Lecture #16: Software Supply Chain Security #2

UCalgary ENSF619

Elements of Software Security

Instructor: Lorenzo De Carli (lorenzo.decarli@ucalgary.ca)

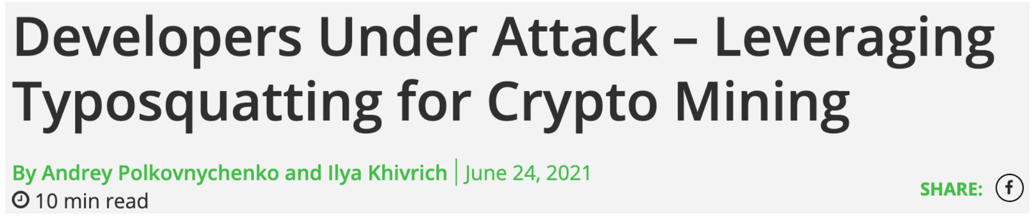
Based on slides by Shradha Neupane

The topic of this lecture

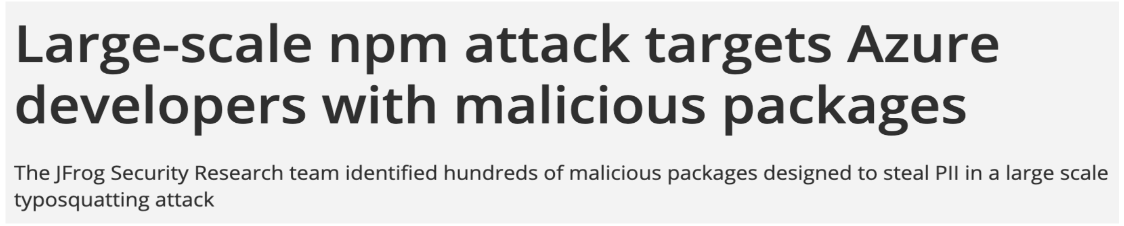
- **Typosquatting** and **package confusion**
- AKA the supply chain version of **phishing attacks**

Package Confusion

- Presence of a package that can be confused with some other package.
- Package confusion attack is the upload a package into ecosystem with **“sole purpose”** of confusing developers into downloading the wrong package.


Developers Under Attack – Leveraging Typosquatting for Crypto Mining

By Andrey Polkovnychenko and Ilya Khivrich | June 24, 2021
10 min read
SHARE:


Large-scale npm attack targets Azure developers with malicious packages

The JFrog Security Research team identified hundreds of malicious packages designed to steal PII in a large scale typosquatting attack

Example:

Confusing package: `mllearnlib`
Original package: `learnlib` and `mllearn`

Source: https://jfrog.com/blog/developers-under-attack-leveraging-typosquatting-for-crypto-mining/

More confusion in the wild

Threat Research | April 16, 2020

Mining for malicious Ruby gems

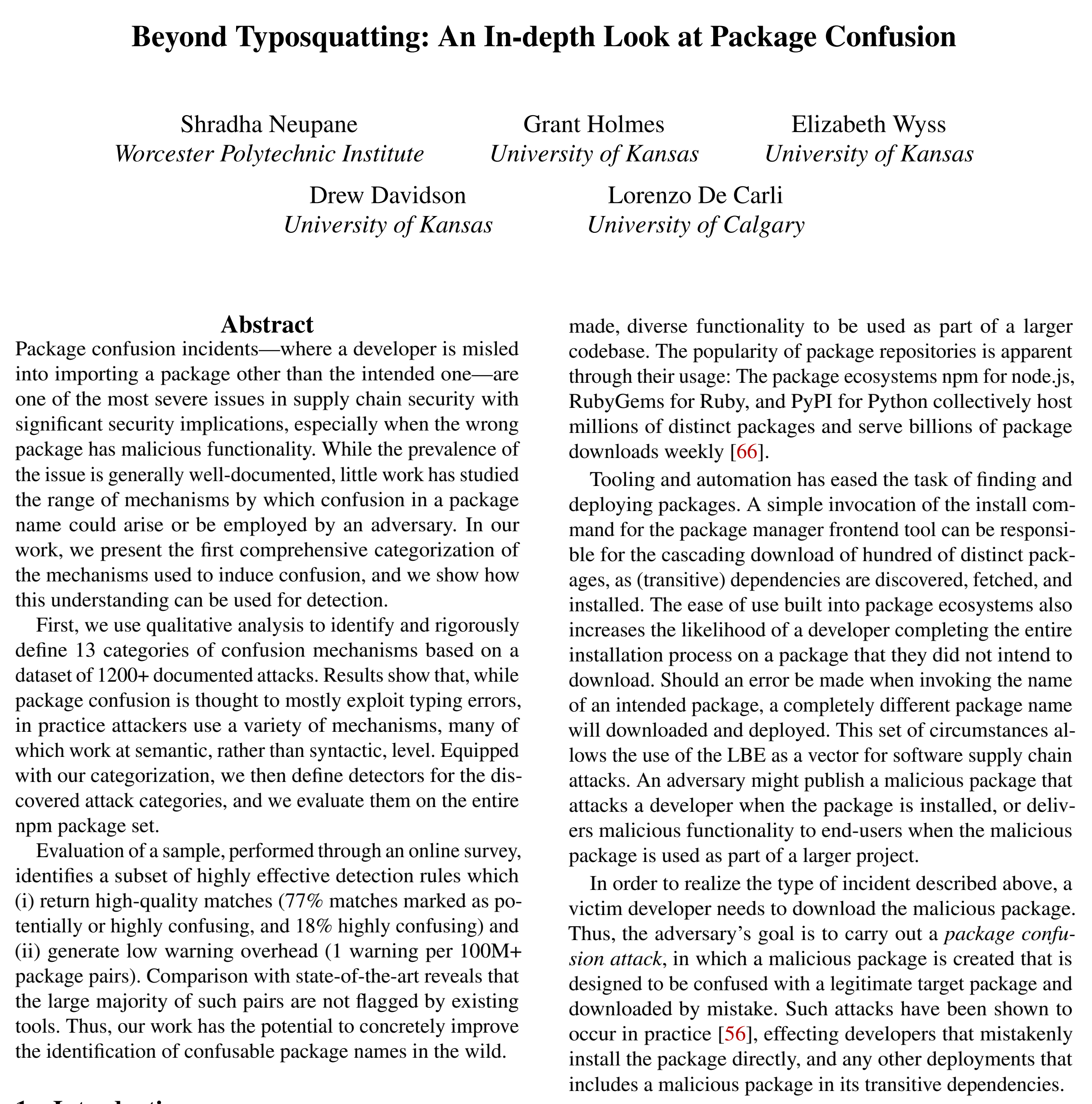
Typosquatting barrage on RubyGems software repository users

Established Package: **atlas_client**
Confuser Package: **atlas-client**

Malicious Behavior: **Crypto-theft** (redirect all potential crytocurreny transactions to their wallet address)

Source: https://www.reversinglabs.com/blog/mining-for-malicious-ruby-gems

Let’s talk about today’s paper



Beyond Typosquatting: An In-depth Look at Package Confusion

Shradha Neupane
Worcester Polytechnic Institute

Grant Holmes
University of Kansas

Elizabeth Wyss
University of Kansas

Drew Davidson
University of Kansas

Lorenzo De Carli
University of Calgary

Abstract

Package confusion incidents—where a developer is misled into importing a package other than the intended one—are one of the most severe issues in supply chain security with significant security implications, especially when the wrong package has malicious functionality. While the prevalence of the issue is generally well-documented, little work has studied the range of mechanisms by which confusion in a package name could arise or be employed by an adversary. In our work, we present the first comprehensive categorization of the mechanisms used to induce confusion, and we show how this understanding can be used for detection.

First, we use qualitative analysis to identify and rigorously define 13 categories of confusion mechanisms based on a dataset of 1200+ documented attacks. Results show that, while package confusion is thought to mostly exploit typing errors, in practice attackers use a variety of mechanisms, many of which work at semantic, rather than syntactic, level. Equipped with our categorization, we then define detectors for the discovered attack categories, and we evaluate them on the entire npm package set.

Evaluation of a sample, performed through an online survey, identifies a subset of highly effective detection rules which (i) return high-quality matches (77% matches marked as potentially or highly confusing, and 18% highly confusing) and (ii) generate low warning overhead (1 warning per 100M+ package pairs). Comparison with state-of-the-art reveals that the large majority of such pairs are not flagged by existing tools. Thus, our work has the potential to concretely improve the identification of confusable package names in the wild.

made, diverse functionality to be used as part of a larger codebase. The popularity of package repositories is apparent through their usage: The package ecosystems npm for node.js, RubyGems for Ruby, and PyPI for Python collectively host millions of distinct packages and serve billions of package downloads weekly [66].

Tooling and automation has eased the task of finding and deploying packages. A simple invocation of the install command for the package manager frontend tool can be responsible for the cascading download of hundred of distinct packages, as (transitive) dependencies are discovered, fetched, and installed. The ease of use built into package ecosystems also increases the likelihood of a developer completing the entire installation process on a package that they did not intend to download. Should an error be made when invoking the name of an intended package, a completely different package name will downloaded and deployed. This set of circumstances allows the use of the LBE as a vector for software supply chain attacks. An adversary might publish a malicious package that attacks a developer when the package is installed, or delivers malicious functionality to end-users when the malicious package is used as part of a larger project.

In order to realize the type of incident described above, a victim developer needs to download the malicious package. Thus, the adversary’s goal is to carry out a *package confusion attack*, in which a malicious package is created that is designed to be confused with a legitimate target package and downloaded by mistake. Such attacks have been shown to occur in practice [56], effecting developers that mistakenly install the package directly, and any other deployments that includes a malicious package in its transitive dependencies.

Typosquatting and Confusion

Our previous research [1] Typogard looks into package confusion through the context of lexical name changes.

Impact of Package confusion

- Confusing packages may contain maliciousness that directly affect the developer or application users
- Non-malicious packages degrade the quality of projects that they are accidently used in, by unintentionally introducing potentially unmaintained, vulnerable code ([2] Wyss et, al. , [3] Zimmermann et, al.)

Need for our research

- People have intuitive notion of how package confusion occurs, which is usually limited to typos.
- No notable research on how packages confusion occur in practice

Does package confusion beyond typo squatting exist, and can we detect it algorithmically?

[1] Matthew Taylor, Ruturaj Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. Defending Against Package Typosquatting. In NSS, 2020
[2] Elizabeth Wyss, Lorenzo De Carli, and Drew Davidson. What the fork?: Finding hidden code clones in npm. In IEEE/ACM ICSE, 2022.
[3] Markus Zimmermann, Cristian-Alexandru Staicu, and Michael Pradel. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In USENIX Security, 2019.

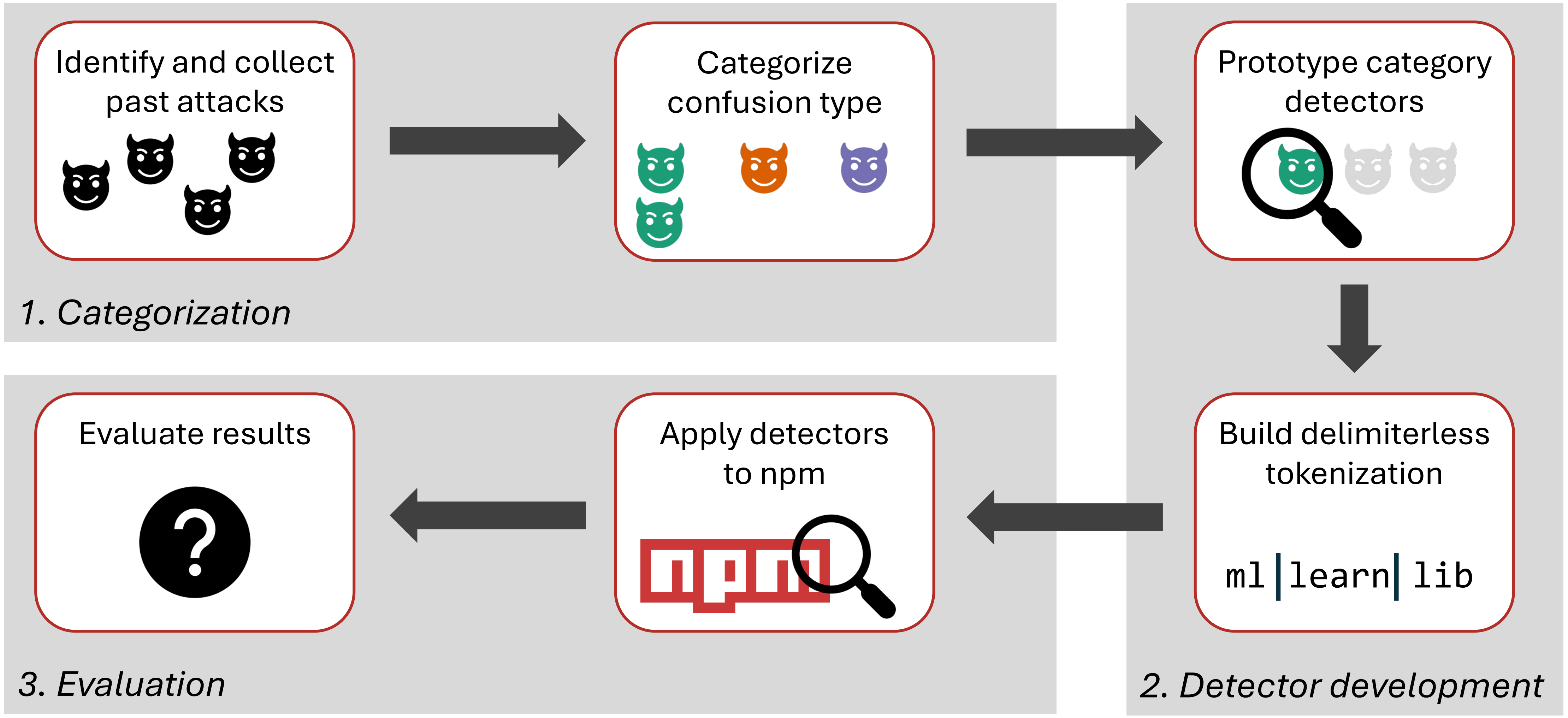
CONTRIBUTION

1. Package confusion occurs beyond typo squatting – identify 13 categories of confusability

2. Found potentially confusing packages in the wild and evaluate effectiveness of detection rules

3. Evaluate the security impact of package confusion

Research Outline


Identify and collect past attacks
Categorize confusion type
Prototype category detectors
Build delimiterless tokenization
ml|learn|lib
Apply detectors to npm
npm
Evaluate results
?
1. Categorization
2. Detector development
3. Evaluation

Collecting Attacks Results

Results of Collecting Historical Data

1232

Distinct attacks / confusing packages uploaded

7

Campaigns with 10 or more packages uploaded

Distribution Across Ecosystems

723

npm

462

python Package Index

48

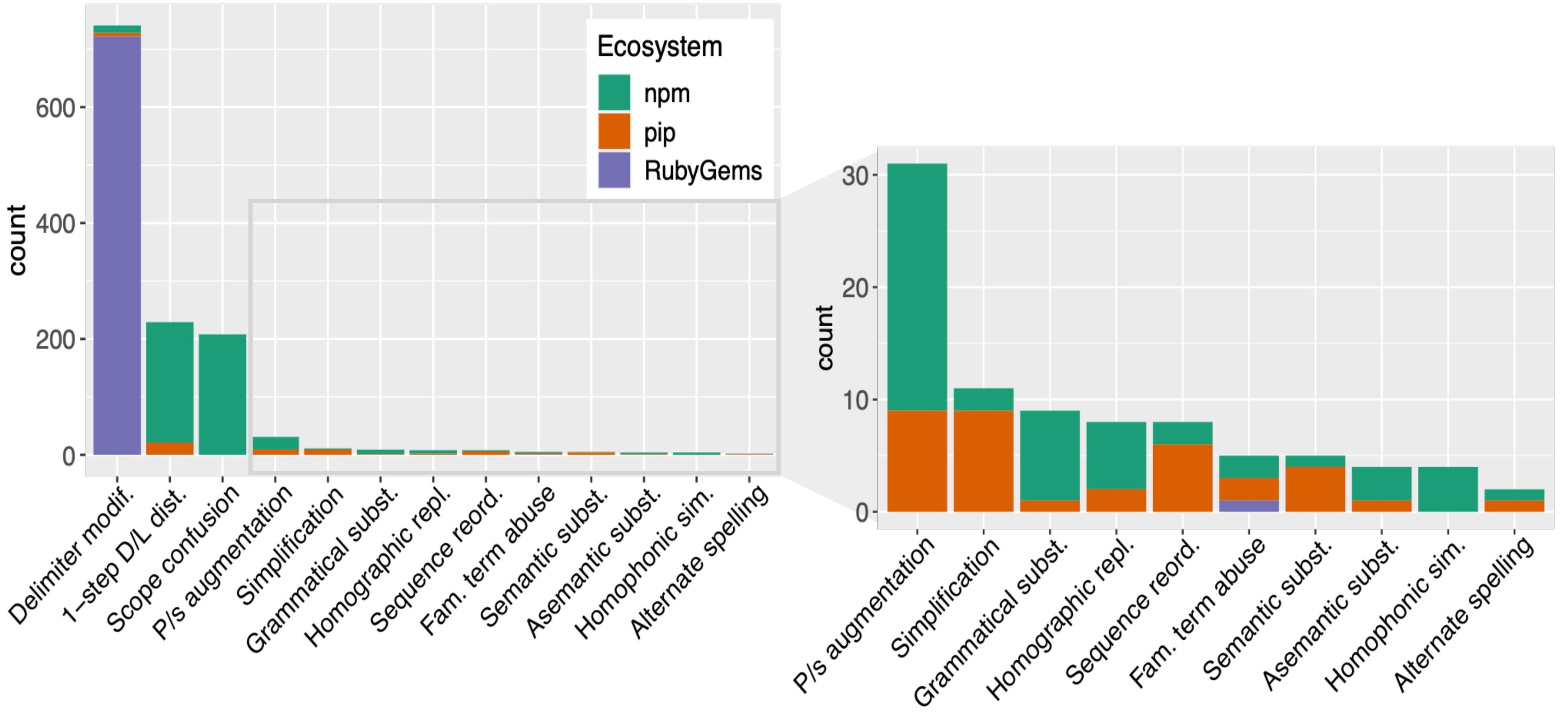
RubyGems

Confusability models and categorization

Thematic Analysis

After Round 4
1-step D/L distance
Alternate spelling
Asemantic substitution
Delimiter modification
Familiar term abuse
Grammatical substitution
Homographic replacement
Prefix/Suffix augmentation
Scope confusion
Semantic substitution
Sequence reordering
Simplification
Homophonic similarity

$\alpha = 0.96\ (0.94, 0.99)$



Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. Qualitative Research in Psychology, 3(2):77–101, 2006.

Why Thematic Analysis

Methodology for qualitative data analysis with application in the usable security domain:

- Emphasis on inductive reasoning which is necessary in our cases due to lack of prior knowledge on the nature of package confusion attacks.
- Mostly applied to expressive, long form text but used to provide methodological rigor to the process of deriving attack categories.

We apply a simplified version of themanic analysis:

- Five members of the research team review the same sample of 100 incidents and come up with an initial set of codes
- Subsequently all coders code the same set, using Krippendorff's alpha to measure inter-coder agreement.
- Repeat the step until $\alpha \geq 0.8$
- After reaching $\alpha \geq 0.8$ we split the remaining incidents among the coders

Delimiter-less Tokenization

- Number of detectors need transformation of package name into sequence of tokens

- Package names consist of technical jargons, which do not have valid English words but assume valid connotation in technical language. (json, db, py, js etc)

- Built a delimiter-less tokenization algorithm using the npm package names.

Example:

Confuser package : `mllearnlib`
Breaking down the package into tokens: [**ml, learn, lib**]

Target package: `mllearn`
Breaking down the package into tokens: [**ml, learn**]

Performance of Detection Rules

Rule	Precision	Recall	F1
P/S augmentation	0.95	0.70	0.81
Sequence reordering	0.88	0.88	0.88
Delimiter modification	1	0.97	0.98
Grammatical subst.	0.88	0.88	0.88
Scope confusion	1	0.90	0.95
Semantic subst.	1	0.4	0.57
Asemantic subst.	0.75	0.75	0.75
Homophonic sim.	0.07	0.75	0.13
Simplification	0.58	0.64	0.61
Alternate spelling	1	1	1
Homographic repl.	0.5	0.88	0.64

Table: Performance of detection rules

Detection Rule Optimization

Created Initial prototype and optimized it on each round

Goal: Maximize the chances of identifying actually confusable packages, at the cost of missing some attacks.

Difficult due to significantly imbalanced samples.

EVALUATION OF DETECTION RULES

RQ1: How many potential instances of package confusion exist in the npm ecosystem?

Methodology
Apply the detection rules to the whole of npm.
Infeasible because analysis of $(1.7e6)^2$ npm package pairs would be required

Popularity threshold
15,000 weekly downloads

Popular package: Established Original Packages

Unpopular packages: Confuser Packages

Total:
1, 727, 553 × 24871

RQ1: How many potential instances of package confusion exist in the npm ecosystem?

Methodology
Apply the detection rules to the whole of npm.
Infeasible because analysis of $(1.7e6)^2$ npm package pairs would be required

Popularity threshold
15,000 weekly downloads

Popular package: Established Original Packages

Unpopular packages: Confuser Packages

Total:
$1, 727, 553 \times 24871$

Results

Rule	#Instance
P/S augmentation	143864
Asemantic subst.	139160
Simplification	27743
Homophonic sim.	24735
Semantic subst.	9610
Delimiter modification	7183
Scope confusion	4247
Grammatical subst.	2461
Homographic repl.	2393
Sequence reord.	1734
Alternate spelling	21

Table: Matches in npm for each category

Results

- **~ 360,000** package pairs detected as confusing
- Analysis took **0.22ms/pair**
- **2799** pairs matching multiple categories
- *Homophonic similarity & Prefix/ suffix augmentation, Delimiter modification & Sequence reordering*, and *Delimiter modification & Grammatical substitution*

RQ2: What is the confusability of identified matches?

Online survey of to perceive confusability of randomly selected package pairs.

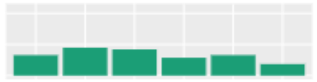
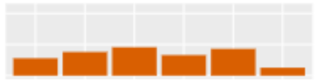
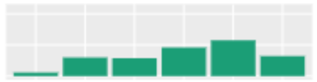
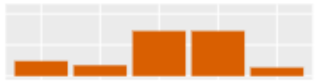
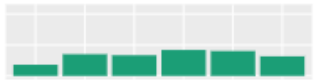
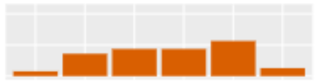
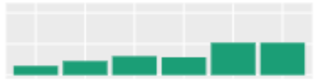
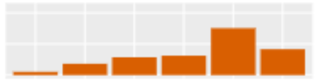
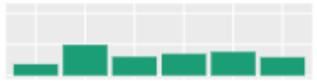
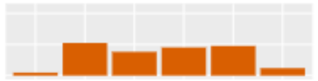
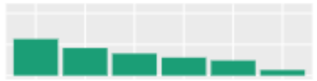
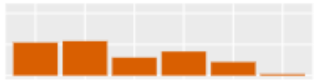
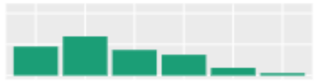
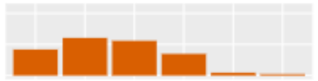
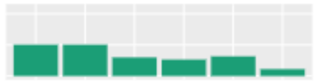
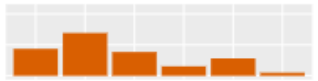
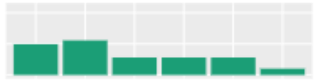
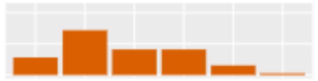
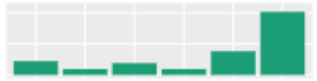
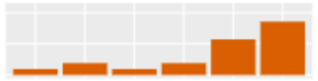
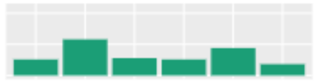
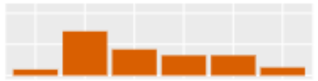
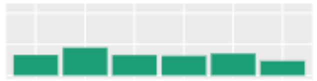
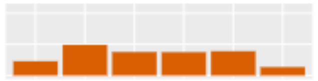
On a scale of 1 to 6, how likely are you to misremember or mistype the package in column V with package column P?

Sampling: 50 questions from a pool of 100 package pairs from each category + 100 control samples

Recruitment: Email recruiting and snowball sampling of student developers

Goal: Determine which rules can return reliable matches.

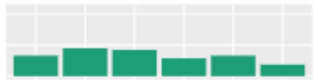
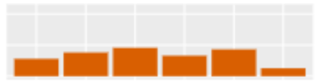
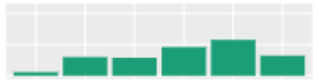
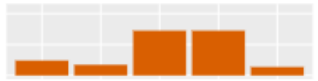
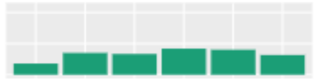
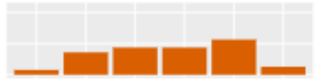
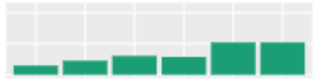
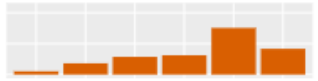
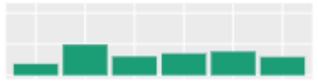
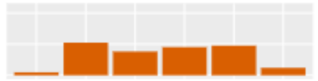
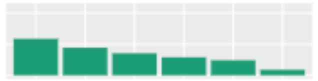
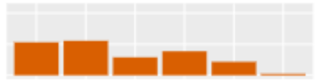
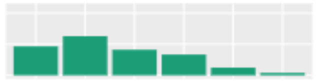
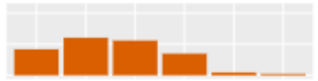
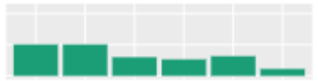
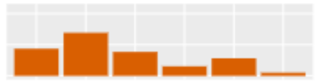
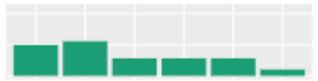
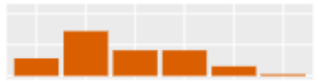
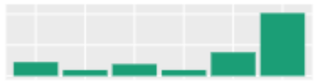
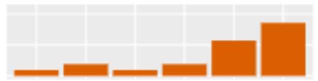
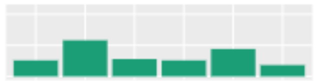
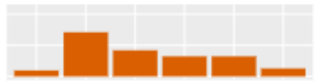
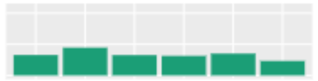
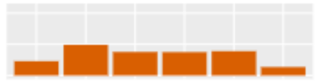
Results

Rule	Rating Distribution	Median Distribution	n samples	%(2+r≥ 4)	%(3r≥ 5)
P/s augmentation			79	44%	2.5%
Sequence reord.			58	79%	10%
Delimiter modif.			78	56%	7.7%
Grammatical subst.			77	74%	18%
Scope confusion			84	52%	4.8%
Semantic subst.			83	31%	0.0%
Asemantic subst.			86	21%	0.0%
Homophonic sim.			78	24%	3.8%
Simplification			78	29%	1.3%
Alternate spelling			21	81%	38%
Homographic repl.			62	39%	6.5%
Overall			**62**	**45%**	**6.1%**

Results

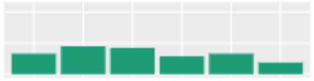
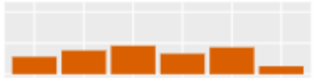
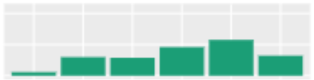
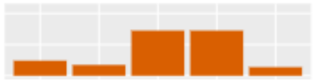
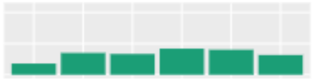
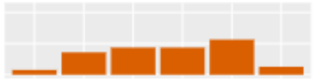
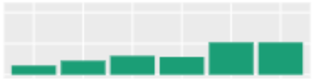
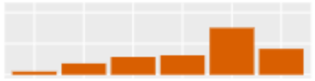
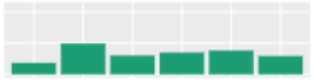
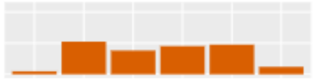
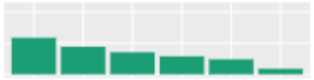
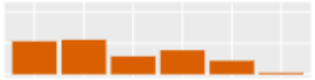
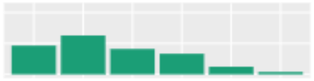
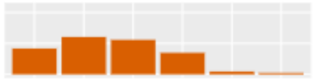
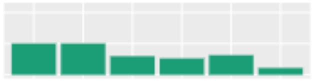
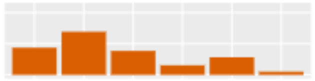
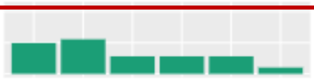
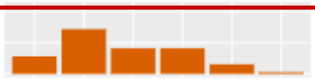
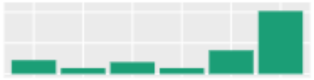
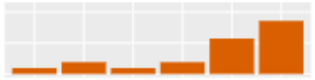
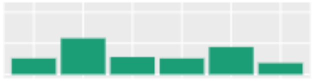
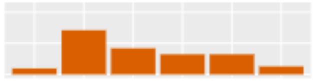
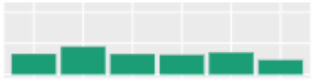
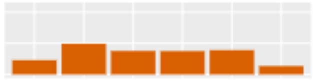
>10% with “highly confusing” criterion

>70% with “potentially confusing” criterion

Rule	Rating Distribution	Median Distribution	n samples	%(2+r≥ 4)	%(3r≥ 5)
P/s augmentation			79	44%	2.5%
Sequence reord.			58	79%	10%
Delimiter modif.			78	56%	7.7%
Grammatical subst.			77	74%	18%
Scope confusion			84	52%	4.8%
Semantic subst.			83	31%	0.0%
Asemantic subst.			86	21%	0.0%
Homophonic sim.			78	24%	3.8%
Simplification			78	29%	1.3%
Alternate spelling			21	81%	38%
Homographic repl.			62	39%	6.5%
Overall			**62**	**45%**	**6.1%**

Results

< 25% with "potentially confusing" criterion

Rule	Rating Distribution	Median Distribution	n samples	%(2+r≥ 4)	%(3r≥ 5)
P/s augmentation			79	44%	2.5%
Sequence reord.			58	79%	10%
Delimiter modif.			78	56%	7.7%
Grammatical subst.			77	74%	18%
Scope confusion			84	52%	4.8%
Semantic subst.			83	31%	0.0%
Asemantic subst.			86	21%	0.0%
Homophonic sim.			78	24%	3.8%
Simplification			78	29%	1.3%
Alternate spelling			21	81%	38%
Homographic repl.			62	39%	6.5%
Overall			**62**	**45%**	**6.1%**

RQ3: What is the security impact of identified confusing packages?

Goal:
Assess density of malicious packages.

Problem:
No ground truth.

Solution:
Analysis of existing vulnerability database (lower bound)

Results:
Packages flagged by our rules are **3 times more** likely to be malicious than control.

Details:
Sample: Unique packages = 210,741, Malicious packages found: 168 (0.079%)
Control: Unique packages = 150,000, Malicious packages found: 39 (0.026%)

Malicious Behavior in Confusing Packages

Attack Category	#pkgs
Stealing	70
Backdoor	9
Sabotage	2
Cryptojacking	2
Virus	1
Maladvertising	2
PoC	1
Cryptotheft	33
Downloader	1
Confusion	2
Unknown	45

Table: Malicious Packages type distribution in results and our known attack set

- We categorized the flagged malicious packages as per [1] Duan et al.
- Added 3 new categories: Crypto Theft, Downloader, Confusion
- Could not be verify malicious behavior in some due to removal of packages from ecosystems

Conclusions

Package confusion is a credible threat and our categorization help to specify how the attacks may occur.

Our categories provide a new dimension to package confusion beyond typosquatting

While some detection rules can still be improved with more data, some of them have low enough warning

Aside: typosquatting in the web

Typosquatting on the web?

- **Approach:** register domains that are textually close to those of well-known websites
- Wait for users to **make typos**, **capture traffic** from those users
- Can be used for **various illicit operations**

Typosquatting on the web /2

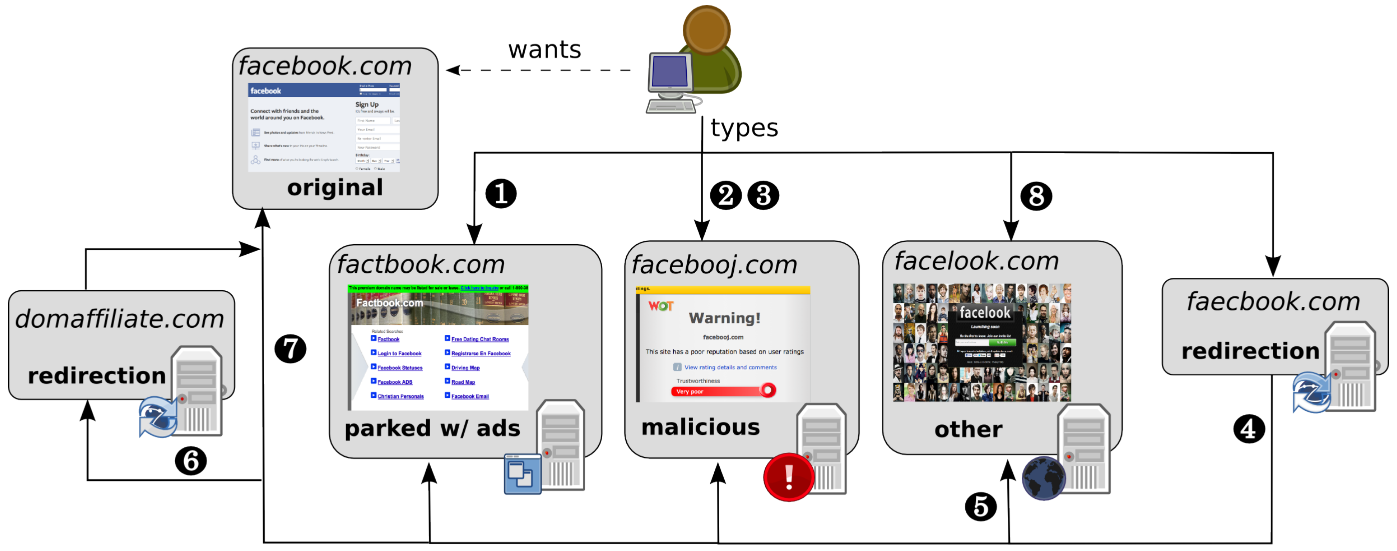


Figure 1: The typosquatting ecosystem with various monetization techniques.

(From Szurdi et al., "The Long "Taile" of Typosquatting Domain Names", USENIX 2014

That’s all for today!