 0 \\ X_t(v) & if\ previous < 0\ and\ SM(v) > 0 \\ 0 & if\ SM(v) = 0 \end{cases} \quad (۱۷)$$

[1] Heuristic

[2] این الگوریتم مشتقات متعددی دارد و ابزار معروف diff در لینوکس نیز با همین الگوریتم ایجاد شده است.

$$S_{TX} = \begin{cases} \hat{S}_{TX} & if\ b(\delta) > 0 \\ \frac{\hat{S}_{TX}}{10} & if\ g(\delta) > 0 \\ \frac{\hat{S}_{TX}}{100} & if\ n(\delta) > 0 \end{cases} \quad (۱۸)$$

$$\hat{S}_X = \max_T S_{TX} \quad (۱۹)$$

$$S_X = \begin{cases} 10.\hat{S}_X & if\ \hat{S}_X > 0 \\ \frac{S_\delta}{2} & if\ \hat{S}_X = 0 \end{cases} \quad (۲۰)$$

$$S_c(v) = S_\delta + S_X - L_\delta^- \quad (۲۱)$$

$$S_t(v) = S_c(v) - \min(l_v, \frac{S_c(v)}{2}) \quad (۲۲)$$

$$\phi_t = h_t(v) = \begin{cases} S_t(v) & if\ S_t(v) > 0 \\ 0 & if\ S_t(v) \le 0 \end{cases} \quad (۲۳)$$

همچنین فاصله جاکارد بین دو بردار $\tau$ و $v^{\bar{t}}$ نیز در بازه ۴ حرفی در رابطه (۱۲) محاسبه می‌گردد. این فاصله به همراه امتیاز شباهت فوق‌الذکر، در محاسبات بعدی نقش خواهند داشت.

**ارزیابی با استفاده از پارسر جاوا اسکریپت:** در رابطه (۱۳)، بردار $v^{\bar{t}}$ به پارسر جاوا اسکریپت تحویل داده می‌شود و مجموعه عناصری که توسط پارسر تشخیص داده می‌شود استخراج می‌گردد. از طرف دیگر تعداد توکن‌های قابل‌تشخیص در بردار هدف $\tau$ در رابطه (۱۴) محاسبه می‌گردد. اشتراک این دو مجموعه نشان می‌دهد که ازنظر پارسر جاوا اسکریپت، چه تعداد از عناصر بردار حمله که قابل پارس کردن هستند با بردار هدف، اشتراک دارند. این میزان به‌صورت نسبی در رابطه ۱۵ محاسبه می‌شود که مقداری بین صفر و یک است.

با در اختیار بودن این مقادیر، یک امتیاز خالص پارسر $X_t(v)$ برای بردار حمله در رابطه ۱۶ تخصیص داده می‌شود. ملاحظه می‌شود که اگر امتیاز پارسر مثبت باشد حداقل ۳۰۰ امتیاز در این مرحله تعلق می‌گیرد.

این امتیاز به همراه میزان شباهت محاسبه‌شده در قبل، منجر به یک امتیاز ترکیبی در رابطه (۱۷) می‌شود و اندازه عبارت $v^t$ نیز از حاصل آن کم خواهد شد (بین بردارهای حمله با قابلیت یکسان، بردارهای طولانی‌تر امتیاز کمتری می‌گیرند). درنهایت در رابطه (۱۸) بسته به اینکه تگ اضافه‌شده فعلی در کدام مجموعه قرار دارد یک ضریب به امتیاز اعمال می‌گردد.

ازآنجاکه ممکن است تگ‌های متعددی به ساختار اضافه شده باشند، روابط (۸ - ۱۸) برای تمامی تگ‌های اضافه‌شده محاسبه می‌شوند و بیشترین مقدار آن در رابطه (۱۹) انتخاب می‌شود. اگر پس از تمام این مراحل، این مقدار مثبت باشد، در رابطه (۲۰) به ده برابر افزایش خواهد یافت و در غیر این صورت فقط نیمی از $S_\delta$ به‌عنوان امتیاز محسوب خواهد شد.

در رابطه (۲۱)، امتیاز محاسبه‌شده تاکنون با خود $S_\delta$ جمع می‌شود. در اینجا تعداد تگ‌هایی که از ساختار صفحه حذف شده‌اند یعنی $L_\delta^-$، امتیاز منفی خود را اعمال می‌کنند. هدف از این کار این است که الگوریتم تکاملی به سمتی هدایت شود که تنها تغییرات مطلوب را به ساختار صفحه اضافه کند و لزوماً از ساختار آن چیزی کم نکند.

رابطه (۲۲) به‌صورت کنترل‌شده، اندازه بردار حمله را از امتیاز محاسبه‌شده آن کسر می‌کند تا بهای بیشتری به بردارهای حمله کوتاه‌تر داده شود. درنهایت در رابطه (۲۳) میزان برازندگی اختصاص‌یافته به بردار حمله تعیین می‌گردد و تضمین می‌کند که حداقل این مقدار برابر صفر خواهد بود.

***محاسبه برازندگی بردار حمله با معیار تغییر در ساختار محاسباتی:*** روابط (۲۴-۲۶) با کمک روابطی که قبلاً گفته شد کمک می‌کنند تا برازندگی بردار حمله با معیار تغییر در ساختار محاسباتی حساب شود. رابطه (۲۴) به حالت‌های مختلف مرزشکنی در سمت چپ و راست عبارت محاسباتی، امتیاز می‌دهد. $B_L$ مکان مرز چپ و $B_R$ مکان مرز راست در محل تزریق بردار حمله هستند و $B_E$ محلی است که پارسر جاوا اسکریپت خطا گزارش کرده است. هرچقدر از سمت چپ به راست پیش برود و خطا در نقطه دورتری رخ داده باشد امتیاز بیشتر خواهد بود.

اگر خطا وجود داشته باشد ولی امتیاز جاکارد $s_J$ صفر باشد بردار حمله مستحق مقدار کمی پاداش است. همچنین اگر خطا وجود داشته باشد ولی ازنظر پارسر، تشابهی بین بردار حمله و بردار هدف نباشد یعنی $s_{PK}$ صفر باشد بازهم مستحق مقداری پاداش خواهد بود که به برازندگی بردار حمله تعلق می‌گیرد. تمامی این پاداش‌ها به شرطی هستند که خطای گزارش‌شده توسط پارسر از جنس خطای نحوی[1] باشد و در این صورت مقدار آن ۱۰۰ برابر خواهد شد. اگر خطای دیگری رخ داده باشد معیار محاسبه امتیاز همان $X_t(v)$ خواهد بود که در رابطه ۱۶ محاسبه می‌شود که ده برابر آن لحاظ خواهد شد. این نکات در رابطه ۲۵ مشخص شده‌اند.

رابطه (۲۶) بیان می‌دارد که امتیاز تغییر محاسباتی تنها در صورتی وجود دارد که یک صفت جاوا اسکریپت (مانند onclick یا onmouseover) در عبارت وجود داشته باشد در غیر این صورت هیچ امتیاز محاسباتی لحاظ نخواهد شد. درنهایت، رابطه ۲۷ بین دو امتیاز تغییر در ساختار صفحه و امتیاز تغییر در ساختار محاسباتی، آن را که بزرگ‌تر است انتخاب می‌کند. البته استفاده از رابطه (۲۷) اختیاری است و می‌توان به‌جای آن، الگوریتم تکاملی را با هریک از این معیارها به‌صورت مستقل اجرا نمود و در

[1] Syntax Error

انتها بین پاسخ‌های نهایی، آن را که بهتر است انتخاب کرد. بر اساس روابط و ضرایبی که ارائه گردید، در تمامی آزمایش‌های انجام‌شده در صورت کشف آسیب‌پذیری و اتهام درست، مقدار محاسبه‌شده برازندگی بیش از ۴۰۰۰ بوده است و در سایر حالات مقدار بسیار کمتری داشته است فلذا مقدار آستانه کشف آسیب‌پذیری در تابع برازندگی ارائه‌شده، برابر ۴۰۰۰ تعیین می‌گردد.

(۲۴)

$$X_b(v)=\begin{cases}0 & if\ B_E<B_L\\ 1 & if\ B_E=B_L\\ 5 & otherwise\ if\ s_J=0\\ 10+5.\dfrac{B_E-B_L}{B_R-B_L} & o.w.\ if\ B_E<B_R\\ 16 & o.w.\ if\ B_E=B_R\\ 5 & o.w.\ if\ s_{PK}=0\\ 20+\dfrac{B_E-B_R}{10} & o.w.\ if\ \dfrac{B_E-B_R}{10}<2\\ 22 & otherwise\end{cases}$$

$$X_a(v)=\begin{cases}100.X_b(v)+100.S_J & if\ SE\\ 10.X_t(v) & if\ not\ SE\end{cases} \quad (۲۵)$$

$$\phi_a=h_a(v)=s_a(v)=\begin{cases}X_a(v) & if\ JS\\ 0 & if\ not\ JS\end{cases} \quad (۲۶)$$

$$\phi=h(v)=\begin{cases}S_t(v) & if\ S_t(v)>S_a(v)\\ S_a(v) & otherwise\end{cases} \quad (۲۷)$$

## ۴- پیاده‌سازی و ارزیابی نتایج

برای ارزیابی توانمندی روش ارائه‌شده این تحقیق در بهره‌برداری از آسیب‌پذیری، تعدادی برنامه وب انتخاب شدند و مورد آزمون قرار گرفتند. فهرست این برنامه‌ها و نتایج حاصله در راستای کشف و بهره‌برداری از آسیب‌پذیری در

جدول (۱) آمده است. به‌منظور مقایسه عملکرد، نتایج مشابه جستجو روی همین برنامه‌ها توسط دو ابزار دیگر انجام شده است. یکی ابزار جعبه سیاه مشهور و رایج ZAP [۳۸] (نسخه 2.8.0) و دیگری ابزار مدرن و جعبه سفید NAVEX [۱۴].

نتایج اجرای ابزار NAVEX روی برنامه‌های وب منتخب، در مقاله مربوطه ذکر شده است [۱۴]. به‌منظور اجرای ابزار ZAP بر روی این برنامه‌ها، هریک از آن‌ها در یک ماشین مجازی مستقل نصب شدند و ابزار ZAP روی تک‌تک آن‌ها اجرا گردید. جهت اجرای روش این تحقیق نیز از همین ماشین‌های مجازی استفاده گردید. به این منظور هریک از آسیب‌پذیری‌ها به‌طور جداگانه به‌منظور یافتن بهره‌برداری، مورد تکامل گرامری قرار گرفتند. تعداد اعضاء هر نسل در فرآیند تکامل برابر ۵۰ بردار حمله انتخاب شد و برای هر آسیب‌پذیری حداکثر تا ۲۰۰ نسل تولید گردید، هرچند در بسیاری موارد پاسخ نهایی قبل از رسیدن به آخرین نسل مشخص شده بود.

در جدول (۱) برای هریک از روش‌ها دو ستون وجود دارد که یکی تعداد بهره‌برداری موفق و دیگری تعداد اتهام غلط (FP) یا اعلام بهره‌برداری غلط (FE) را نمایش می‌دهد. منظور از بهره‌برداری غلط این است که انعکاس، یعنی امکان بالقوه تزریق، وجود داشته و بهره‌برداری ارائه شده است ولی بهره‌برداری ارائه‌شده درواقع کار نمی‌کند. ستون مکان بالقوه تزریق مجموع تعداد آسیب‌پذیری‌های تزریق اسکریپت انعکاسی را نشان می‌دهد که توسط منابع مختلف گزارش شده و در اینترنت قابل یافتن است.

**جدول (۱):** نتایج آزمون و مقایسه با سایر روش‌ها

| | برنامه وب تحت آزمون | تعداد مکان بالقوه تزریق | نتایج این تحقیق | | ZAP | | NAVEX | |
|---|---|---|---|---|---|---|---|---|
| | | | تعداد اکسپلویت موفق | FP/FE | تعداد اکسپلویت موفق | FP/FE | تعداد اکسپلویت موفق | FP/FE |
| ۱ | osCommerce-2.3.4 | ۶ | ۳ | ۰ | ۱ | ۳ | ۵ | ۰ |
| ۲ | myBloggie-2.1.3 | ۱۳ | ۱۱ | ۰ | ۱ | ۶ | ۲ | ۰ |
| ۳ | WeBid-1.2.1 | ۱۱ | ۱۱ | ۰ | ۹ | ۰ | ۸ | ۴ |
| ۴ | SchoolMate-1.5.4 | ۱۲ | ۱۲ | ۰ | ۱۲ | ۰ | ۰ | ۰ |
| ۵ | FaqForge-1.3.2 | ۱۶ | ۷ | ۰ | ۶ | ۱۰ | ۷ | ۰ |
| ۶ | WebChess-0.9 | ۱۰ | ۸ | ۰ | ۸ | ۲ | ۱۴ | ۰ |
| ۷ | CPG-1.5.46 | ۰ | ۰ | ۰ | ۰ | ۰ | ۰ | ۰ |
| ۸ | phpBB-2.0.23 | ۹ | ۰ | ۰ | ۰ | ۹ | ۲ | ۰ |
| ۹ | **جمع** | **۷۷** | **۵۲** | **۰** | **۳۷** | **۳۰** | **۳۸** | **۴** |

همان‌طور که از روی این جدول مشخص است، بیشترین تعداد بهره‌برداری موفق (۵۲ عدد) با استفاده از روش این تحقیق به‌دست آمده است. همچنین هیچ مورد اتهام غلط با روش این تحقیق ایجاد نشده است. مفهوم این موضوع این است که اختلاف بین تعداد بهره‌برداری با تعداد کل گزارش آسیب‌پذیری با روش این تحقیق، تبرئه صحیح[1] (TN) بوده است که البته در جدول قید نشده است. در برخی موارد تعداد بهره‌برداری گزارش‌شده توسط NAVEX از روش این تحقیق و حتی از تعداد آسیب‌پذیری گزارش‌شده در اینترنت بیشتر است(خط ۶ ستون اکسپلویت NAVEX). این امر به دلیل این است که این ابزار جعبه سفید عمل می‌کند و البته گزارش کامل نتایج خود را ارائه نکرده است. درمجموع با روش این تحقیق ۱۹٪ بهبود در تعداد اکسپلویت‌های کشف‌شده نسبت به هر دو ابزار جعبه سفید و جعبه سیاه به‌دست آمده است ( ۷۷/(۳۷-۵۲) ).

## ۵- بحث و مقایسه

با توجه به اینکه این تحقیق نحوه بهره‌برداری از آسیب‌پذیری تزریق اسکریپت را با استفاده از گرامر و تکامل گرامری بررسی می‌کند از زوایای متعدد با سایر مقالات در این زمینه قابل‌مقایسه است.

کار دوچن و همکارانش [۴] شباهت‌های زیادی با این تحقیق دارد و هردو با استفاده از گرامر و الگوریتم تکاملی در محل بالقوه تزریق به جستجوی بهره‌برداری از آسیب‌پذیری می‌پردازند. تفاوت در اینجا است که در آن مقاله، گرامر وظیفه اصلی تولید ورودی‌های آزمون را به عهده ندارد ولی در این تحقیق از تکامل گرامری استفاده شده و گرامر نقش محوری در آن دارد.

محمدی و همکارانش [۱۹] حمله تزریق اسکریپت را به یک اجرای تابع جاوا اسکریپت یا یک عملگر تخصیص محدود می‌کنند. گرامر پیشنهادی محمدی و همکارانش بازگشتی نیست و امکان تولید بردار حمله با ساختار متنوع از روی آن وجود ندارد. اما این تحقیق حمله تزریق اسکریپت را به موارد مذکور محدود نمی‌کند.

مقاله اوانسینی و سکاتو [۳۶] در ارائه یک اوراکل برای تشخیص موفقیت‌آمیز بودن تزریق اسکریپت با این تحقیق شباهت دارد. البته اوانسینی و سکاتو موفقیت تزریق را تنها در تغییر ساختار صفحه وب می‌بینند درحالی‌که در این تحقیق، اوراکل به این حالت محدود نمی‌شود. مهم‌تر اینکه همان‌گونه که قبلاً ذکر شد اوانسینی و سکاتو درنهایت تنها در مورد وجود یا عدم وجود یک حمله تزریق اسکریپت تصمیم‌گیری می‌کنند درحالی‌که این تحقیق به دنبال بهره‌برداری از آسیب‌پذیری نیز هست و به همین دلیل یک معیار اندازه‌گیری برای محاسبه اندازه و مقدار موفقیت حمله ارائه کرده است.

الهذلی و همکارانش [۱۴] با رویکرد جعبه سفید توانسته‌اند در مواردی بهره‌برداری از آسیب‌پذیری ارائه کنند که قبلاً توسط هیچ روش جعبه سفید و جعبه سیاه ارائه نشده بود. به‌عنوان نمونه در برنامه osCommerce-2.3.4 چنین آسیب‌پذیری کشف و بهره‌برداری کرده‌اند. اما روش این تحقیق با رویکرد جعبه سیاه قادر بوده است همان بهره‌برداری را کشف نماید.

## ۶- نتیجه‌گیری

در این مقاله برای اولین بار ایده استفاده از تکامل گرامری به‌منظور بهره‌برداری از آسیب‌پذیری تزریق اسکریپت مطرح گردید. یک روش استنتاج گرامر از روی بردارهای حمله ارائه شد و با استفاده از این روش، یک گرامر مخصوص تولید بردارهای حمله جهت استفاده در فازرهای تزریق اسکریپت، تولید و ارائه گردید. چهار معیار برای ارزیابی بردارهای حمله معرفی شد. تابع برازندگی برای سنجش بردارهای حمله ارائه و شرح داده شد.

نتیجه این تحقیق نشان می‌دهد که اولاً بهره‌برداری خودکار از آسیب‌پذیری، از طریق آزمون فاز و تولید داده ورودی آزمون امکان‌پذیر است و ثانیاً تکامل گرامری می‌تواند با در اختیار داشتن یک گرامر مناسب، این فرآیند آزمون را پیاده‌سازی نموده و امکان بهره‌برداری را محقق نماید. همچنین در بعضی موارد بهره‌برداری از آسیب‌پذیری توسط فازرهای جعبه سیاه میسر نبوده و تنها با بعضی رویکردهای جعبه سفید امکان داشته است ولی به کمک روش این تحقیق با رویکرد جعبه سیاه نیز میسر شده است.

کارهای زیادی در ادامه این تحقیق می‌تواند انجام شود. اولاً در مورد نحوه محاسبه برازندگی با صرف زمان و هزینه کمتر جای تحقیق وجود دارد. ثانیاً می‌توان روش‌های زمان‌بندی و روش‌های احتمالاتی را با این روش ترکیب کرد تا بتوان بجای یکی، هم‌زمان تعداد زیادی بردار حمله تزریق اسکریپت و بردار هدف مورد جستجو قرار بگیرند. ثالثاً، به‌منظور تمرکز روی خود رویکرد انتخابی، در این تحقیق تنها به بررسی بهره‌برداری از تزریق اسکریپت از نوع انعکاسی پرداخته شده است؛ ولی مباحث ارائه‌شده می‌تواند با تغییرات مناسب در مورد سایر انواع تزریق اسکریپت نیز به‌کار رود. رابعاً روش پیشنهادی این تحقیق را می‌توان در مورد سایر آسیب‌پذیری‌ها نیز به‌کار برد.

[1] True Negative

## ۷- مراجع


[1] OWASP-2017-Top-10, 2017. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf

[2] A. Avancini and M. Ceccato, "Comparison and integration of genetic algorithms and dynamic symbolic execution for security testing of cross-site scripting vulnerabilities," Information and Software Technology, vol. 55, no. 12, pp. 2209-2222, 2013.

[3] X. Guo, S. Jin, and Y. Zhang, "XSS Vulnerability Detection Using Optimized Attack Vector Repertory," in 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 29-36, 2015.

[4] F. Duchene, R. Groz, S. Rawat, and J.-L. Richier, "XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing," in SECTEST 2012 - 3rd International Workshop on Security Testing (affiliated with ICST), Montreal, Canada, pp. 815-817, 2012.

[5] J. Yang and Q. Tang, "RTF Editor XSS Fuzz Framework," in International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Springer, Cham, pp. 941-951, 2017.

[6] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in Proceedings of the 13th International Conference on Availability, Reliability and Security, p. 6, 2018.

[7] M. E. Ruse and S. Basu, "Detecting cross-site scripting vulnerability using concolic testing," in Information Technology: New Generations (ITNG), 2013 Tenth International Conference on, pp. 633-638, 2013.

[8] M. A. Ahmed and F. Ali, "Multiple-path testing for cross site scripting using genetic algorithms," Journal of Systems Architecture, vol. 64, pp. 50-62, 2016.

[9] A. W. Marashdih, Z. F. Zaaba, and H. K. Omer, "Web Security: Detection of Cross Site Scripting in PHP Web Application using Genetic Algorithm," International Journal of Advanced Computer Science and Applications, vol. 8, no. 5, pp. 64-75, 2017.

[10] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting," Journal of Parallel and Distributed Computing, 2017.

[11] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in Proceedings of the 15th international conference on world wide web, pp. 247-256, 2006.

[12] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, pp. 199-209, 2009.

[13] H. Homaei and H. R. Shahriari, "Athena: A framework to automatically generate security test oracle via extracting policies from source code and intended software behaviour," Information and Software Technology, vol. 107, pp. 112-124, 2019.

[14] A. Alhuzali, R. Gjomemo, B. Eshete, and V. Venkatakrishnan, "NAVEX: Precise and Scalable Exploit Generation for Dynamic Web Applications," in 27th USENIX Security Symposium (USENIX Security 18), pp. 377-392, 2018.

[15] H.-Y. Shih, H.-L. Lu, C.-C. Yeh, H.-C. Hsiao, and S.-K. Huang, "A Generic Web Application Testing and Attack Data Generation Method," in International Conference on Security with Intelligent Computing and Big-data Services, pp. 232-247, 2017.

[16] A. Alhuzali, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Chainsaw: Chained automated workflow-based exploit generation," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 641-652, 2016.

[17] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, "Steelix: program-state based binary fuzzing," presented at the Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 2017.

[18] H. C. Huang, Z. K. Zhang, H. W. Cheng, and S. W. Shieh, "Web Application Security: Threats, Countermeasures, and Pitfalls," Computer, vol. 50, no. 6, pp. 81-85, 2017.

[19] M. Mohammadi, B. Chu, and H. R. Lipford, "Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing," in 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 364-373, 2017.

[20] M. Mohammadi, B. Chu, H. R. Lipford, and E. Murphy-Hill, "Automatic Web Security Unit Testing: XSS Vulnerability Detection," in 2016 IEEE/ACM 11th International Workshop in Automation of Software Test (AST), pp. 78-84, 2016.

[21] T. Zhushou, Z. Haojin, C. Zhenfu, and Z. Shuai, "L-WMxD: Lexical based Webmail XSS Discoverer," in 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 976-981, 2011.

[22] Y. H. Wang, C. H. Mao, and H. M. Lee, "Structural learning of attack vectors for generating mutated xss attacks," arXiv preprint arXiv:1009.3711, 2010.

[23] V. Atlidakis, R. Geambasu, P. Godefroid, M. Polishchuk, and B. Ray, "Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations," arXiv preprint arXiv:2005.11498, 2020.

[24] M. Eberlein, Y. Noller, T. Vogel, and L. Grunske, "Evolutionary Grammar-Based Fuzzing," in International Symposium on Search Based Software Engineering, pp. 105-120, 2020.

[25] M. Z. Nasrabadi and S. Parsa, "Automatic Test Data Generation in File Format Fuzzers," Journal of Electronical and Cyber Defence, vol. 8, no. 1, 2020.

[26] O. Caño Bellatriu, "Penetration testing automation system," Barcelona School of Informatic, 2014.

[27] C. Ryan, M. O'Neill, and J. Collins, "Grammatical evolution: Solving trigonometric identities," in proceedings of Mendel, p. 4th, 1998.

[28] M. O'Neill and C. Ryan, "Grammatical evolution," Trans. Evol. Comp, vol. 5, no. 4, pp. 349-358, 2001.

[29] C. Ryan, "Grammatical evolution tutorial," presented at the Proceedings of the 12th annual conference companion on Genetic and evolutionary computation, Portland, Oregon, USA, 2010.

[30] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, and M. O'Neill, "PonyGE2: Grammatical evolution in python," in Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1194-1201, 2017.

[31] A. Sołtysik-Piorunkiewicz and M. Krysiak, "The Cyber Threats Analysis for Web Applications Security in Industry

4.0," in Towards Industry 4.0—Current Challenges in Information Systems, ed: Springer, pp. 127-141, 2020.

[32] N. H. Nguyen, V. H. Le, V. O. Phung, and P. H. Du, "Toward a Deep Learning Approach for Detecting PHP Webshell," in Proceedings of the Tenth International Symposium on Information and Communication Technology, pp. 514-521, 2019.

[33] C. Lv, L. Zhang, F. Zeng, and J. Zhang, "Adaptive Random Testing for XSS Vulnerability," in 2019 26th Asia-Pacific Software Engineering Conference (APSEC), pp. 63-69, 2019.

[34] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," Journal of Artificial Intelligence Research, vol. 7, pp. 67-82, 1997.

[35] C. D. L. Higuera, "Grammatical Inference Learning Automata and Grammars," Cambridge University Press, 2010.

[36] A. Avancini and M. Ceccato, "Circe: A grammar-based oracle for testing cross-site scripting in web applications," in 2013 20th Working Conference on Reverse Engineering (WCRE), pp. 262-271, 2013.

[37] J. Hunt and T. Szymanski, "A fast algorithm for computing longest common subsequences," Communications of the ACM, vol. 20, no. 5, pp. 350-353, 1977.

[38] S. Bennetts, "Owasp zed attack proxy," App. Sec. USA, 2013.

## ۸- پیوست

در این پیوست نحوه استنتاج گرامر موردنیاز این تحقیق با ذکر مثال توضیح داده می‌شود.

***استنتاج گرامر اولیه:*** به‌عنوان مثالی از نحوه کار این الگوریتم، جدول (۲) را در نظر بگیرید که ۷ بردار حمله خوش‌فرم تزریق اسکریپت را نشان می‌دهد. نتیجه اجرای الگوریتم سکوئیتور تغییریافته با ورودی جدول (۲) در شکل (۱۰) مشاهده می‌شود.

***تعمیم‌های سه‌گانه:*** نتیجه اجرای الگوریتم (۱) به‌صورت شکل (۱۱) خواهد بود. نتیجه نهایی اعمال عملگرهای تعمیم پرشی کردن، جهشی کردن و توقف‌پذیری گرامر در شکل (۱۲) آمده است. همان‌طور که در شکل (۱۲) ملاحظه می‌شود، در مرحله پایانی چند تغییر دیگر اعمال شده است ازجمله این‌که به‌جای ( ' ) و ( " ) از قواعد تولید <RSQ> و <RDQ> استفاده شده است که در <RBORDER> آمده‌اند.

**جدول (۲):** فهرستی از چند بردار حمله خوش‌فرم تزریق اسکریپت.

| شماره | بردار حمله |
|---|---|
| 1: | <SCRIPT>alert('XSS');</SCRIPT> |
| 2: | <SCRIPT SRC=http://xss.rocks/xss.js></SCRIPT> |
| 3: | <IMG SRC="javascript:alert('XSS');"> |
| 4: | <SCRIPT SRC=http://xss.rocks/xss.js?<B> |
| 5: | <IMG SRC="javascript:alert('XSS')" |
| 6: | <SCRIPT SRC="http://xss.rocks/xss.jpg"></SCRIPT> |
| 7: | <SCRIPT a=">" SRC="http://xss.rocks/xss.js"></SCRIPT> |

```
<R0> ::=
"<"<R1><R2><R3>|<R4>">"<R3>|<R5><R2><R6>|<R4>"?<b>"|<R5><R7>""|<R8>""<R9>"jpg"<R10>
                |<R11><RSPACE>"a="<R6>""<R12><R13><R10>
<R1> ::= "script>"
<R2> ::= <R7>";"
<R3> ::= "</"<R1>
<R4> ::= <R8><R13>
<R5> ::= "<img"<R12>"javascript:"
<R6> ::= """>"
<R7> ::= "alert('xss')"
<R8> ::= <R11><R14>
<R9> ::= "http:/"<R15>"rocks"<R15>
<R10> ::= <R6><R3>
<R11> ::= "<script"
<R12> ::= <R14>""
<R13> ::= <R9>"js"
<R14> ::= <RSPACE>"src="
<R15> ::= "/xss."
<RSPACE> ::= " "
<RTAB> ::= "\t"
```

**شکل (۱۰):** خروجی الگوریتم سکوئیتور اختصاصی شده با ورودی جدول (۲).

```
<R0> ::=
"<"<R1><R2><R3>|<R4>">"<R3>|<R5><R2><R6>|<R4>"?<b>"|<R5><R7>""|<R8>""<R9>"jpg"<R10>
        |<R11><RSPACE>"a="<R6>""<R12><R13><R10>
<R1> ::= "script>" | "script>"<R2><R3>
<R2> ::= <R7>";" | <R7>";"<R3> | <R7>";"<R6>
<R3> ::= "</"<R1>
<R4> ::= <R8><R13> | <R8><R13>">"<R3> | <R8><R13>"?<b>"
<R5> ::= "<img"<R12>"javascript:" | "<img"<R12>"javascript:"<R2><R6> |
"<img"<R12>"javascript:"<R7>""
<R6> ::= """>" | """>"""<R12><R13><R10> | """>"<R3>
<R7> ::= "alert('xss')" | "alert('xss')""" | "alert('xss');"
<R8> ::= <R11><R14> | <R11><R14>""<R9>"jpg"<R10> | <R11><R14><R13>
<R9> ::= "http:/"<R15>"rocks"<R15> | "http:/"<R15>"rocks"<R15>"jpg"<R10> |
"http:/"<R15>"rocks"<R15>"js"
<R10> ::= <R6><R3>
<R11> ::= "<script" | "<script"<RSPACE>"a="<R6>""<R12><R13><R10> | "<script"<R14>
<R12> ::= <R14>"" | <R14>""<R13><R10> | <R14>"""javascript:"
<R13> ::= <R9>"js" | <R9>"js"<R10>
<R14> ::= <RSPACE>"src=" | <RSPACE>"src="""
<R15> ::= "/xss." | "/xss.rocks"<R15>
<RSPACE> ::= " "
<RTAB> ::= "\t"
```

**شکل (۱۱):** خروجی الگوریتم (۱) با ورودی شکل (۱۰).

```
<R0> ::=
"<"<M1><M2><M3>|<M4>">"<M3>|<M5><M2><M6>|<M4>"?<b>"|<M5><M7><RDQ>

|<M8><RDQ><M9>"jpg"<M10>|<M11><RSPACE>"a="<M6><RDQ><M12><M13><M10>
<R1> ::= "script>" | "script>"<R2><R3> | <RFINISH>
<R2> ::= <R7>";" | <R7>";"<R3> | <R7>";"<R6> | <RFINISH>
<R3> ::= "</"<R1> | <RFINISH>
<R4> ::= <R8><R13> | <R8><R13>">"<R3> | <R8><R13>"?<b>" | <RFINISH>
<R5> ::= "<img"<R12>"javascript:" | "<img"<R12>"javascript:"<R2><R6> |
"<img"<R12>"javascript:"<R7><RDQ> | <RFINISH>
<R6> ::= <RDQ>">" | <RDQ>">"<RDQ><R12><R13><R10> | <RDQ>">"<R3> |
<RFINISH>
<R7> ::= "alert('xss')" | "alert('xss')"<RDQ> | "alert('xss');" | <RFINISH>
<R8> ::= <R11><R14> | <R11><R14><RDQ><R9>"jpg"<R10> | <R11><R14><R13> |
<RFINISH>
……..
……..
<R14> ::= <RSPACE>"src=" | <RSPACE>"src="<RDQ> | <RFINISH>
<R15> ::= "/xss." | "/xss.rocks"<R15> | <RFINISH>
<M1> ::= <RBORDER><R1><RBORDER>
<M2> ::= <RBORDER><R2><RBORDER>
……..
……..
<M13> ::= <RBORDER><R13><RBORDER>
<M14> ::= <RBORDER><R14><RBORDER>
<M15> ::= <RBORDER><R15><RBORDER>
<RSPACE> ::= " "
<RTAB> ::= "\t"
<RDQ>  ::= "" | "&#34;" | "%22" | "%26%2334%3B"
<RSQ>  ::= "" | "&#39;" | "%27" | "%26%2339%3B"
<RBORDER>  ::= "" | <RDQ> | <RSQ>
<RFINISH> ::= "" | "" | "" | ";" | "%00"
```

**شکل (۱۲):** خروجی نهایی الگوریتم‌های (۱) و (۲) با ورودی جدول (۲) پس از مراحل تکمیلی.

<SCR%00Ipt>%00Ipt>alert(\"xss\";alert('xss'));"></SCR%00Ipt>script>pt src=""xss.rocks/xss.js"></script>script>al
ert("xss"xss")>)>xss"xss")>)>'xss');">alert('xss'))'xss'))</script>">alert('xss');">b
ody{background:url("javascript:alert('xss')"RSnake says, 'xss')")}</script>style><a class=xss><C><![CDATA[<i
mg style="stylesheet" id=I></script>xml>hrefxss:expr/*");;}</style><a class=xss><!--#exec cmd="/bin/echo '=htt
p:///xss.rocks//xssxss.jsrocks/xss.jsxss.jsjs"> </SCR%00Ipt>alert(\"xss\"xss\"\"xss\"xss\"xss\")>)</script>alert('xss');
</style>script></style>script>'"--></a>!--#exec cmd="stylesheet" href/bin/echo '<script a=/xss//"get";>" data="&
{alert('xss')</script>"></script>'"----><meta http-equiv="http://xss.rocks/rocks//xssscriptlet.html"></style>scri
pt>alert('xss')'xss'))>;//"></xml></script>pt src=""http:///xss.jsrocks//xss.rocks//rocks/scriptlet.html">body{backgrou
nd:url("<SCR)';I")`>;}</style><a class=xss><

**شکل (۱۳)**: یک نمونه بردار حمله توسط گرامر استنتاج‌شده.